UNIVERSITY OF OSLO
Department of Informatics

# ERG Tokenization and Lexical Categorization

A sequence labeling approach

Master's thesis

Murhaf Fares

May 16, 2013

## Acknowledgements

I would like to express my heartfelt gratitude to my advisor Stephan Oepen. Working with Stephan was a real privilege; with his comprehensive knowledge, unrelenting enthusiasm and dedication, high scientific standards and genuine friendliness, he set an example to me.

I owe a special debt to the nice friends, Arne Skjærholt and Emanuele Lapponi, for generously sharing their time and knowledge, for discussions out of which many ideas in this thesis have grown, and for caring conversations.

To Yi Zhang, I am grateful for helping to develop some ideas in this thesis and for hosting me at DFKI Saarbrücken. My gratitude shall be extended to Rebecca Dridan for practical help in tokenization.

To my wonderful family, thank you, for being supportive and for always believing in me. My father, *muchas gracias*, for reminding me in every possible opportunity that knowledge is the most important thing in one's life. My mother, for the words she said when I told her I want to pursue my study in Oslo and for unwavering care even from afar. Anas and Munes, my brothers, for their encouragement, friendship, and boundless support.

And to Zeina, my special gratitude, for being the reason why I can still smile and carry on.

*In memory of Ghiyath, Mazhar, Mustafa, Ayham, Bassel, Emad and thousands of Syrians who sacrificed their lives for us to have better ones.*

*To Syria, my love and my heartache, with hope for a brighter future…*

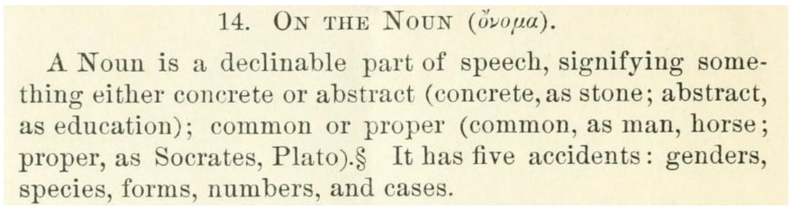
# Contents

# List of Tables

# List of Figures

Chapter 1

# Introduction


14. On the Noun (ὄνομα).

A Noun is a declinable part of speech, signifying something either concrete or abstract (concrete, as stone; abstract, as education); common or proper (common, as man, horse; proper, as Socrates, Plato).§ It has five accidents: genders, species, forms, numbers, and cases.

Translated from Greek by Davidson (1874)

In the olden days, circa 100 BC, Dionysius Thrax wrote a grammatical sketch of Greek, named *Art of Grammar*.[1] He distinguished between different *categories* of word based on inflectional affixes, e.g. *nouns* inflect for case while *verbs* inflect for tense and person. Ever since then, linguistic descriptions would always contain some definition of word categories or lexical categories. Today, lexical categories play no less important role in computational linguistics. Most syntactic parsers (systems that identify grammatical relationships between words and phrases in sentences) use lexical categories in pre- or post-processing steps. The information in lexical categories can also be exploited in other natural language processing (NLP) tasks such as noun phrase chunking (dividing text into non-overlapping noun phrases), semantic role labeling (detecting and classifying semantic arguments associated with the verb) and machine translation (translating text from one natural language to another).

In this thesis, we study *lexical categorization* (assigning lexical categories to words) in the realm of *English Resource Grammar*—a broad-coverage computational grammar for English (ERG; Flickinger, 2000). The aforementioned definition of lexical categorization assumes some conception of ‘words’, hence, we take a step back to define what is considered to be a word.

[1] In fact the exact authorship of the grammar is disputable (Jurafsky & Martin, 2008, p. 157).

Linguists might define up to eight kinds of word, most notably: (a) the *orthographic word* which is, in alphabetic writing systems,[2] a sequence of letters with spaces around it; and (b) the *lexical word* which consists of one or more orthographic words but makes up one basic element in the language's lexicon, e.g. multi-word expressions such as 'San Francisco' and 'all of a sudden'. The ERG conception of word blends orthographic with lexical words through recognizing some types of multi-word expressions. Hence, the whitespace is neither sufficient nor necessary to identify ERG words (tokens). By the time of this study, to the best of our knowledge, ERG tokenization (splitting a stream of text into words) hasn't been investigated. Therefore, in this thesis we study *tokenization* prior to *lexical categorization*, because assigning ERG lexical categories presupposes identifying words which comply with the ERG notion of word. Our overarching aim, however, is to improve ERG syntactic parsing.

ERG lexical categories encode rich syntactic information in addition to the lexical and morphological information. Even ERG tokens, arguably, encode some aspects of syntax through recognizing multi-word expressions. Hence, we can exploit the information in ERG lexical categories, and to a lesser degree in ERG tokens, to prune the parser search space.

In technical terms, this thesis investigates the use of sequence labeling (assigning labels or categories to each element within a sequence) to model the tasks of tokenization and lexical categorization in the realm of English Resource Grammar (ERG). We use Conditional Random Fields (an instance of sequence labeling models; Lafferty et al., 2001) to train statistical models for tokenization and lexical categorization. Then, we study the impact of such models on ERG syntactic parsing in terms of efficiency, accuracy and coverage.

## 1.1 Motivation and Research Questions

The English Resource Grammar (ERG) has been continuously developed for the last 20 years achieving higher grammatical coverage and resilience to domain variation, i.e. in 2002, the hand-crafted ERG lexicon contained some 8,000 lexical entries, whereas in the 2012 version of ERG there are 38,500 lexical entries. This superior linguistic precision, however, comes at a price of expensive parsing times. The need for more efficient ERG parsing is growing with the sophistication of ERG. Hence, the aim of this thesis is to improve the efficiency of ERG parsing through the use of ERG lexical categories.

In order to study ERG lexical categorization we start by considering related works. Dridan (2009) and Ytrestøl (2012) are of the most recent and comprehensive studies on improving ERG-based syntactic parsing. As in every research project, they both open the door for new questions, leave some questions unanswered and make assumptions to facilitate their work.

[2]In logographic or syllabic writing systems, such as Chinese, there is no explicit marker for word boundaries.

First, neither Dridan (2009) nor Ytrestøl (2012) handle ERG tokenization; while the latter assumes gold standard ERG tokens, the former does not use ERG-conformant tokens. In fact, to date, parsers working with the ERG either operate off an ambiguous full token lattice (Adolphs et al., 2008) or assume idealized gold-standard ERG tokenization (Zhang & Krieger, 2011; Ytrestøl, 2011; Evensberget, 2012). Therefore, in this thesis we seek to determine:

**(a)** How to apply sequence labeling techniques to approach tokenization.

**(b)** To what degree CRF sequence labeling in isolation scales to the ERG conception of the tokenization task.

**(c)** What levels of accuracy CRFs attain on the more classic tokenization scheme that of the Penn Treebank (PTB; Marcus et al., 1993)—specially that PTB tokenization plays an important role in ERG parsing.

Second, Dridan (2009) approaches lexical categorization through training Hidden Markov Models (HMM) and Maximum Entropy (MaxEnt) learners, and Ytrestøl (2012) experiments with Support Vector Machine (SVM) and MaxEnt, in all cases using off-the-shelf tools. In this thesis, we investigate the utility of CRFs for learning ERG lexical categorization with respect to:

**(d)** What features can be used to model ERG lexical categories.

**(e)** What levels of accuracy can be attained across several degrees of linguistic granularity in lexical categories.

**(f)** How to efficiently train and decode CRFs with the very fine-grained set of ERG lexical categories.

Third, Dridan (2009) reports that using ERG lexical categories to restrict the ERG parser search space leads to relatively modest improvements in parsing efficiency. However, ERG is regularly updated, hence new versions of the grammar and more gold-standard training data has been released since Dridan (2009) carried out her experiments. Therefore, an update on the study of Dridan (2009) is needed. Here, we seek to define:

**(g)** The best setup to integrate our CRF lexical categorization and tokenization models with ERG syntactic parsing.

**(h)** What levels of parsing efficiency, coverage and accuracy we can achieve using our lexical categorization and tokenization models.

**(i)** Whether or not the degree of linguistic granularity in lexical categories significantly affect parsing efficiency.

## 1.2 Results

**Tokenization** We present innovative work on ERG tokenization that is accurate enough to serve as a front-end for the ERG parser. Our tokenization results improve over previously reported PTB 'state-of-the-art' tokenization and suggest an advantage of supervised machine learning over finite-state techniques, in terms of accuracy, adaptability to variable schemes, and resilience to domain and genre variation. Our results on tokenization are also published in Fares et al. (2013).

**Lexical Categorization** We evaluate the utility of CRFs to model ERG lexical categories with a detailed feature ablation study. We also show that the heavy computational cost of training and decoding CRFs can be alleviated through a divide-and-conquer strategy. Our results also suggest that training CRFs on rather large amounts of automatically annotated data helps improve lexical categorization accuracy, especially on out-of-domain data.

**Integration** We empirically prove that our ERG tokenization model can used to provide the ERG parser with token boundaries leading to small improvements in its efficiency without losses in accuracy or coverage. Further, we show that despite the imperfectness of our ERG lexical categorization models in isolation, they can substantially reduce ERG parsing time and improve its coverage and accuracy all together.

## 1.3 Thesis Outline

**Chapter 2: Background** This chapter presents the concepts, techniques and resources we rely upon in this thesis. We introduce the concepts of tokenization and lexical categorization in general. We present the parsing pipeline of the English Resource Grammar (ERG), which shows the need for better ERG tokenization and lexical categorization. Then, we explain Conditional Random Fields, the primary sequence labeling technique used throughout this thesis, and linguistic resources used to train and test CRFs. We review previous research studies related to our work.

**Chapter 3: Tokenization** This chapter presents the use of CRFs to model the ERG and PTB tokenization conventions using generally available, 'standard' resources. It provide empirical results on the utility of various types of features and sensitivity to genre and domain variation, as well as in-depth error analyses for the most pertinent experiments.

**Chapter 4: Lexical Categorization** This chapter investigates the use of CRFs to model ERG lexical categorization. Most notably, it details our experiments with two main degrees of lexical category granularities, viz. *major syntactic categories* and *lexical types*. It also presents our *specified lexical types* approach to avoid the computational cost of modeling `ERG` lexical categories with `CRF`s.

**Chapter 5: Integration** In this chapter, we review approaches to integrate our tokenization and lexical categorization models into the syntactic parsing task. Then, we empirically decide to what degree our tokenization and lexical categorization models can improve parsing efficiency, coverage and accuracy.

**Chapter 6: Conclusion** This chapter concludes the outcomes of our work and considers possible improvements and remaining questions for future work.

# Chapter 2

# Background

Although some familiarity with topics in machine learning and natural language processing is presupposed, in this chapter we survey briefly some of the main background assumptions of our project. We describe the concepts of tokenization and lexical categorization. Then, we introduce the parsing pipeline of the English Resource Grammar which constitutes our general framework for tokenization and lexical categorization. We lay down the definition of Conditional Random Fields, our primary machine learning approach, through general definitions of sequence labeling and Hidden Markov Models. Then, we introduce the linguistic resources on which we train and test our sequence labeling models for both tokenization and lexical categorization. Finally, we review previous and related research on tokenization and lexical categorization.

## 2.1 Tokenization

Tokenization is the task of mapping a sequence of characters to a sequence of *tokens*. But, what is a token? Manning and Schütze (1999, p. 124) define a token as "either a *word* or something else like a number or a punctuation mark". What is a word, then? There is no definitive answer for this question. In fact, linguists suggest defining 'words' at various levels, e.g. McArthur (1992, p. 1120–1121) defines eight kinds of word, such as phonological words and grammatical words. Furthermore, as we shall see in Chapter 3, the definition of the token might vary with the grammar formalism and the downstream natural language processing (NLP) application.

In this project, we study two interpretations of tokenization, namely the Penn Treebank (`PTB`) and the English Resource Grammar (`ERG`) tokenization schemes. Example 2.1 shows how a given sentence would be tokenized with respect to the `PTB` and the `ERG` tokenization conventions, noting significant differences between the two schemes.

All aspects of `PTB` and `ERG` tokenization conventions are described in § 3.2, but for now suffice to say, one striking difference between `PTB` and `ERG`

**Example 2.1**
*Untokenized raw sentence:*
For example, this isn't a well-formed example.
*PTB-style tokenization:*
`¦For¦ ¦example¦ ¦,¦ ¦this¦ ¦is¦ ¦n't¦ ¦a¦ ¦well-formed¦ ¦example¦ ¦.¦`
*ERG-style tokenization:*
`¦For example,¦ ¦this¦ ¦isn't¦ ¦a¦ ¦well-¦ ¦formed¦ ¦example.¦`

tokenization schemes is the treatment of punctuation marks; the PTB splits them from adjacent tokens, whereas the ERG does not. Another significant difference is that the ERG recognizes some classes of multi-word expressions (MWEs), such as '`for example`' and '`New York`'.

One of the 'classic' tokenization difficulties is punctuation ambiguity; for example, the period is a highly ambiguous punctuation mark because it can serve as a full-stop, a part of an abbreviation, or even both[1]. Parentheses and commas typically form individual tokens, but they can be part of names and numeric expressions, e.g. in '`Ca(2+)`' or '`390,926`'. Handling contracted verb forms and the Saxon genitive of nouns is also problematic because there is no generally agreed-upon standard on how to tokenize them. But corner cases in tokenization are by no means restricted to the aforementioned ones; with different tokenization conventions, abstractly rather similar problems emerge.

What we just presented is just a general, high-level description of tokenization, which suffices to understand the content of this chapter. Chapter 3, however, presents our work on tokenization down to the last detail.

## 2.2 Lexical Categorization

Lexical categorization is the task of assigning *lexical categories* to a given sequence of words. Lexical categories (also called parts of speech or word classes) define classes of words that have similar linguistic behavior. This linguistic behavior, however, can be defined with respect to several inextricable criteria (Hopper & Thompson, 1984) such as:

1. Morphological properties: the set of inflectional morphemes that words of a particular category combine with.

[1]When abbreviations appear at the end of the sentence, then only one period occurs and it serves two functions, viz. an abbreviation's period and a full stop. Within morphology, this phenomenon is referred to as 'haplology'; when two consecutive identical syllables occur one of them is eliminated (Manning & Schütze, 1999, p. 125).

2. Syntactic properties: the context where words of a particular category occur with respect to the preceding and following words and categories. Consequently, words within one lexico-syntactic category can be swapped while preserving grammaticality (e.g. swapping verbs in the following sentence I *read* the book vs. I *eat* the book).

3. Semantic properties: the set of meanings that words of a particular category convey, though in most lexical classes there is no complete semantic coherence (Brown, 1957).

Although the definition of lexical categories might seem intuitively clear, in practice the boundaries between lexical categories are very blurred. On the one hand, there are several views through which word classes can be defined, or as Bloomfield (1935) put it:

*"A system of parts of speech in a language like English cannot be set up in any fully satisfactory way: our list of parts of speech will depend upon which functions we take to be the most important."* (Bloomfield, 1935, p. 269)

On the other hand, under each of the aforementioned criteria (morphological, syntactic and semantic) there are degrees of how much detail is encoded in the lexical categories – in other words, how fine-grained or coarse-grained they are, e.g. we can distinguish between main and auxiliary verbs, and on a finer degree of granularity we distinguish between the types of auxiliary verbs such as modal auxiliary and perfective auxiliary verbs.

Furthermore, lexical categories tend to be formalism-dependent, for example in linguistically precise grammar formalisms (such as Head-driven Phrase Structure Grammar; HPSG) the lexical categories may encode subcategorization information and possible syntactic operations (e.g. passivization), while in a context-free grammar (CFG) such information is typically not included. What is more, in statistical NLP, we use annotated language resources to learn the lexical categories, thus the division of lexical categories depends on the language resource in use; the Brown Corpus tagset, for instance, consists of 87 tags, while in the Penn Treebank (PTB) there are 45 tags.

Even though the definition of lexical categories is a vexed one, the task of lexical categorization is well-established and has been investigated rather early in the NLP field (Greene & Rubin, 1971; Marshall, 1983; Schmid, 1994). Moreover, one can distinguish two general subdivisions of lexical categories in NLP, namely *part-of-speech* (PoS) tags and what is often called *supertags*. While PoS tags often are defined independent of a specific framework[2], supertags seem to be special to some grammar formalisms. In our view, however, the boundary between PoS tags and supertags is not completely definable; in full generality, lexical categorization is a long continuum, where lexical categories vary on

[2]Even more, Petrov et al. (2012) define a language-independent universal set of PoS tags

**Example 2.2**
*More*$_{JJR}$ *than*$_{IN}$ *a*$_{DT}$ *few*$_{JJ}$ *CEOs*$_{NNS}$ *say*$_{VBP}$ *the*$_{DT}$ *red-carpet*$_{JJ}$ *treatment*$_{NN}$ *tempts*$_{VBZ}$ *them*$_{PRP}$ *to*$_{TO}$ *return*$_{VB}$ *to*$_{TO}$ *a*$_{DT}$ *heartland*$_{NN}$ *city*$_{NN}$ *for*$_{IN}$ *future*$_{JJ}$ *meetings*$_{NNS}$ .$_{.}$

`JJR: Adjective, comparative; IN: Preposition or subordinating conjunction; DT: Determiner; JJ: Adjective; NNS: Noun, plural`

two dimensions. First, the type of information encoded within the lexical categories (morphological, syntactic and semantic). Second, the granularity of the lexical categories, e.g. main vs. modal verbs distinction. We refer to both PoS tags and supertags as 'lexical categories', but in this chapter we shall keep the distinction between them in order to better explain the concept of supertags.

### 2.2.1 Part-of-Speech Tags

Anyone who has used dictionaries must have encountered PoS tags; if you pick up *any* dictionary and look up *any* word, you will notice that it is classified as a 'verb', a 'noun', an 'adjective', an 'adverb' or some other 'part of speech'.

Part-of-speech tags follow the definition of lexical categories we provided in the previous section. To clarify the idea of PoS tags, Example 2.2 shows a PoS tagged sentence from the `PTB`.

As shown in Example 2.2, the `PTB` tagset[3] distinguishes between singular and plural nouns (`NN` vs. `NNS`) and between verb tenses, e.g. past, past participle, and present participle (gerund) (`VBD`, `VBN` and `VBG` respectively). However, it lacks the distinction between '`to`' as a preposition, an infinitival marker and a complementizer.

Automatic PoS tagging has been exhaustively studied in the field of NLP and many rule-based and machine learning models have been proposed to approach it. The Penn Treebank (`PTB`; introduced in § 2.6) plays an important role in the development and evaluation of many PoS taggers. The best-performing PoS taggers are based on models such as Hidden-Markov Model (Brants, 2000), Maximum Entropy Model (Denis & Sagot, 2009; Toutanova et al., 2003), Support Vector Machines (Søgaard, 2010), Decision Trees (Søgaard, 2010), Perceptrons (Collins, 2002; Spoustová et al., 2009); where Toutanova et al. (2003) and Søgaard (2010) achieve the highest accuracies, 97.32% and 97.50%, respectively, on the last three sections of the `PTB`. In § 2.4 we will refer to some of these techniques when describing our approach to tokenization and lexical categorization.

[3]The tagset is the set of possible PoS tags.

### 2.2.2 Supertags

Supertags are linguistically rich lexical categories with more descriptive power than PoS tags.

The idea of supertags as lexical descriptions originates back to Joshi and Srinivas (1994) who proposed enhancing the concept of PoS tags to include richer syntactic and semantic information. Given the lexicalized grammar they were using, Lexicalized Tree Adjoining Grammar (LTAG; Schabes & Joshi, 1990), it was possible to use the LTAG elementary trees to assign each word more information than that of PoS tags. Hence, they created super PoS tags (supertags) that encode long-distance dependencies in addition to the morphological and localized information PoS tags typically include.

Joshi and Srinivas (1994) describe supertagging as “almost parsing”; they believe that the large amount of information provided by supertags leaves very little and light work for the parser. In other words, the supertags impose complex constraints on the parse search space which makes parsing a ‘trivial’ task.

More recently, other lexicalized grammar formalisms adopted the approach of supertagging (Clark, 2002; Ninomiya et al., 2006), but unlike PoS tags, there is no ‘standard’ definition for supertags, rather the set of supertags is entirely dependent on the grammar formalism. Dridan (2009) reports that the number of supertags across grammar formalisms ranges from 300 supertags in LTAG (Joshi & Srinivas, 1994) to 8000 supertags in some versions of HPSG (Toutanova et al., 2002).

As such, supertags encode more information than PoS tags, and the number of supertags is noticeably larger than that of PoS tags. However, the problem of supertagging is still a sequence classification one (with a relatively large number of classes), and so the models used for PoS tagging can also be used for supertagging. However, the accuracy of the supertagging models would be much lower than that of PoS tagging models which land around 97% (cf. previous section). In fact, a supertagging accuracy of 97% is not easily attainable, if at all, with a single-tag supertagger.

Finally, to illustrate the difference between supertags and PoS tags, Example 2.3 shows the same sentence of Example 2.2 but with ERG lexical types (supertags) assigned to its words. We will come back to explaining the details of the ERG lexical types in Chapter 4.

### 2.2.3 Lexical Categories for Parsing

Lexical categorization can be seen as a by-product of syntactic analysis. In other words, lexical categories are dependent on the syntactic analysis as they are derived from the syntactic structure of the sentence. Accordingly, some may ask: if lexical categorization is *conceptually* a result of syntactic parsing, how would we use it to help solve the problem of parsing? It is true that using lexical

**Example 2.3**

*More*$_{\textit{aj\_pp\_i-more\_le}}$ *than*$_{\textit{p\_np\_ptcl-ngap\_le}}$ *a few*$_{\textit{aj\_-\_i-svrl\_le}}$ *CEOs*$_{\textit{n\_-\_mc\_le}}$ *say*$_{\textit{v\_pp*-cp\_fin-imp\_le}}$ *the*$_{\textit{d\_-\_the\_le}}$ *red-*$_{\textit{aj\_-\_i-color-er\_le}}$ *carpet*$_{\textit{n\_-\_mc\_le}}$ *treatment*$_{\textit{n\_pp\_mc-of\_le}}$ *tempts*$_{\textit{v\_np-vp\_oeq\_le}}$ *them*$_{\textit{n\_-\_pr-them\_le}}$ *to*$_{\textit{cm\_vp\_to\_le}}$ *return*$_{\textit{v\_pp*\_dir\_le}}$ *to*$_{\textit{p\_np\_i-nm-no-tm\_le}}$ *a*$_{\textit{d\_-\_sg-nmd\_le}}$ *heartland*$_{\textit{n\_-\_mc\_le}}$ *city*$_{\textit{n\_-\_c\_le}}$ *for*$_{\textit{p\_np\_i\_le}}$ *future*$_{\textit{aj\_-\_i-att\_le}}$ *meetings.*$_{\textit{n\_pp\_c-nt-of\_le}}$

```
aj_pp_i-more_le:  Adjective, selects for prepositional phrase;
p_np_ptcl-ngap_le:  Preposition, selects for noun phrase
```

categorization to improve parsing may seem counter-intuitive, but empirically, it is possible to assign lexical categories using statistical morphological and local syntactic information. For example, X. Chen and Kit (2011) show that the Stanford tagger (Toutanova et al., 2003) outperforms the Berkeley parser (Petrov & Klein, 2007) when the latter is evaluated on the correctness of the PoS tags it assigns. Thus, it does make sense, especially in the empiricist world, to use lexical categorization in order to improve parsing.

Driven by the high performance of lexical categorization, especially PoS tagging, many studies have suggested integrating lexical categorization with syntactic parsing. Lexical categories can be used in pre- or post-processing models of parsing in order to improve parsing accuracy and efficiency. Additionally, taggers can make parser domain adaptation easier, especially with the relatively low cost of retraining taggers compared to that of retraining parsers. Creating training data for taggers is cheaper than doing so for parsers for two main reasons: (1) the training data for taggers consist of flat sequences of tags, while for parsers it is hierarchical treebanks (2) one can train taggers on the output of parsers, i.e. obtain new training data for the taggers merely by running a parser.

We distinguish between two approaches for integrating lexical categorization with parsers:

1. Soft-constraint: the usage of lexical categories in a disambiguation model for resulting parses of some sentence, or incorporating information from lexical categories into an already existing model as auxiliary distribution features (Plank & van Noord, 2008).

2. Hard-constraint: the usage of lexical categories to further restrict the rules that could be licensed in the syntactic analysis of some sentence. Chart pruning is an example of using lexical categories as hard constraints to prune parse chart cells (Dridan, 2009).

In order to compensate for errors resulting from PoS taggers and supertaggers, several studies suggested increasing the number of tags assigned per word, i.e. allowing the tagger to retain some degrees of ambiguity in its output. Accordingly, supertaggers and PoS taggers can be categorized as follows:

1. Single-tagger: single tag per word, which is the most probable tag.
2. Multi-tagger: $n$-best tags per word, where the top $n$ tags are regarded as equally probable tags.
3. Prob-tagger: $n$-best tags per word each weighted by its probability.
4. Selective-tagger: only tags that recieve a probability higher than a predefined value $\beta$.
5. All-tagger: the set of all candidate tags with a non-zero probability.

Curran et al. (2006), for example, implemented multi-tagging by assigning multiple tags to each word where the probabilities of these tags are within a factor of $\beta$ to the most probable tag for that word.

Instead of assigning multiple tags per word, one could assign multiple sequences of tags per sentence. We refer to such models as "$n$-best list models". One instance of $n$-best decoding algorithms is the *List Viterbi Algorithm* (LVA) (Seshadri & Sundberg, 1994) which finds the top $n$ globally best tag sequences. The rationale behind using LAV is that in most cases the difference between the best tag sequence and second best one may be one or two tags, hence, the tag ambiguity need be retained only in the positions (words) where the $n$-best lists differ, the rest of the words would receive one tag each.

## 2.3 ERG Parsing Pipeline

The Linguistic Grammars Online (LinGO) English Resource Grammar (ERG; Flickinger, 2000) is a broad-coverage, linguistically precise grammar for English written within the linguistic framework of Head-driven Phrase Structure Grammar (HPSG), a heavily lexicalized, unification-based formal theory of language (Pollard & Sag, 1994). The ERG has been under continuous development since 1993, and in this thesis we use the 2012 version of the ERG (dubbed 'ERG1212'). As of the time, the ERG provides a manually constructed lexicon of approximately 38,500 lexical entries (we will explain the concept of lexical entries in Chapter 4).

The deep linguistic analysis pipeline of the ERG consists of several 'shallow' pre-processing components (Adolphs et al., 2008). As depicted in Figure 2.1, the ERG parsing pipeline consists of four main stages prior to the syntactic parsing proper.

The pipeline's first input is 'raw untokenized text' (a sequence of characters). In stage one, the pipeline starts by normalizing the input text, which includes tasks such as quote normalization and disambiguation. Then it splits the

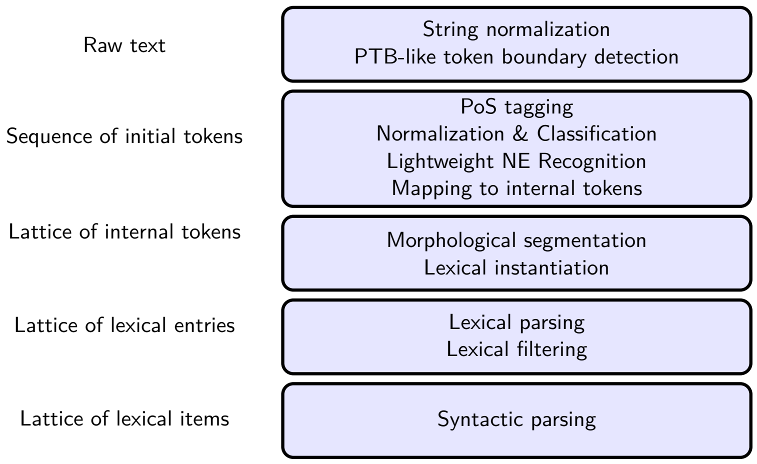

Raw text
String normalization
PTB-like token boundary detection
Sequence of initial tokens
PoS tagging
Normalization & Classification
Lightweight NE Recognition
Mapping to internal tokens
Lattice of internal tokens
Morphological segmentation
Lexical instantiation
Lattice of lexical entries
Lexical parsing
Lexical filtering
Lattice of lexical items
Syntactic parsing

Figure 2.1: The `ERG` parsing pipeline showing the input to each stage on its left side

normalized text into `PTB`-compliant tokens, thus producing a sequence of so-called *initial tokens*.

In the second stage, the sequence of initial tokens is tagged using a `PTB`-trained PoS tagger. Subsequently, a small number of spell correction rules are applied as a second round of text normalization. Additionally, each initial token is assigned a 'surface-form' class (such as numeric, alphanumeric, non-alphanumeric). A 'lightweight' named entity recognizer is utilized to assign categories such as numbers, email and web addresses to initial tokens. The second stage ends by mapping 'initial tokens' to the so-called 'internal tokens' producing a lattice of *internal tokens*. The internal token mapping phase transforms initial tokens to conform to the `ERG` tokenization conventions, e.g. among other things (to which we shall come back in Chapter 3), attaching punctuations to adjacent initial tokens (remember that one of the main differences between `PTB` and `ERG` tokenization is the treatment of punctuation marks).

The third stage consists of (a) *morphological segmentation* followed by (b) *lexical instantiation*. Morphological segmentation enumerates all possible ways of deriving a word (from some inflected form), e.g. '`sleeps`' has at least two segmentations (1) '`sleep`' as a verb, third person singular, or (2) '`sleep`' a noun, in the plural. During the lexical instantiation phase, lexical entries that can be made up from (sequences of) tokens in the internal token lattice are instantiated (as each lexicon entry consists of one or more internal tokens).

In the fourth stage, lexical parsing is essentially chart parsing, however limited to the grammar's lexical rules (the inflectional and derivational rules) and constrained by the segmentation decisions made in the morphological segmentation phase. The output of the lexical parser is filtered by hard constraints, e.g. eliminating redundant lexical items, and passed to the syntactic parser (the fifth stage).

This 'hybrid processing' of the ERG allows achieving richer syntactic analysis, but it comes at a price. The ERG parsing pipeline is computationally expensive as it follows a conservative strategy of not making uncertain decisions, but rather deferring them to the syntactic parsing phase. Hence, we believe that by improving the ERG 'pre-parsing' steps (stages 1–4) the overall parsing performance would improve. In this project, we aim to improve the ERG tokenization and lexical categorization through the use of sequence labeling techniques, namely CRF, which we will explain in the following section.

## 2.4 Sequence Labeling

Sequence labeling (or sequence classification) models fall under two types, non-sequential and sequential models. Non-sequential (point-wise) classification considers one observation at a time, extracts features about that observation, and then accordingly classifies the observation instance into one class out of a predefined discrete set. Sequential (or sequence) classifiers, however, predict an output vector of classes given an input vector of observations (through extracting a vector of features for each input item).

Hidden Markov models (HMMs), maximum entropy Markov models (MEMM) and conditional random fields (CRFs) are examples of 'probabilistic' sequence classifiers, in contrast to a finite-state transducer which is a non-probabilistic sequence classifier.

We rely on CRFs to model the tasks of tokenization and lexical categorization; first because CRFs have achieved competitive results and proven very successful in many NLP tasks (Sha & Pereira, 2003; Skjærholt, 2011; Lapponi et al., 2012). Second, recent works on ERG lexical categorization (Dridan, 2009; Ytrestøl, 2012) investigated the use of HMM, Maximum Entropy (MaxEnt) and support vector machine (SVM) techniques but none, to our knowledge, looked into using CRFs. Third, with our definition of lexical categories we can examine the scalability of CRFs across label sets of different granularity.

CRF can be viewed as a discriminative analogue of the generative HMM, hence we first give an explanation of HMM and build on that to introduce CRF.

### 2.4.1 Hidden Markov Model

Hidden Markov Models (HMM) are one of the most important, and yet simplest, language modeling techniques in NLP.

An HMM is a *generative model* based on a Markov chain which is a directed probabilistic graphical model with the *Markov assumption*. So in order to define HMM we first need to lay down the concepts of 'generative models' and 'Markov assumption'.

**Generative models** Suppose we have a set of training examples $x^{(i)}, y^{(i)}$ for $i = 1 \dots m$ where each $x^{(i)}$ is an input and $y^{(i)}$ is the corresponding output (labels). The task is to learn a function $f$ that maps inputs $x$ to labels $f(x)$, e.g. in PoS tagging we want the function $f$ to take a sentence (a sequence of words) as an input and map it to a sequence of PoS tags.

In a generative model this mapping is achieved by learning (from some set of training examples) the joint distribution $P(x, y)$ and this joint distribution is what mainly characterizes generative models. However, with the rule of conditional probability we can factor $P(x, y)$ into

$$P(x, y) = P(y)P(x|y) \tag{2.1}$$

Now that we introduced Equation 2.1, let's see how it can be used to predict outputs from given inputs. For any input $x$ we want to find $f(x)$, the sequence of labels that maximizes the probability of $y$ given the input $x$

$$f(x) = \arg\max_y P(y|x)$$

Using Bayes' theorem[4] we can write

$$f(x) = \arg\max_y \frac{P(y)P(x|y)}{P(x)}$$

Observe that $P(x)$ does not vary with $y$, so we can discard it as it is constant with respect to $y$, thus

$$f(x) = \arg\max_y P(y)P(x|y)$$

The right-hand side of the last equation corresponds to the right-hand side of Equation 2.1.

**Markov assumption** Consider a sequence of random variables $X_1 \dots X_n$ each of which can take any value in a predefined finite set $\mathcal{V}$. We want to model the joint probability distribution

$$P(X_1 = x_1, X_2 = x_2 \dots X_n = x_n) \text{ where } x_1 \dots x_n \in \mathcal{V} \tag{2.2}$$

We can rewrite Equation 2.2 using the chain rule (which follows by the definition of conditional probablity) as such

$$\begin{aligned} &P(X_1 = x_1, X_2 = x_2 \dots X_n = x_n) \\ &= P(X_1 = x_1) \prod_{t=2}^{n} P(X_t = x_t | X_1 = x_1 \dots X_{t-1} = x_{t-1}) \end{aligned} \tag{2.3}$$

[4]The simple form of Bayes' theorem for events $A$ and $B$:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Now the 'First-Order Markov' assumption states that $X_t$ is conditionally independent of all the previous random variables when $X_t$ is conditioned only on $X_{t-1}$. More formally, for any $t \in 2 \dots n$ for any sequence $x_1 \dots x_n$

$$P(X_t = x_t | X_1 = x_1 \dots X_{t-1} = x_{t-1}) \approx P(X_t = x_t | X_{t-1} = x_{t-1}) \tag{2.4}$$

By applying the Markov assumption to Equation 2.3 we get

$$\begin{aligned} &P(X_1 = x_1, X_2 = x_2, \dots X_n = x_n) \\ &\approx P(X_1 = x_1) \prod_{t=2}^{n} P(X_t = x_t | X_{t-1} = x_{t-1}) \end{aligned}$$

If we consider the 'Second-Order Markov' assumption, Equation 2.3 would be rewritten as such

$$\begin{aligned} &P(X_1 = x_1, X_2 = x_2 \dots X_n = x_n) \\ &\approx P(X_1 = x_1) \times P(X_2 = x_2 | X_1 = x_1) \times \prod_{t=3}^{n} P(X_t = x_t | X_{t-2} = x_{t-2}, X_{t-1} = x_{t-1}) \end{aligned}$$

For convenience, we assume special random variables $X_0$ and $X_{-1}$ (start symbols) so that we can write

$$\begin{aligned} &P(X_1 = x_1, X_2 = x_2 \dots X_n = x_n) \\ &\approx \prod_{t=1}^{n} P(X_t = x_t | X_{t-2} = x_{t-2}, X_{t-1} = x_{t-1}) \end{aligned}$$

Notice that, contrary to the exact decomposition we get from the chain rule, the Markov assumption does not lead to exact equality, obviously because of the independence *assumption.*

**Bigram `HMM`** In general, an `HMM` defines a joint distribution over a word sequence $x = x_1, x_2 \dots x_n$ paired with a tag sequence $y = y_1, y_2 \dots y_n$ where $x_t$ is the $t^{th}$ word in the sentence and $y_t$ is the tag of the $t^{th}$ word (depicted in Figure 2.2).

For any word sequence $x = x_1, x_2 \dots x_n$ where $x_t \in \mathcal{V}$ for $t = 1 \dots n$, and $\mathcal{V}$ is the set of possible words, and for any tag sequence $y = y_1, y_2 \dots y_n$ where $y_t \in \mathcal{S}$ for $t = 1 \dots n$ and $\mathcal{S}$ is the set of possible labels (states), the *joint probability* of the word and tag sequences in a first-order `HMM` is

$$p(x_1 \dots x_n, y_1 \dots y_n) = \prod_{t=1}^{n} P(y_t | y_{t-1}) P(x_t | y_t) \tag{2.5}$$

Equation 2.5 consists of two terms (parameters), *transition probability* and *emission probability*, $P(y_t | y_{t-1})$ and $P(x_t | y_t)$, respectively. Notice that the transition probability depends on first-order Markov assumption, hence the

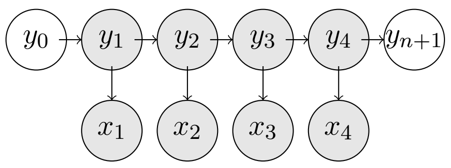


Figure 2.2: A bigram HMM — This figure is also called 'independency graph' where only dependency information is represented, and we can see that $y_t$ depends only on $y_{t-1}$.

| | |
|---|---|
| $\mathcal{V}$ | the vocabulary, the set of possible words |
| $x = x_1, x_2 \ldots x_n$ | a sequence of observations where $x_t \in \mathcal{V}$ |
| $\mathcal{S}$ | the set of possible states (tags or labels) |
| $P(y_t\|y_{t-1})$ | the probability of moving from state (label) $y_{t-1}$ to state $y_t$ |
| $P(x_t\|y_t)$ | the probability of observation $x_t$ being generated from the state $y_t$ |
| $y_0$,$y_{n+1}$ | start and final states, special states that don't generate observations |

Table 2.1: HMM specifications

name 'bigram' HMM. Table 2.1 lists the components of an HMM that depends only on the immediately preceding observation.

From Equation 2.5, the HMM makes two independence assumptions that, as we shall see, impair its performance. First, the Markov assumption and second the assumption that single observations are conditionally independent from each other. In the following section we introduce discriminative models which make no unwarranted assumptions on the dependencies among observations.

### 2.4.2 Conditional Random Fields

As described in the previous section, generative models explicitly model a joint probability distribution $P(x, y)$ over inputs and outputs. Moreover, they do not model all dependencies among inputs, but rather make strong independence assumptions which can lead to reduced performance. The observed features (inputs) are often correlated with each other, meaning that they either have a lot of redundant information or are strong indicators of each other, and they are best represented in terms of interacting features and long-range dependencies between the observations. To clearly illustrate the point here, we borrow an example from Alpaydin (2004). One is still able to read this w?rd because the sequence of characters which makes up the word is constrained by the words of the language. These contextual dependencies may also occur on higher levels such as the syntax (dependencies among words) or even the pragmatics (dependencies among sentences). As a consequence of the independence assumption, the model may overestimate or underestimate some correlated features which may result in a skewed probability distribution. However, trying to model the 'correct' (in)dependencies (by adding edges between correlated features) is hard and might lead to densely connected, intractable models.

So instead of modeling the joint probability, discriminative models attempt to model the conditional distribution $P(y|x)$ considering only the distribution of $y$ over $x$. Therefore, discriminative models can use arbitrary, non-independent features of the observation sequence without having to model those dependencies.

Conditional random fields (`CRF`; Lafferty et al., 2001) are an instance of sequence labeling discriminative models. For simplicity and clarity, we will extend the definition of `HMM`s to arrive at that of the `CRF` following the explanation of Sutton and McCallum (2006).

We first reformulate the bigram `HMM` equation (Equation 2.5) to allow generalization

$$P(x,y) = \frac{1}{Z}\exp\left(\sum_t \sum_{i,j\in S} \lambda_{ij} 1_{\{y_t=i\}} 1_{\{y_{t-1}=j\}} + \sum_t \sum_{i\in S} \sum_{o\in O} \mu_{oi} 1_{\{y_t=i\}} 1_{\{x_t=o\}}\right) \tag{2.6}$$

$\lambda_{ij}$ and $\mu_{oi}$ are the main parameters in the equation above. $1_{\{y_t=i\}}1_{\{y_{t-1}=j\}}$ and $1_{\{y_t=i\}}1_{\{x_t=o\}}$ are indicator functions that return one if the subscript conditions are satisfied and zero otherwise.

By setting the distribution parameters $\lambda_{ij} = logP(y' = i|y = j)$ and $\mu_{oi} = logP(y = i|x = o)$ we get the original `HMM` equation[5]. But since $\lambda_{ij}$ and $\mu_{io}$ are not necessarily log probabilities, we need a normalization function $Z$ to guarantee that the distribution sums to one.

Now we can introduce feature functions $f_k(y_t, y_{t-1}, x_t)$, which are just a notational trick to make the formula compact

$$P(x,y) = \frac{1}{Z}\exp\left(\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t)\right) \tag{2.7}$$

We know that $P(y|x) = \frac{P(x,y)}{P(x)}$, and one way to compute $P(x)$ is through marginalizing $y$ over $P(x,y)$, thus

$$P(y|x) = \frac{P(x,y)}{\sum_{y'} P(x,y')}$$

[5]Remember: $a^{log_a(x)} = x$ and $a^{x+y} = a^x a^y$

Now we substitute Equation 2.7 into the previous equation

$$P(y|x) = \frac{\exp\left(\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t)\right)}{\sum_{y'} \exp\left(\sum_{k=1}^{K} \lambda_k f_k(y'_t, y'_{t-1}, x_t)\right)} \tag{2.8}$$

Equation 2.8 represents a linear-chain CRF that encodes features only for the current word's surface form. By allowing the feature functions to be more general than indicator functions we arrive at the general definition of linear-chain CRFs.

At this point we can present the general definition of linear-chain CRFs by Sutton and McCallum (2006):

Let $Y$, $X$ be random vectors, $\Lambda = \lambda_k \in \Re^K$ be a parameter vector, and $\{f_k(y, y', x_t)\}_{k=1}^{K}$ be a set of real-valued feature functions. Then a linear-chain conditional random field is a distribution $P(y|x)$ that takes the form

$$P(y|x) = \frac{1}{Z(x)} \exp\left(\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t)\right)$$

where $Z(x)$ is an instance-specific normalization function

$$Z(x) = \sum_{y} \exp\left(\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t)\right)$$

(Sutton & McCallum, 2006)

Lavergne et al. (2010) distinguish between two types of feature functions, *unigram* and *bigram*. Given our notion of feature functions, these two types can be defined as

$$f_{y,x}(y_t, y_{t-1}, x_t) = \mathbf{1}(y_t = y, x_t = x)$$
$$f_{y',y,x}(y_t, y_{t-1}, x_t) = \mathbf{1}(y_{t-1} = y', y_t = y, x_t = x)$$

Bigram features are a special type of *structural features* which encode information about the $n$ adjacent labels, and in the case of bigram features $n = 2$.

The learning and inference complexities of a linear chain CRF grow quadratically with the number of labels (Lavergne et al., 2010). Moreover, the number of feature parameters also increases with the size of the label set as follows, where $|X|$ and $|Y|$ are the cardinals of observations and labels, respectively:

$$Unigramfeatures : |Y| \times |X|$$
$$Bigramfeatures : |Y^2| \times |X|$$

In concrete terms, if we have a unigram feature that encodes the word's surface form within a training set of 10,000 word forms and 100 different labels, the number of feature parameters would be $10,000 \times 100 = 1,000,000$. Furthermore, if we use a bigram feature instead of unigram, the number of parameters would be $10,000 \times 100 \times 100 = 100,000,000$. As we shall see in Chapter 4, due to its complexity, training CRFs with very large label sets and massive amounts of data becomes prohibitively expensive.

What we just explained in this section, is just a brief formal definition of CRFs, for further reading we recommend Sutton and McCallum (2006) who explain the parameter estimation, regularization and inference aspects of CRFs; and Lafferty et al. (2001) who introduce CRF as a general graphical structure.

Finally, for tokenization and lexical categorization experiments we use the Wapiti toolkit[6] (Lavergne et al., 2010) which implements, among other models, linear-chain CRFs. All of our experiments in Chapter 3 and Chapter 4, use the L-BFGS algorithm for parameter estimation, because it gives the best results for our setup, even though it takes long time to get to these results.

## 2.5 Significance Hypothesis Testing

In many scientific fields, it is common practice to test for statistical significance of the experimentation results. In statistical natural language processing, statistical hypothesis testing is also applied to compare the experimental results of different systems (models) evaluated on the same test sets.

In spite of its wide acceptance and usage, Velldal (2008) argues that statistical significance in NLP is not all-clear because the details of statistical significance testing are often left out of discussion, only $p$ and $\alpha$ values are reported. We agree with the reservations of Velldal (2008), but still we test for statistical significance in this thesis, occasionally though when the number differences look very small.

In this section, we briefly present the details of statistical significance testing, making sure, however, that all the details needed to replicate the tests are provided.

In NLP, statistical significance tests seek to define whether or not the difference in performance between two systems (or models) is due to chance—that is, to decide if two sets of observations are drawn from the same probability distribution. Hence, we define the null hypothesis $H_0$ as any variation in the set of observations (observed pairs of results) is due to chance. We reject the null hypothesis if the $p$-value is smaller than or equal to a predefined significance level $\alpha$. Throughout this thesis, we reject the null hypothesis at the 0.05 significance level ($\alpha = 0.05$).

We use the Wilcoxon signed-rank test (Wilcoxon, 1945), a non-parametric test that does not assume any distribution from which the observations are

[6]See `http://wapiti.limsi.fr/`

supposedly has been sampled. The null hypothesis in the Wilcoxon test is that the median difference between pairs of observations is zero.

We use the Wilcoxon signed-rank test because, as a non-parametric test, it makes fewer and less stringent assumptions than parametric tests which assume that the observations would approximate a normal distribution. Moreover, within the family of non-parametric tests, the Wilcoxon test is more sensitive than the sign test (Velldal, 2008).

Finally, to collect observations, we follow Spoustová et al. (2009) in splitting the test set into numerous subsets and then evaluate our models on these subsets.

## 2.6 Resources

This section gives a short introduction to the linguistic resources used throughout this project.

### Penn Treebank

The Penn Treebank (`PTB`; Marcus et al., 1993) is a corpus of more than 4.5 million words of American English. It was initially released in 1992 and played a significant role in boosting statistical NLP research.

The `PTB` consists of several sub-corpora ('portions') such as the Wall Street Journal (`WSJ`), the `Brown` corpus and the Switchboard corpus. In this project, we use the one-million-word `WSJ` portion and the `Brown` corpus (Francis, 1964), which contains 500 samples of modern English text, distributed over 15 genres, and totaling more than a million words.

### NANC

The North American News Text Corpus (Graff, 1995) is composed of unannotated news text. The corpus contains 300 million words from different news sources such as the Wall Street Journal and the New York Times. In this thesis, we use 25 million words of the NANC `WSJ` portion.

### GENIA

The `GENIA` corpus (Kim et al., 2003) is a semantically annotated corpus of biomedical literature. It contains 2000 abstracts with more than 400,000 words.

### Redwoods & DeepBank

The `Redwoods` Treebank (Oepen et al., 2004) is a collection of manually disambiguated corpora analyzed with the `ERG` within the `HPSG` framework. It samples a varity of domains, including transcribed dialogue, e-commerce emails, tourism information, and a 100-article portion of the English Wikipedia.

As Redwoods is continuously expanded with new corpora, a large proportion of the PTB WSJ text with gold-standard ERG analyses was added to the treebank recently (DeepBank; Flickinger et al., 2012). By the time of writing this thesis, two versions of DeepBank were available, DeepBank v0.9 (Sections 0 – 15 of the PTB WSJ) and DeepBank v1.0 (Sections 0 – 21).

In all cases, the aforementioned resources require considerable processing in order to be used in tokenization and lexical categorization experiments. Some degrees of inconsistency, furthermore, appears to be a common attribute to almost all of them. More details on the use of these resources will be given in Chapter 3 and Chapter 4.

## 2.7 Literature Review

We are not the first to study the problems of tokenization and lexical categorization. In the following, we review threads of prior research related to our work. First, we introduce Dridan (2009) and Ytrestøl (2012), two recent PhD projects on ERG parsing, as they are closely related to our work.

Dridan (2009) studies the use of lexical information to improve ERG HPSG-based parsing. She investigates improving the PET parser (Callmeier, 2000) using several supertag granularities and two taggers with two different tagging models, namely TnT (HMM-based; Brants, 2000) and C&C (MaxEnt-based; Clark & Curran, 2007). She reports that although TnT achieves higher accuracy than C&C, the latter gives better results than former if its probability distribution is used to guide tag assignment instead of just using the best single-tag. Furthermore, Dridan (2009) looks into the usefulness of tag sequences assigned by TnT and C&C, and again she finds that C&C gives better results because it assigns tag sequences that are acceptable by the parser if not necessarily correct. Finally, Dridan (2009) employs lexical information to improve parsing from three aspects. First, increasing parser robustness by using lexical statistics to handle unknown words. Second, reducing lexical ambiguity to boost parse efficiency. Third, using lexical statistics in parse ranking.

Ytrestøl (2012) evaluates the use of transition-based parsing for ERG HPSG, using C&C and SVM$^{hmm}$ (SVM-based; Joachims et al., 2009) supertaggers to assign ERG lexical categories prior to parsing. Ytrestøl (2012) reports that SVM$^{hmm}$ performs better than C&C when trained on *relatively* limited amounts of data. He finds, however, when training C&C on very large amounts of automatically produced training data, namely the WikiWoods Treecache (Flickinger et al., 2010), it outperforms SVM$^{hmm}$. However, we should point out that (a) C&C needs one million training sentences to achieve the accuracy of the SVM$^{hmm}$ when trained on 250,000 sentences, and (b) SVM$^{hmm}$ is very computationally expensive to train while C&C scales linearly in time to the amount of training sentences.

### 2.7.1 Tokenization

Dridan and Oepen (2012); Tomanek et al. (2007); Foster (2010), inter alios, revived the research on tokenization for English after it had been regarded as an 'uninteresting' sub-problem of NLP for several decades.

Foster (2010), in a study on parsing user-generated content, reports an improvement in statistical parsing accuracy of 2.8 points $F_1$ over user-generated Web content, when making available to the parser gold-standard token boundaries.

While Foster (2010) exemplifies the effect of tokenization on downstream NLP tasks, other studies point out problems in tokenization using different tools. Øvrelid et al. (2010), for example, observe that tokenizing the `GENIA` text using the `GENIA` tagger (Tsuruoka et al., 2005) leads to token mismatches in almost 20% of the sentences in the treebank.

Dridan and Oepen (2012) observe that state-of-the-art statistical constituency parsers perform relatively poorly at replicating even the most common tokenization scheme, viz. that of the `PTB`. They develop, therefore, a cascade of regular, string-level rewrite rules (dubbed `REPP`) for `PTB` tokenization, with almost 99% sentence accuracy. We will use their results as our main point of reference in Chapter 3.

Tomanek et al. (2007) apply `CRF` learners to sentence and token boundary detection in bio-medical research literature. They defined their own uniform tokenization scheme for bio-medical texts and built novel corpora based on the PennBioIE text (Kulick et al., 2004) to train their tools on. In our experiments we also use `CRF`s but we experiment with a range of tokenization conventions and text types—in all cases using generally available, 'standard' resources.

### 2.7.2 Lexical Categorization

As explained earlier, part-of-speech tagging and supertagging, in our view, are instances of lexical categorization. In the following, however, we use the terms 'PoS tagging' and 'supertagging' as they occur in the surveyed works, but we don't convey any fundamental conceptual distinction when one term or the other is used.

Dalrymple (2006) studies the use of a PoS tagger as a soft constraint to reduce the ambiguity of a Lexical Functional Grammar (LFG) parser. She introduces the concept of "tag-sequence equivalence classes" which is the set of parses of a sentence that lead to the same tag sequence. Accordingly, Dalrymple (2006) states that if different parses tend to be associated with different tag sequences (fall in different equivalence classes) then PoS tagging would greatly help parse disambiguation. In the analysis of the parser's output, she finds that 29.47% of the sentences in section 13 from the `WSJ` have the tag sequences of their parses all in the same equivalence class, and so tagging would not help disambiguating for about 30% of the sentences in section 13. However, it was not possible to say how much exactly tagging would help disambiguate the remaining 70%, since she did not have gold standard for LFG-based parses in WSJ section 13. Thus, she assumes that the correct tag sequence would most probably be found in the largest equivalence class. She concludes: "If these data are representative, we can expect to rule out 45-50% of the potential parses for a sentence by choosing the correct tag sequence for the sentence" (Dalrymple, 2006).

Watson (2006) studies the choice of PoS tagging models in order to improve parsing. She conducts her study using an unlexicalized statistical parser, RASP (Robust Accurate Statistical Parsing (Briscoe & Carroll, 1995)), and tests on the PARC 700 Dependency Bank (700 sentences from the WSJ section 23 parsed by the same LFG grammar in Dalrymple (2006); King et al., 2003). She reports if parsing efficiency is the main concern then a single tag should be used, however, if either accuracy or coverage is the more important factor then a multi-tagger can be used.

Prins and van Noord (2003) use HMM PoS taggers to remove the unlikely lexical categories for the Alpino Parser (an HPSG-based parser for Dutch (Bouma et al., 2001). What is interesting about this study is training the tagger on the output of the parser which would have two direct implications. First, it eliminates the problem of incompatibility between the tagset and the pre-terminals of parses. Second, the tagger would learn the tag sequence favored by the parser and not necessarily the correct ones.

Prins and van Noord (2003) experiment with 1365 lexical categories, train on 24 millions words and test on 604 randomly selected sentences from the Alpino Treebank.[7] They conclude that the use of a tagger greatly reduces parsing times, from 60 seconds to 14 seconds of CPU time per sentence, and increases parsing accuracy.

Curran et al. (2006) discuss the use of a supertagger as a front-end for parsers in the realm of Combinatory Categorial Grammar (CCG). They propose assigning one or more lexical categories (supertags) to each word in order to compensate for the high supertagging error rate. Moreover, they report an increase in the accuracy of supertagging when allowing ambiguity in PoS tags, since PoS tags are essential features for supertagging. Testing on section 23 of CCGbank (Hockenmaier, 2003), their single-tag supertagger accuracy rises from 92.0% to 97.7% with a supertag ambiguity of 1.4 categories per word and a PoS tag ambiguity of 1.1 tags per word. Curran et al. (2006) do not explicitly report how this improvement in supertagging accuracy is reflected on parsing.

Yoshida et al. (2007) investigate the effectiveness of ambiguous, multiple PoS tagging to improve an HPSG parser trained on the PTB, as well as to study parser domain adaptation. They report that multiple tags do not have a

[7] "The Alpino Treebank 1.0", University of Groningen, 11 2002, CDROM; http://www.let.rug.nl/~vannoord/trees/

noticeable effect unless combined with a probability distribution that indicates the likelihood of each tag. Additionally, when evaluating on section 23 of the PTB WSJ, the $F_1$ score for the single-tag parser was 84.10% while for the prob-tag parser it was 85.06%, however, the mean parsing time was 936 msec for the latter and 785 msec for the former. As for adaptation to the GENIA corpus, retraining the PoS tagger on the Penn BioIE corpus caused the $F_1$ to increase from 78.29% for a multiple-tag unadapted parser to 82.02%.

Rimell and Clark (2009) investigate adapting a CCG-based parser to the biomedical domain. Taking advantage of the CCG formalism they adapt the parser on two levels, PoS tags and CCG lexical categories (supertags), to which they refer as “adaptation at low levels of representation”.

Retraining a PTB-based PoS tagger on in-domain vocabulary, specifically the first 1000 sentences from the GENIA corpus, results in an accuracy rise from 93.4% to 98.7%. However, to retrain the supertagger, they had to manually annotate the first 1000 sentences from GENIA, and then they created three supertagging models, the first model is the standard model trained on WSJ Sections 02-21 of CCGBank, the second model is trained on the 1000 manually annotated sentences from the GENIA text, and the third one is a hybrid model trained on a combination of WSJ CCGBank and GENIA where they use a weighting factor of 10 for the GENIA sentences. To test these models, they created a combination of PoS taggers and supertaggers, where the baseline is the WSJ PoS tagger and WSJ supertagger with an accuracy of 89.0% and the best-performing one is the GENIA PoS tagger and hybrid supertagger with an accuracy of 93.0%.

Rimell and Clark (2009) evaluate the effect of these models on parsing results using the BioInfer corpus (Pyysalo et al., 2007) and different pipelines of PoS taggers and supertaggers. They report that among different settings of the CCG parser, the GENIA PoS tagger and hybrid supertagger pipeline perform best with an $F_1$ score of 81.5%, while the unadapted pipeline, WSJ PoS tagger and supertagger, has an $F_1$ score of 76.0%.

From the studies surveyed, we see that lexical categories can be used to disambiguate the parser’s output and to improve its accuracy and efficiency. In order to improve the parsing accuracy, multiple tags per word may be needed. Improving parsing efficiency, however, requires limiting the number of tags per word, so that the parse space can be pruned.

## 2.8 Summary

In this chapter, we gave introductory definitions of tokenization and lexical categorization which are the topics of Chapter 3 and Chapter 4, respectively. We also introduced the ERG parsing pipeline to which we will refer repeatedly throughout this thesis. Then, we explained CRFs and its advantages as a discriminative machine learning models, as well as its complexity that might be limiting if the number of labels is very large (as we shall see in Chapter 4). Finally, we reviewed related studies on tokenization and lexical categorization. Through these studies, we showed the effect of tokenization on downstream application. We also surveyed how lexical categorization can be used to improve parsing efficiency and accuracy with different setups and tradeoffs.

# Chapter 3

# Tokenization

Tokenization constitutes a foundational pre-processing step for almost any subsequent natural language processing task. As evidenced by Forst and Kaplan (2006) and Foster (2010) token errors will inevitably propagate into any downstream analysis. Lease and Charniak (2005) also emphasize that many problems can be resolved early in the processing pipeline instead of deferring them to the parser. With a few notable exceptions, however, tokenization quality and adaptability to (more or less subtly) different conventions have received comparatively little attention over the past decades.

In this chapter, we investigate the use of sequence labeling techniques for tokenization, which has been predominantly approached through rule-based techniques, typically finite-state machines or (cascades of) regular expressions (Dridan & Oepen, 2012). We study two interpretations of the tokenization problem, namely the `PTB` and `ERG` tokenization schemes, as they both contribute to the `ERG` parsing pipeline. We present unprecedented work on `ERG` lexical tokenization and a finely-detailed study on `PTB`-like tokenization. Observing variation in tokenization conventions across corpora and processing tasks, we train and test multiple `CRF` sequence labelers and obtain substantial reductions in tokenization error rate over state-of-the-art `PTB` tokenization tools. We also augment our study with a domain adaptation perspective, determining the effects of training on mixed gold-standard data sets and making tentative recommendations for practical usage. In the `ERG` tokenization setup, we also look at partial disambiguation by making available a token lattice to downstream processing.

As our tokenization models improve over the state-of-the-art `PTB` tokenizers and innovate on `ERG` tokenization, we already published our results in Fares et al. (2013). This chapter presents an elaborated version of that work.

## 3.1 Tokenization for ERG Parsing

The English Resource Grammar (ERG) defines its own tokenization conventions. However, as shown in § 2.3, the standard parsing setup for the ERG relies on a PoS tagging pre-processing stage, to provide the grammar with candidate lexical categories for unknown words. For compatibility with external off-the-shelf tools, the ERG adopts PTB-like tokenization at this stage. Afterwards, the PTB-compliant tokens (dubbed *initial tokens*) are mapped to an ambiguous lattice of *internal tokens* to conform to the ERG tokenization conventions.

While PTB tokenization is relatively well-understood (at least in comparison to that of the ERG) ERG-like tokenization has not been investigated before. To date, parsers working with the ERG either operate off an ambiguous full token lattice (Adolphs et al., 2008) or assume idealized gold standard ERG tokenization (Zhang & Krieger, 2011; Ytrestøl, 2011; Evensberget, 2012). Hence, ERG token boundaries are typically determined as a by-product of syntactic analysis. As we shall see later in this chapter, the ERG tokenization scheme shows a stark contrast to ‘classic’ schemes (PTB-like), presenting many more token-level ambiguities to the sequence labeler (reflecting use of punctuation and multi-word lexical units).

In this work, we seek to determine to what degree CRF sequence labeling in isolation scales to the ERG conception of the tokenization task, considering both one-best and $n$-best decoding. We see at least two candidate applications of such stand-alone ERG tokenization, viz. (a) to enable ERG lexical categorization (cf. Chapter 4); and (b) to speed up parsing with the ERG, reducing or eliminating tokenization ambiguity.

Finally, as PTB-like tokenization plays an essential role in ERG parsing, we demonstrate that using sequence labeling for PTB tokenization scheme as well improves over state-of-the-art results. Moreover, since the PTB-style tokenization constitutes the basis for other broadly used resources, such as the GENIA Treebank, we further expand our experiments to include different corpora that variously adhere to the PTB tokenization scheme.

## 3.2 Formal Definition

Tokenization is the process of splitting a stream of characters into smaller, word-like units for downstream processing; or in the definition of Kaplan (2005), tokenization is breaking up *“natural language text … into distinct meaningful units (or tokens)”*.

Given this definition, the concept of tokenization seems rather self-evident. It is quite possible, however, that different types of downstream processing may call for variation in tokenization, or different notions of ‘meaningful units’, e.g. morphosyntactic analysis vs. full syntactic analysis[1] (Chiarcos et al., 2009).

[1]One can argue that splitting ‘`department store`’ into two words is better for morphosyn-

In fact, we observe that in the context of broad-coverage grammars such as the ERG, very different views on tokenization can be motivated, including the recognition of some types of multi-word expressions. These lead to more token boundary ambiguities and, thus, make tokenization a task that is intricately entangled with downstream analysis.

What is more, tokenization is sometimes interpreted to include some amount of string-level normalization, such as the disambiguation of quote marks (into opening and closing, or left and right delimiters). Assuming that the tokenizer input does not make this distinction already, i.e. text using non-directional, straight ASCII quotes only, such disambiguation can be performed with relative ease at the string level—based on adjacency to whitespace—but not later on.

In our view, however, separating the two sub-tasks of tokenization explicitly (boundary detection vs. text normalization) is a methodological advantage, making it possible to experiment more freely with different techniques[2]. Quite possibly owing to this somewhat diffuse interpretation of the task, there has been very little work on machine learning for high-quality tokenization. Throughout this thesis, therefore, tokenization concerns only token boundary detection.

As mentioned in § 2.3, ERG parsing relies on, among other components, two substantively different tokenization schemes. In the following we contrast the PTB and ERG tokenization schemes, highlighting main design principles and key differences.

### 3.2.1 PTB-Style Tokenization

According to the original PTB tokenization documentation[3], the PTB-style tokenization is "fairly simple". The key principles in PTB-compliant tokenization are:

- Whitespaces are explicit token boundaries.
- Most punctuation marks are split from adjacent tokens.
- Contracted negations are split into two different tokens.
- Hyphenated words and ones containing slashes are not split.

Even though PTB tokenization seems to be uncomplicated, not so many standard NLP tools can *accurately* reproduce PTB tokenization from raw strings (Dridan & Oepen, 2012).

tactic analysis, whereas the syntactic analyzer would prefer treating '`department store`' as a single token (Chiarcos et al., 2009).

[2] Maršík and Bojar (2012) present a related but again interestingly different view, by combining tokenization with sentence boundary detection to form one joint classification task, which they approach through point-wise Maximum Entropy classification; their experiments, however, do not address tokenization accuracy for English.

[3] `http://www.cis.upenn.edu/~treebank/tokenization.html`

Among other problems (to which we will return in § 3.3), one recurring issue in the PTB tokenization are sentence-final abbreviations, especially the U.S. Observe how the preceding sentence, in standard typography, ends in only one period. In the PTB, however, there seems to be a special treatment for the 'U.S.' abbreviation, so whenever it occurs at the end of a sentence, an extra period is added to that sentence; however, if for example, 'etc.' ends the sentence no extra period would be added.

While the original PTB scheme is reasonably defined and broadly understood, today there exist many variants. These often address peculiarities of specific types of text, e.g. bio-medical research literature or user-generated content, or they refine individual aspects of the original PTB scheme, for example the treatment of hyphens and slashes.[4] Variations on the PTB scheme are at times only defined vaguely or extensionally, i.e. through annotated data, as evidenced for example in resources like the Google 1T n-gram corpus (LDC #2006T13) and the OntoNotes Initiative (Hovy et al., 2006).

The NLP community is moving toward producing—and processing—new data sets other than the PTB. At the same time, much current work uses PTB tokenization conventions with "some exceptions".[5] A consequence of these developments, in our view, is the decreasing community agreement about 'correct' tokenization and, thus, greater variety in best practices and types of input expected by downstream processing. For these reasons, we believe that ease of adaptability to subtle variation (e.g. retraining) is of growing importance, even in the realm of PTB-style tokenization.

### 3.2.2 ERG-Style Tokenization

The ERG assumes a quite different, and challenging, conception of tokenization that diverges from PTB legacy in three key aspects. First, the ERG treats most punctuation marks as pseudo-affixes to words i.e. commas, periods, quote marks, parentheses, et al. are *not* tokenized off. The punctuation affixes are treated akin to inflectional morphology, rather than as separate tokens that attach syntactically. While Adolphs et al. (2008) offer a linguistic argument for this analysis, we just note that it eliminates the dilemma presented by periods in sentence-final abbreviations and in general appears to predict well the interactions of whitespace and punctuation in standard orthography. As a consequence, however, some punctuation marks are ambiguous between being units of 'morphology' or syntax, for example colons and hyphens. Second, unlike in the PTB conventions, hyphens (or dashes) and slashes in the ERG

[4]The 2008 Shared Task of the Conference on Natural Language Learning, for example, produced variants of PTB-derived annotations with most hyphens as separate tokens, to match NomBank annotation conventions of propositional semantics (Surdeanu et al., 2008).

[5]The 2012 Shared Task on Parsing the Web, for example, used portions from OntoNotes 4.0 which is tokenized with "slightly different standards" from the original PTB conventions (Petrov & McDonald, 2012).

introduce token boundaries, e.g. ⟨`open-`, `source`⟩ or ⟨`3`, `-`, `4`⟩. Observe, however, that the functional distinction between intra-word hyphens and inter-token dashes projects into different tokenizations, with the hyphen as a pseudo-affix in the first example, but the n-dash a separate token in the second. As both functions can be encoded typographically as either a single hyphen or an n-dash (- or –, respectively), the functional asymmetry introduces token-level ambiguity. Third, the `ERG` treats contracted negations (e.g. '`don't`') as a single token, contrary to the paradigm suggested in the `PTB`. Observing the sharp grammaticality contrast between, say, *Don't you see?* vs. **Do not you see?*, Adolphs et al. (2008) argue linguistically against tokenizing off the contracted negation in these cases.

In addition to the three aspects just mentioned, the `ERG` introduces the concept of 'lexical tokens' (or multi-word expressions) that constitutes a major challenge in `ERG` tokenization.

**Multi-word expressions** Multi-word expressions (`MWEs`), in the definition of Sag et al. (2002), are *"idiosyncratic interpretations that cross word boundaries (or spaces)"*

The `ERG` includes a lexicon of some classes of multi-word expressions, 'fixed' and 'semi-fixed' `MWEs` according to the classification of Sag et al. (2002). The `ERG` lexicon contains, most notably, so-called 'words with spaces' (e.g. ⟨`ad`, `hoc`⟩ or ⟨`cul`, `de`, `sac`⟩), which are 'fixed' `MWEs` that disallow variability, meaning that expressions like ⟨`ad`, `hoc`⟩ can be externally modified (*very ad hoc*) but not internally (**ad very hoc*). Other instances of `ERG` `MWEs` include proper names with syntactic idiosyncrasies (⟨`New`, `Year's`, `Eve`⟩, for example, can be a temporal modifier by itself, i.e. without a preposition), multi-word prepositions and adjectives (e.g. ⟨`as`, `such`⟩ or ⟨`laid`, `back`⟩), and of course 'singleton' words split at hyphens (e.g. ⟨`e-`, `mail`⟩ or ⟨`low-`, `key`⟩).

It is noteworthy that `MWEs` are quite frequent in the `ERG`; almost 10% of the `ERG1212` lexicon are multiword lexical entries[6].

Taken together, these conventions make `ERG`-style tokenization a more intricate task, calling for a certain degree of context-aware disambiguation and lexical knowledge.

### 3.2.3 Tokenization as a Sequence Labeling Problem

The interpretations of tokenization as a sequence labeling problem might vary on five different dimensions, some are specific to the tokenization task itself, others are common to various machine learning problems. In general, to recast tokenization as a sequence labeling problem we need to define:

[6]The `ERG1212` (introduced before) is the 2012 release of the `ERG` and it includes a hand-built lexicon of some 38,500 lemmata (i.e. uninflected stems).

1. **Target tokenization scheme** The tokenization conventions which the model is expected to learn. This can be any coherently defined or practised tokenization scheme such as `PTB` or `ERG`.

2. **Basic processing unit** The smallest unit that can make up a single token, or said differently, the instances the sequence labeling model is supposed to label. We will elaborate more on this below.

3. **Tokenization labels** The set of classification labels. We perceive the tokenization of an input as a sequence of binary classifications, hence we define two classification labels, namely 'SPLIT' and 'NONSPLIT'.

4. **Sequence labeling models and features** In the tokenization task as such, there isn't any restriction on the type of sequence labeling algorithm one can employ.

5. **Data split** The train-development-test data split.

The core of the tokenization task is to split raw untokenized text into smaller units. To identify the potential splitting points, we need to first identify the atomic units that cannot be further split.

One can look at 'tokens' from two perspectives: First, what might be a token separator or what is not a token. Second, what constitutes a token. In the *what-is-not-a-token* view, there is one main approach that is to define a set of characters as token separators such as whitespace, period, and comma. While in the *what-is-a-token* perspective, we can distinguish at least two approaches.

First, the character-based method where each character in the raw string is a candidate token (thus followed by a candidate tokenization point). While this is the common practice for word segmentation of CJK (Chinese, Japanese, and Korean) languages which have to varying degrees logographic writing systems, this is probably not suitable for languages with phoneme-based writing system where single characters are typically representing basic significant sounds rather than a word, morpheme or semantic unit. An alternative approach would be to group the characters into different character classes and, accordingly, define the concept of **sub-tokens** where each one consists of a sequence of homogeneous characters belonging to the same character class.

In this approach we define the candidate tokenization points as the points between each pair of sub-tokens, and a token consists of one or more sub-tokens. The rationale behind this notion of character classes is that a consecutive sequence of homogeneous characters is most likely to constitute one single token and should not be further split. Table 3.1 lists all the character classes we define for English tokenization. If a character does not fall under any of the classes in Table 3.1, it would be classified as 'other'.

More concretely, a word like '`well-educated`' consists of three sub-tokens: '`well`', '`-`' and '`educated`', where these three sub-tokens belong to three character classes, respectively: '`alpha`', '`hyphen`' and '`alpha`'. Hence, the candidate

Table 3.1: The character classes

| Class | Description |
|---|---|
| alphaC | Alphabetical characters with the initial one capitalized |
| alpha | Alphabetical characters |
| num | Numerical characters |
| H | Hyphen |
| C | Comma |
| D | Dot |
| L | Colon |
| SL | Semicolon |
| OP | Open parenthesis |
| CP | Close parenthesis |
| SQ | Single quote |
| DQ | Double quote |
| D | Dash |
| FS | Forward slash |
| BS | Backward slash |
| OQ | Open quote |

tokenization points are: (1) between alpha and hyphen, (2) between hyphen and alpha, `well¦-¦educated`.

## 3.3 PTB-Style Tokenization

First, we carried out in-domain and out-of-domain tests of our machine learning based tokenizers with the PTB-style tokenized data.

For the training of the PTB model, we followed the standard data split used in part-of-speech tagging experiments. We train our model on PTB WSJ sections 0 to 18, improve it on sections 19 to 21 and test it on sections 22 to 24. Moreover, we extract the 'gold-tokens' from the treebank and align them with raw strings provided with the 1995 release of the PTB (LDC #1995T07)[7].

Assuming the gold sentence boundaries, we split each sentence into a sequence of sub-tokens as defined in § 3.2.3, hence the classifier's task is to decide whether to join adjacent sub-tokens or keep them split. However, our notion of character classes as such doesn't cover all possible token boundaries in the PTB conventions. More concretely, according to the PTB tokenization conventions the word '`cannot`' must be split into two tokens, '`can`' and '`not`', but our character classes approach would recognize '`cannot`' as a single sub-token, eliminating the possibility of a split within a homogenous sequence of characters. We see a similar problem with contractions in the PTB, but we

[7]We are indebted to Rebecca Dridan for sharing her version of the 'raw' WSJ text, aligned to sentences from the PTB for gold-standard tokenization.

Table 3.2: Features used for PTB tokenization classifiers — W: word's surface form, Space: followed by space, $CC_i$: character class, Len: surface form length, FC: first character, LC: last character; †: Bigram features, ‡: Unigram and bigram features, unmarked features are unigram features.

| **Feature** | **Feature** | **Feature** |
|---|---|---|
| $W_i$ | $W_i$ & $W_{i-1}$ & $W_{i-2}$ & $W_{i-3}$ | $W_{i+1}$ & $CC_{i+1}$‡ |
| $W_{i+1}$‡ | $W_i$ & $W_{i+1}$ & $W_{i+2}$ & $W_{i+3}$ | $FC_i$ |
| $W_{i+2}$‡ | $Space_i$† | $LC_i$ |
| $W_{i+3}$‡ | $W_i$ & $Space_i$ | $FC_i$ & $FC_{i+1}$ |
| $W_{i-1}$‡ | $Space_i$ & $Space_{i+1}$† | $FC_i$ & $FC_{i-1}$ |
| $W_{i-2}$‡ | $Space_i$ & $Space_{i-1}$† | $LC_{i-1}$ & $FC_i$ |
| $W_{i-3}$‡ | $CC_i$‡ | $LC_i$ & $FC_{i+1}$ |
| $W_i$ & $W_{i+1}$ & $W_{i+2}$ | $CC_i$ & $CC_{i+1}$‡ | $Len_i$ |
| $W_i$ & $W_{i-1}$ & $W_{i-2}$ | $CC_i$ & $CC_{i-1}$‡ | $CC_i$ & $Len_i$ |
| $W_i$ & $W_{i-1}$ & $W_{i+1}$ | $W_{i-1}$ & $CC_{i-1}$‡ | $W_i$ & $CC_i$ & $Len_i$† |

believe this is a restricted phenomenon peculiar to PTB tokenization conventions. Extra regular rules were added to recognize the sub-tokens in these special cases[8].

As listed in Table 3.2, we construct a set of 30 features for each sub-token exploiting their lexical and orthographical information as well as their local context in a bidirectional-window of seven sub-tokens (denoted by $W_i$ with $i \in [-3\,..\,3]$). The following is an example sentence with *sub-token* indices below and the corresponding *character classes* above each unit, respectively.

$$\underbrace{\overbrace{\textit{PC}}^{\textit{alphaC}}}_{1}\ \underbrace{\overbrace{\textit{shipments}}^{\textit{alpha}}}_{2}\ \underbrace{\overbrace{\textit{total}}^{\textit{alpha}}}_{3}\ \underbrace{\overbrace{\textit{some}}^{\textit{alpha}}}_{4}\ \underbrace{\overbrace{\$}^{\textit{dollar}}}_{5}\ \underbrace{\overbrace{38}^{\textit{num}}}_{6}\ \underbrace{\overbrace{.}^{\textit{dot}}}_{7}\ \underbrace{\overbrace{3}^{\textit{num}}}_{8}\ \underbrace{\overbrace{\textit{billion}}^{\textit{alpha}}}_{9}\ \underbrace{\overbrace{\textit{world}}^{\textit{alpha}}}_{10}\ \underbrace{\overbrace{\text{-}}^{\textit{hyphen}}}_{11}\ \underbrace{\overbrace{\textit{wide}}^{\textit{alpha}}}_{12}\ \underbrace{\overbrace{.}^{\textit{dot}}}_{13}$$

### 3.3.1 Results and Error Analysis

To allow a fair comparison with other tokenization tools, we measure tokenization performance on the sentence level, i.e. will consider a sentence erroneous when it contains at least one token mismatch against the gold standard.[9]

Testing our PTB model on the last three sections of PTB WSJ led to an error rate of **0.93%**. As a reference point we compared our system with the best-performing set of rules from Dridan and Oepen (2012) (henceforth

[8]These rules are based on the *sed* script provided by the PTB; see `http://www.cis.upenn.edu/~treebank/tokenizer.sed`

[9]Thus, unless explicitly said otherwise, all experimental results we report in this chapter indicate sentence-level metrics i.e. full-sentence accuracy or full-sentence error rate.

the REPP tokenizer). Evaluating their rules on the last three sections of PTB resulted in **1.40%** error rate.[10]

Examining the errors of our PTB model revealed that about 45% of the sentence mismatches are due to tokenization inconsistencies within the PTB (gold-standard errors), while the rest (55%) are classification errors. More precisely, almost 30% of the PTB model's 'errors' are to be blamed on the 'U.S.' idiosyncrasy in the PTB (discussed in § 3.2.1). Inconsistencies in splitting hyphenated words are the source of another 4% of the total error rate, for example 'trade-ethnic', in the following sentence, is split into three tokens in the gold-standard in contrast to the PTB tokenization scheme.

*"But too often, these routines lack spark because this sitcom, like all sitcoms, is timid about confronting Mr. Mason's stock in trade-ethnic differences."*

Yet another problem with hyphens in the PTB are cases like *"[. . . ] on energy-, environmental- and fair-trade-related [. . . ]"* where the hyphens are separated from 'environmental-' and 'energy-', while 'fair-trade-related' is *correctly* kept all together; this case shows that it is impossible to exactly replicate the PTB tokenization. Other types of inconsistencies, such as not splitting off some punctuation and splitting periods from acronyms (e.g. 'Cie.', 'Inc.', and 'Ltd.') sum up to almost 11% of the errors.

Interestingly enough, our PTB model shares 77% of the errors with the REPP tokenizer (apart from 'U.S.'-related mismatches).

Finally, to build a better understanding of the non-trivialty of the PTB tokenization task, we present a unigram baseline model which assigns to each sub-token its most frequent label and for unseen sub-tokens the most frequently occurring label in the training data. This method results in merely a 40.04% sentence-level accuracy.

In Figure 3.1 we observe the CRF learner's in-domain behavior with increasing amounts of training data. We see that the high accuracy can be established with relatively small amount of training data, and no steep improvement in accuracy thereafter, which can be partially due to the noisy (inconsistent) annotations in the PTB.

### 3.3.2 Domain Variation

To further validate our system, and also to investigate the robustness of ML-based and rule-based approaches for tokenization, we test the two systems on a different genre of text, the Brown Corpus.[11] Although the Brown annotations

[10]We had to replicate the experiments of Oepen & Dridan because the results reported in Dridan and Oepen (2012) are against the full PTB.

[11]For this experiment, we again could rely on data made available to us by Rebecca Dridan, produced in a manner exactly parallel to her WSJ test data: a raw-text version of the Brown Corpus was aligned with the gold-standard sentence and token boundaries available in the 1999 release of the PTB (LDC #1999T42).

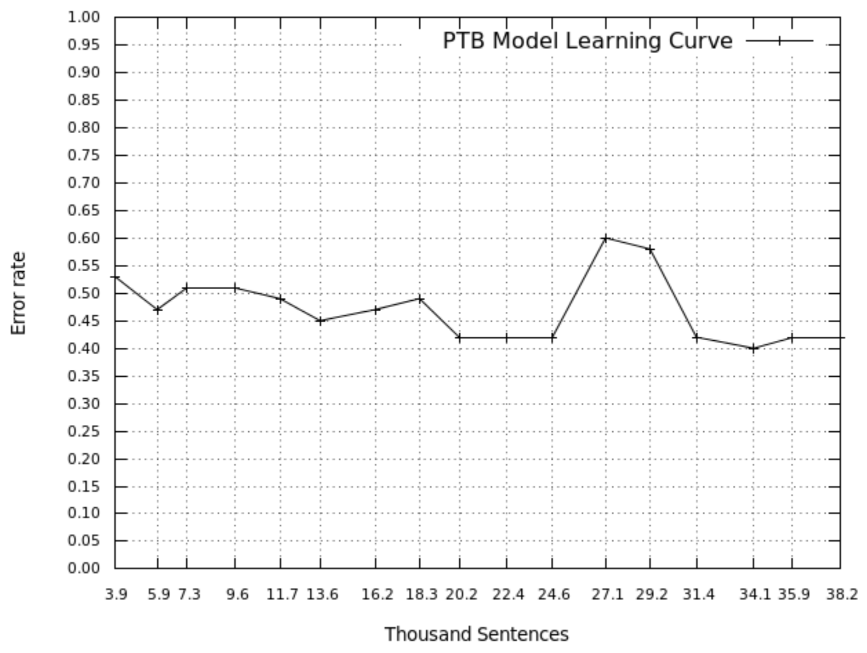


Figure 3.1: The learning curve of the PTB model (tested on PTB WSJ).

strictly follow the PTB tokenization scheme, there are many sentences where the mapping from the raw text to the tokenization in the treebank cannot be established, mainly because of notorious duplication of punctuation marks like semicolons and question and exclamation marks in the PTB annotations. As both tokenizers, ours and REPP, would be equally affected by these artifacts, we excluded 3,129 Brown sentences showing spurious duplicates from our evaluation.

In this experiment, the ML-based approach (i.e. our PTB model trained with WSJ sections) outperforms the REPP tokenizer by a good margin: it delivers a sentence error rate of **0.48%**, contrasting with **2.87%** for the REPP rules of Dridan and Oepen (2012). While for the ML-based tokenizer, performance on Brown is comparable to that on the WSJ text, the REPP tokenizer appears far less resilient to the genre variation. Although speculative, we conjecture that REPP's premium performance in our WSJ tests may in part be owed to tuning against this very data set, as discussed in Dridan and Oepen (2012) and evidenced by the explicit coding for sentence-final 'U.S.', for example.

Furthermore, we experiment with tokenizing out-of-domain texts from the GENIA Corpus. Using 90% of its sentences for training and the rest for testing, we carry out a new set of experiments: First, testing 'pure' PTB- or GENIA-only models on GENIA text; second, training and testing adapted models combining the PTB with either a small portion of GENIA (992 sentences; dubbed PTB+GENIA(1k)), or the entire GENIA (dubbed PTB+GENIA). The results of these

Table 3.3: The results of the GENIA experiments.

| Model | Error rate |
|---|---|
| REPP Tokenizer | 8.48% |
| PTB | 25.64% |
| GENIA | 2.41% |
| PTB+GENIA(1k) | 3.36% |
| PTB+GENIA | 2.36% |

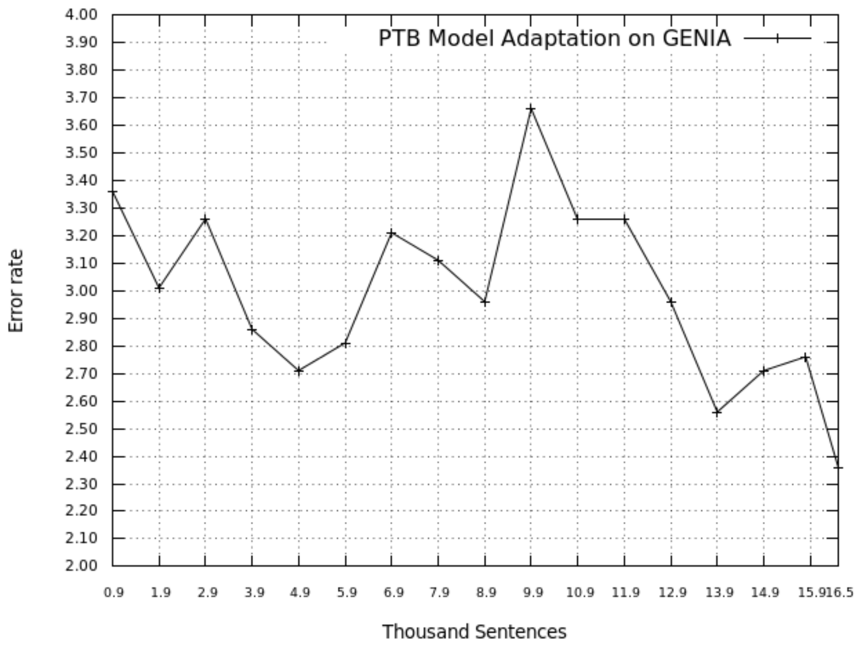


Figure 3.2: Learning curve of retraining the PTB model on GENIA corpus (tested on GENIA).

experiments are presented in Table 3.3 together with the result of testing the REPP tokenizer on GENIA.

From the results above, we see that the rule-based REPP tokenizer outperforms the unadapted PTB model. However, we also see that domain adaptation works quite well even with limited in-domain annotation. We further contrast the learning curve of the adapted model with that of the PTB in-domain experiment (see Figure 3.2). The extended training with more in-domain annotation continues to improve the tokenization accuracy, contributing to the best performing tokenizer using all available annotations.

## 3.4 ERG-Style Tokenization

For experimentation with ERG-style tokenization, we rely on the Redwoods and DeepBank Treebanks introduced in § 2.6. However, over the course of doing these experiments two versions of DeepBank were available, viz. version 0.9 and version 1.0. Using the Redwoods and DeepBank v0.9, we deployed the standard splits of the treebank (as published with the ERG), for 55,867 sentences of training data, and 5,012 and 5,967 sentences for development and testing, respectively. Additionally, we carried out a 'pure' DeepBank experiment with only DeepBank v1.0 data (we will come back to the data split later when we present the results).

Some of the Redwoods texts include residual markup, which the ERG interprets, for example using italics as a cue in recognizing foreign-language phrases in Wikipedia. To focus on the 'pure' linguistic content of the text, our experiments on ERG-style tokenization actually start from a pre-tokenized, markup-free variant recorded in the treebank, essentially the result of a very early stage of parsing with the ERG (Adolphs et al., 2008). As discussed in § 2.3, in ERG terminology, this tokenization is dubbed *initial* tokenization, and it actually follows quite closely the PTB scheme. During lexical analysis, the ERG parser combines the 'token mapping' rules of Adolphs et al. (2008) with lexical look-up in the grammar-internal lexicon to arrive at the ERG-internal tokenization that we characterized in § 3.2.2 above (dubbed *lexical* tokenization). In standard ERG parsing, this mapping from initial to lexical tokenization is a by-product of full syntactic analysis only, but in the following experiments we seek to disambiguate lexical tokenization independent of the parser.

As such, we train a binary classifier (ERG model henceforward) to reproduce ERG-like tokenization (lexical tokens) starting from PTB-compliant initial tokens. As noted in § 3.2.2 above, however, mapping from initial to lexical tokens can require additional candidate splitting points around hyphens and slashes. Somewhat parallel to our notion of sub-tokens in the PTB universe (cf. § 3.3), we employ three heuristic rules to hypothesize additional candidate token boundaries, for example breaking a hyphenated initial token '`well-educated`' into '`well-`' and '`educated`'.[12]

Green et al. (2011) study multi-word expression (MWE) identification using syntactic information and argue that MWEs cannot in general be identified reliably by "surface statistics" only. We largely agree with this point of view, but note that the task of ERG tokenization (as sequence labeling) only comprises a very limited range of MWEs, viz. multi-token lexical entries (see § 3.2.2 above).

[12]Our three splitting heuristics are actually borrowed literally from the 'token mapping' rules that come with the ERG (Adolphs et al., 2008). We also experimented with the introduction of sub-token splits points based on character classes, as in our PTB-style experiments, but found that—given PTB-style initial tokens as our starting point—the more specific heuristics targeting only hyphens and slashes led to fewer candidate boundaries and mildly superior overall results.

Table 3.4: Features used for ERG tokenization classifiers — W: word's surface form, Space: followed by space, $CC_i$: character class, Len: surface form length, FC: first character, LC: last character; † Bigram features, ‡: Unigram and bigram

| **Feature** | **Feature** | **Feature** |
|---|---|---|
| $W_i$‡ | $Space_i$† | $W_{i+1}$ & $CC_{i+1}$‡ |
| $W_{i+1}$‡ | $W_i$ & $Space_i$‡ | $FC_i$‡ |
| $W_{i+2}$‡ | $Space_i$ & $Space_{i+1}$† | $LC_i$‡ |
| $W_{i+3}$‡ | $Space_i$ & $Space_{i-1}$† | $FC_i$ & $FC_{i+1}$ |
| $W_{i-1}$‡ | $CC_i$‡ | $FC_i$ & $FC_{i-1}$ |
| $W_{i-2}$‡ | $CC_i$ & $CC_{i+1}$‡ | $LC_{i-1}$ & $FC_i$ |
| $W_{i-3}$‡ | $CC_i$ & $CC_{i-1}$‡ | $LC_i$ & $FC_{i+1}$ |
| $W_i$ & $W_{i+1}$ & $W_{i+2}$ | $W_{i-1}$ & $CC_{i-1}$‡ | $Len_i$ |
| $W_i$ & $W_{i-1}$ & $W_{i-2}$ | $W_{i-1}$ & $Len_{i-1}$ | $CC_i$ & $Len_i$‡ |
| $W_i$ & $W_{i-1}$ & $W_{i+1}$ | $W_{i+1}$ & $Len_{i+1}$ | $W_i$ & $CC_i$ & $Len_i$† |
| $W_i$ & $W_{i-1}$ & $Len_i$ & $Len_{i-1}$‡ | | |

In analogy to the (practically) circular interdependencies observed between sequence labeling vs. full syntactic analysis for the task of lexical category disambiguation (i.e. part-of-speech tagging), we hypothesize that the arguments of Green et al. (2011) do not apply in full to our ERG tokenization experiments.

### 3.4.1 Results and Error Analysis

Our ERG tokenization model deploys the set of features presented in Table 3.4, complemented by a couple of additional features recording the length, in characters, of adjacent words.

The accuracy of the ERG model, trained, tuned, and tested on the Redwoods and DeepBank v0.9 data sets and splits described above, is **93.88%**, i.e. noticeably below PTB-style tokenization performance, but still rather decent at the level of full sentences. Looking at tokenization mismatches of the ERG model showed that 41% of the errors involve MWEs unseen in training, such as '`a priori`'. The rest of the errors contain ambiguous multi-word lexical units e.g. '`as well as`', which should sometimes be split and sometimes not. Another source of errors are hyphenated multi-word lexical units, such as '`south-west`', which our ERG model regards as a hyphenated word and hence wrongly splits into two tokens.

To try and compensate for the higher error rate in ERG tokenization, we investigate possibilities and trade-offs in *ambiguous* tokenization, i.e. apply $n$-best (or list Viterbi) CRF decoding to generate lattices representing $n$-best lists of token sequences;

The results in Table 3.5 suggest that quite small values of $n$ lead to substantial accuracy gains, with 5-best decoding reaching sentence accuracies approaching our PTB results.

Table 3.5: N-best list for ERG-style tokenization

| N | Accuracy |
|---|---|
| 1 | 93.88% |
| 2 | 97.97% |
| 3 | 98.77% |
| 4 | 99.02% |
| 5 | 99.16% |

As discussed above, at least for the purpose of pruning the search space of the full ERG parser, $n$-best decoding offers attractive flexibility in trading off accuracy and efficiency, abstractly parallel to work in interweaving sequence labeling for lexical category assignments and structured prediction for syntactic analysis (e.g. Curran et al., 2006; Yoshida et al., 2007).

To gauge cross-domain sensitivity in these results (over the full, relatively diverse Redwoods Corpus), we train and test our ERG model on the WSJ portion of the data only. Using sections 0 – 13 to train, and sections 14 and 15 to test, this WSJ-only ERG model delivers a moderately improved 94.06% per-sentence accuracy; its performance in 2-best mode is 98.79%, however, substantially better than the more general, cross-domain ERG model.

Finally, as more training data became available, DeepBank v1.0, we train and test our WSJ-only ERG model again, but this time training on sections 00–20 from DeepBank and testing on section 21. The accuracy improves to 94.77% with one-best decoding suggesting that with more training data better results might be attained.

## 3.5 Summary

In this chapter we have presented an innovative data-driven approach towards tokenization. We have shown that our sequence labeling models substantially outperform state-of-the-art rule-based systems for PTB-like tokenization. Furthermore, we have shown that domain-adaptable tokenization models can achieve very high accuracies, and again, outperform rule-based systems.

In the ERG tokenization setup, we have shown that multi-word lexical units constitute a major challenge. However, more training data is expected to improve the model's performance.

While a certain degree of feature engineering has been invested during our experiments, we believe that further empirical improvements can be made in the design of the sequence labeler, e.g. using a constrained decoder (informed by the ERG lexicon) for the sequence labeler. Also, extra labels could be introduced in the sequence labeler to achieve certain rewriting operations in a sense similar to those of a finite-state transducer, e.g. adding special a label to replicate the `PTB` 'U.S.' idiosyncrasy. These remain to be investigated in future work.

Examining the generalizations of our models and mismatches against gold-standard annotations, already has proven a useful technique in the identification of inconsistencies and errors within existing resources. Hence, we also see a possible perspective on this work as a feedback mechanism to resource creation, i.e. error detection in annotated corpora.

Finally, we have already published our tokenization experiments results in Fares et al. (2013), and for replicability and general uptake, the tokenization toolkit and models will be made available as open source software.

# Chapter 4

# Lexical Categorization

Lexical categories are very effective features for parsers, as they can be used to annotate the input of the parser allowing it to treat unknown words according to their lexical categories. Additionally, lexical categorization information can be exploited to prune parse forests or disambiguate the output of the parser (cf. Chapter 5). Other applications of lexical categorization include noun phrase chunking (Shen & Joshi, 2003), semantic role labeling (J. Chen & Rambow, 2003) and statistical machine translation (Hassan et al., 2007).

In this chapter, we present a selection of experiments on the use of CRFs to learn the ERG lexical categories. First, we define lexical categories in the ERG realm. Then, we sketch out the dimensions of our experiments. We closely study which features are most useful in training a sequence labeling model for lexical categorization. To gauge the accuracy vs. granularity view, we experiment with using only the ERG major syntactic categories, a very coarse-grained generalization of lexical categories. Finally, as CRFs are very expensive to train, we suggest an efficient division of labor through training eleven separate, but interacting, models to learn the ERG lexical types.

## 4.1 ERG Lexical Categories

In Chapter 2 we introduced lexical categories in general; in this section, however, we focus on the definition of lexical categories in the realm of the HPSG-based ERG. As evidenced by earlier studies, Toutanova et al. (2002); Dridan (2009); Ytrestøl (2012) inter alios, defining lexical categories in the ERG, and HPSG in general, is not as clear-cut as in the LTAG and CCG formalisms[1]. Thus, the ERG allows flexible design choices in defining lexical categories, e.g. balancing the accuracy and linguistic granularity, or as Dridan (2009) refers to it, balancing 'predictability' and 'usefulness'. In the following we introduce these choices by explaining the ERG grammar components that can make up lexical categories.

[1]In LTAG the so-called elementary trees directly correspond to the concept of supertags, and so do the categories of CCG.

(1) think_of := (2) v_pp_e_le &

$$\begin{bmatrix} \text{ORTH} & (3)\langle\text{“think”}\rangle \\ \text{SYNSEM} & \begin{bmatrix} \text{LKEYS} & \begin{bmatrix} \text{–COMPKEY} & (4)\ \text{\_of\_p\_sel\_rel} \\ \text{KEYREL.PRED} & (5)\ \text{“\_think\_v\_of\_rel”} \end{bmatrix} \\ \text{PHON.ONSET} & (6)\ \text{con} \end{bmatrix} \end{bmatrix}$$

Figure 4.1: A lexical entry in the `ERG` lexicon – *think_of*

**Lexical Entry** The lexical entry constitutes the basic unit within the `ERG` lexicon. Figure 4.1 shows an example of a lexical entry from the lexicon of the 2012 version of `ERG`. Each lexical entry consists of: (1) an identifier (2) a lexical type (3) a stem (4) an optional selectional relation (5) a semantic predicate and (6) phonetic information.

**Lexical Type** The lexical type (letype) encodes some linguistic information about the lexical entry, such as its syntactic category and subcategorization frame. All lexical types in the `ERG` consist of four fields as follows:

`⟨major-syntactic-cat⟩_⟨subcategorization⟩_⟨description⟩_le`

Where:

1. Major syntactic category (MSC): one of eleven broad lexical categories, such as verb, noun, adjective and adverb.

2. Subcategorization: describes the possible arguments, specifiers or complements, for which a lexical entry selects.

3. Description: an optional field that provides annotations of the finer-grained distinctions among types with the same major syntactic category and complement selection (subcategorization frame).

4. le: a constant suffix to facilitate regular-expression searches within the grammar source files.

We can now explain the lexical type `v_pp_e_le` in Figure 4.1: the first field (`v`) describes a verbal lexical entry, the second field (`pp`) means that this lexical entry selects for a prepositional phrase complement, and the third field (`e`) specifies that the preposition (`pp`, in the second field) is semantically empty.

It is worth observing that the lexical types in the `ERG` describe lexical entries, not words; or said differently, the example in Figure 4.1 represents all inflected forms of the verb ‘think’, such as ‘thought’ and ‘thinks’.

Table 4.1: Lexical categories in Dridan (2009) — Note that Dridan (2009) used the term POS instead of MSC

| **Lexical Category** | **Tagset Size** | **Description** |
|---|---|---|
| letype+sel+morph | 1217 | Lexical type, selectional relation & morphological rules |
| letype+sel | 803 | Lexical type & selectional relation |
| letype+morph | 996 | Lexical type & morphological rules |
| letype | 676 | Lexical type |
| subcat+morph | 254 | Subcategorization & morphological rules |
| subcat | 110 | subcategorization |
| MSC+morph | 36 | Major syntactic category & morphological rules |
| MSC | 13 | Major syntactic category |

**Lexical Item** As just mentioned, 'think', 'thought' and 'thinks' all map to the same lexical entry (in Figure 4.1). However, in the morphological analysis phase each form triggers different lexical rules (morphological and punctuation 'inflection' rules) which are applied to the lexical entry to form the so-called lexical item. Hence, different lexical rules are combined with the lexical entry to represent each of the inflected forms; the lexical item of 'thought', for example, includes a lexical type, `v_pp_e_le`, and a lexical rule, `v_pst_olr` which indicates that 'thought' is a verb in the past tense.

Now that we have presented possible definitions of lexical categories in the ERG, we can review the choice of ERG lexical categories in previous studies. Toutanova et al. (2002) used the ERG lexical entries (i.e. their unique identifier, (1) in Figure 4.1) as lexical categories in an approach for parse disambiguation.[2] This resulted in a very fine-grained tagset of 8,000 lexical entries, which was the size of the ERG lexicon at the time of their study. If we were to opt for the same choice of lexical categories, our tagset would consist of 38,500 lexical entries (the size of the lexicon in ERG1212).

Dridan (2009) defined eight degrees of granularity in lexical categories over the 2009 version of the ERG, summarized in Table 4.1, where the tagset size here reflects the number of tags seen in the training data. First, she identified four types of lexical categories based on syntactic information, `letype+sel`, `letype`, `subcat` and MSC. Each 'syntactic' granularity, then, can be combined with morphological information `+morph` (i.e. can be used with and without adding morphological rules) which leads to eight types of lexical categories. Observe that in Dridan (2009) the number of lexical types seen in the training data is 676.

Ytrestøl (2012) found that lexical types (as lexical categories) provide

[2]Toutanova et al. (2002) used the term *lexical labels* for lexical entries.

the "best possible balance" between restricting the parser's search space and achieving high supertagging accuracy. In the 2011 version of the `ERG`, which Ytrestøl (2012) used, the set of lexical types consists of almost 1,000 types.[3]

## 4.2 Experimental Setup

In this thesis, we define two main granularities of lexical categories. First, the *Lexical Types* (letype) a very fine-grained set of lexical categories. Second, the *Major Syntactic Categories* (MSC) a very coarse-grained set of lexical categories. Experimenting with such starkly different tagsets allows defining: (a) the relation between linguistic granularity and accuracy; (b) the scalability of `CRF`s to large-scale tagging tasks; and (c) the impact of linguistic granularity on restricting the parser search space (in Chapter 5).

Somewhat parallel to our tokenization experiments (§ 3.2.3), we identify five dimensions on which `ERG` lexical categorization experiments might vary:

1. **Grammar**: The grammar version from which the lexical categories are derived.

2. **Observations**: The instances (or minimal processing units) to which the labeling model is supposed to assign lexical categories.

3. **Lexical categories**: The set of labels (tagset), which could be the set of: lexical entries, lexical types, lexical items or major syntactic categories.

4. **Learning model**: The machine learning model and the feature template used to learn the lexical categories.

5. **Data set**: The train, development and test data sets.

In order to better understand our experimental setup, Table 4.2 provides a comparison between the works of Dridan (2009), Ytrestøl (2012) and ours, based on the five dimensions just described.

Observe that Dridan (2009) simplified the problem by choosing initial tokens (`PTB`-compliant) instead of lexical tokens (`ERG`-compliant). In her setup, a multi-word lexical token such as '`at least`' would be split into two initial tokens '`at`' and '`least`'. Then, she would assign the adverbial lexical type of '`at least`' to the preposition '`to`' and the adverb '`least`', which is potentially confusing to the tagger.

[3]Ytrestøl (2012) didn't specify whether 1,000 is the number of lexical types in the `ERG` version he was using or the number of lexical types seen in the training data. Nonetheless, we speculate that almost all lexical types in the `ERG` would be seen in his training data set which contained 140 million words.

Table 4.2: A comparison between Dridan (2009), Ytrestøl (2012) and our work

| | Dridan (2009) | Ytrestøl (2012) | Our experiments |
|---|---|---|---|
| **Grammar** | ERG 2009 | ERG 2011 | ERG 2012 |
| **Observations** | Initial tokens | Lexical tokens | Lexical tokens |
| **Lexical categories** | Table 4.1 | letype | letype & MSC |
| **Learning model** | HMM & MaxEnt | MaxEnt & SVM | CRFs |
| **Data set** | Redwoods 2009 | Redwoods 2011 WikiWoods | DeepBank |
| **Train set** (# tokens) | 157,920 | 141,893,437 | 656,507 |

Besides the differences we see in Table 4.2, each of the three studies focuses on one aspect of the problem, i.e. varying one of the five dimensions while keeping the rest constant. Dridan (2009) examined the effect of varying the size of the tagset on the syntactic analyzer. Ytrestøl (2012) studied increasing the training data size in order to improve tagging accuracy. As both Dridan (2009) and Ytrestøl (2012) used ready-made taggers, they did not look into feature engineering for lexical categorization. In this project, therefore, we closely investigate the efficacy of various features in learning ERG lexical categories.

The following section details our experiments with developing various feature templates to learn the ERG lexical types.

## 4.3 Lexical Type Experiments

Training a CRF model requires specifying, among other variables, the *data*, *feature* and *label* sets. Throughout all the lexical categorization experiments, we use DeepBank v1.0, sections 0 – 19 for training, section 20 for development and section 21 for testing. We extract the data from the so-called DeepBank profiles (which include, inter alia, the output of the ERG parsing pipeline[4]). The data files contain one word per line, with sentences separated by empty lines (newlines). Each lexical token is represented by its sentence ID, surface form ($W_i$), PTB PoS tag ($T_i$) and lexical category. The PTB PoS tags, however, are assigned to initial tokens (cf. § 2.3), which means that some lexical tokens might have more than one PoS tag or none at all. On the one hand, multi-word expressions and contracted negations consist of more than one initial token, hence they have more than one PoS tag. On the other hand, a hyphenated word is one initial token that maps to two lexical tokens, hence an extra PoS tag is needed. In cases where many initial tokens map to one lexical token, we concatenate all the PoS tags of the initial tokens. However, when one initial token maps to many lexical tokens, we use the PoS tag of the one initial token for all the lexical tokens.

[4]For more information about the ERG profiles http://moin.delph-in.net/ItsdbProfile

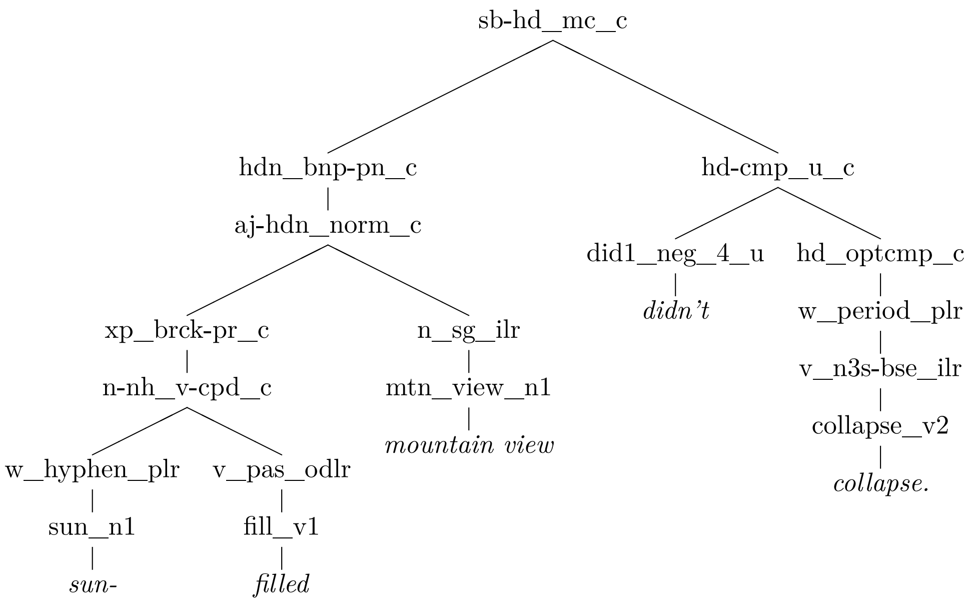


Figure 4.2: The ERG analysis of “Sun-filled Mountain View didn't collapse.”

We clarify the data extraction and representation through an example. Figure 4.2 shows the ERG analysis of the sentence “Sun-filled Mountain View didn't collapse.” The pre-terminals in the derivation tree (Figure 4.2) are lexical entries, and above them are the lexical rules of each token, e.g. ‘filled’ has the lexical entry *fill_v1* and a passivization lexical rule *v_pas_odlr*. The lexical types, however, are not apparent in the derivation tree and are extracted from the lexicon.

Observe that none of the tree leaves correspond to PTB initial tokens, Table 4.3 shows the differences in tokenization and, accordingly, the mapping of PoS tags from initial tokens to lexical tokens; e.g. ‘didn't’ is one lexical token that corresponds to two initial tokens ‘did’ and ‘n't’, hence it gets two PoS tags, ‘VBD’ and ‘JJ’.

Since a lexical type does not encode information about the punctuation marks attached to the token (cf. § 4.1), we normalize all lexical tokens by stripping off the following punctuation marks: *. , : ; ! ?*. Furthermore, in order to reduce the number of word types (by word types we mean unique word forms) we convert all numerals to ‘9’, e.g. 2013 is rewritten as 9999.

The second aspect of training a CRF model is defining the label set. The ERG1212 contains 1,092 lexical types[5], however only 879 of these lexical types occur in our training set. Furthermore, we convert all *generic lexical types*, within the DeepBank, to their corresponding native ones, hence decreasing the

[5]See the Lexical Type Database at http://cypriot.stanford.edu/~danf/cgi-bin/ERG_1212/list.cgi

Table 4.3: Example of initial to lexical tokens mismatch — Note that the PoS tags in this example are automatically assigned.

| | Sun-filled Mountain View didn't collapse. | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PTB tokens | Sun-filled | | Mountain | View | did | n't | collapse | . |
| ERG tokens | Sun- | filled | Mountain View | | didn't | | collapse. | |
| PTB PoS tags | NNP | | NNP | NNP | VBD | JJ | NN | . |
| PoS feats. | NNP | NNP | NNP+NNP | | VBD+JJ | | NN+. | |

number of lexical types in the training set to **857**. The DeepBank was created by automatically parsing the WSJ text, and then manually correcting and disambiguating the output of the parser. One side effect of this methodology, in the ERG realm, is that unknown words would be assigned so-called 'generic lexical entries' (in contrast to known words which are assigned 'native lexical entries'). The generic lexical entries are essentially underspecified lexical entries instantiated to fill the gaps in the lexical chart, i.e. fill the input positions for which no native entries were found. Appendix A lists the mapping rules we apply in the conversion process. Note that a similar normalization was performed in the work of Ytrestøl (2012), whereas Dridan (2009) normalized only four or so generic lexical types.

Now we have our data and label sets ready, but we still need to define the feature set. In the following we introduce a series of feature ablation experiments, where we identify sets of candidate features and then test with some combinations of these sets to arrive at the highest accuracy of lexical types assignment.

### 4.3.1 Feature Ablation Study

As a result of preliminary experiments we define 27 candidate features grouped in four sets based on linguistic criteria as follows:

- Lexical features: encode the surface form of a lexical token in the context of up to four words, in addition to the target word.
- Morphosyntactic features: encode the PTB part-of-speech tags in the context of up to six words.
- Morphological features: encodes the prefixes and suffixes of the target word up to five characters.
- Orthographic features: encode information such as the capitalization of the target word and whether or not it ends with a hyphen.

Table 4.4 presents our 27 features, where $\mathrm{W}_i$ with $i \in [-2\,..\,2]$ denotes the surface form of the $i^{th}$ token relative to the current position, $\mathrm{T}_i$ with

Table 4.4: Candidate features to learn the ERG lexical types

| **Lexical** | **Morphosyntactic** | **Morphological** | **Orthographic** |
|---|---|---|---|
| $W_i$ | $T_i$ | 5-prefix$_i$ | Cap$_i$ & $W_i$ |
| $W_{i-1}$ | $W_i$ & $T_i$ | 5-suffix$_i$ | Cap$_i$ & Cap $_{i-1}$ |
| $W_{i+1}$ | $T_i$ & $T_{i+1}$ | 4-prefix$_i$ | Hyph$_i$ |
| $W_i$ & $W_{i-1}$ & $W_{i-2}$ | $T_i$ & $T_{i-1}$ | 4-suffix$_i$ | |
| $W_i$ & $W_{i+1}$ & $W_{i+2}$ | $T_i$ & $T_{i+2}$ | 3-prefix$_i$ | |
| | $T_i$ & $T_{i-2}$ | 3-suffix$_i$ | |
| | $T_i$ & $T_{i+3}$ | 2-prefix$_i$ | |
| | $T_i$ & $T_{i-3}$ | 2-suffix$_i$ | |
| | $T_i$ & $T_{i+1}$ & $T_{i-1}$ | 1-prefix$_i$ | |
| | | 1-suffix$_i$ | |

Table 4.5: Feature ablation experiments — $\alpha$ trained using 8 threads, $\beta$ trained using 4 threads and $\gamma$ trained using 10 threads

| **Model** | **Accuracy** | **Features size** GB | **Training time** hours |
|---|---|---|---|
| L | 90.37% | 6.83 | $15.24^{\gamma}$ |
| MS | 90.57% | 0.68 | $15.55^{\gamma}$ |
| MS+O | 90.73% | 0.92 | $16.77^{\alpha}$ |
| L+O | 91.35% | 7.06 | $18.46^{\alpha}$ |
| MS+M | 91.37% | 1.17 | $15.59^{\alpha}$ |
| L+M | 92.09% | 7.31 | $17.64^{\gamma}$ |
| L+MS | 92.52% | 7.52 | $20.14^{\alpha}$ |
| L+M+O | 92.33% | 7.55 | $17.45^{\gamma}$ |
| L+MS+O | 92.70% | 7.75 | $17.11^{\gamma}$ |
| L+MS+M | 93.48% | 8.00 | $16.58^{\gamma}$ |
| L+MS+M+O | 93.54% | 8.24 | $49.08^{\beta}$ |

$i \in [-3\,..\,3]$ denotes the PTB PoS tag of the the $i^{th}$ token, 'Cap' and 'Hyph' are binary features which refer to 'Capitalized' and 'Ends with a hyphen', respectively. All of the features in Table 4.4 are unigram features, bigram features, as we shall see in § 4.3.3, are very expensive to deploy.[6]

To study the effect of each feature set, we train eleven models on combinations of these feature sets noting differences in accuracy, training time and main memory (RAM) usage.[7] Table 4.5 shows the results of testing the eleven models on DeepBank section 20, where 'L', 'MS', 'M' and 'O' are 'lexical', 'morphosyntactic', 'morphological' and 'orthographic', respectively.

[6]As described in § 2.4.2, Lavergne et al. (2010) define two types of features, unigram and bigram; unigram features model the dependency between an observed word and a label, while bigram features model the dependencies between successive labels as well.

[7]The training time is comparable only across the models that were trained using the same number of threads. We ran the feature ablation experiments simultaneously using different computational resources, and hence different number of threads.

From Table 4.5, the model that uses all of the feature sets, 'L+MS+M+O', and the 'L+MS+M' model perform best, but their accuracies comes at a price of large memory requirement. Note that the *Feature set size* only reflects the size of the feature vectors, the actual memory requirement to train the 'L+MS+M+O' model, for example, is almost 200 gigabytes.

The orthographic features seem to be most effective when combined with lexical ones. This is evidenced by the accuracies of 'L' vs. 'L+O' and 'MS' vs. 'MS+O'.

Memory-wise, the morphosyntactic (MS) model is the cheapest to train. It is worth observing, however, that the morphosyntactic features are automatically assigned, using the `TnT` tagger (Brants, 2000) which achieves an accuracy of approximately 97% (on `PTB` `WSJ` sections 22 − 24). Thus, these features might be inconsistent or noisy, hence impair the model (cf. § 4.3.2).

The lexical features are the most expensive to encode, due to the large number (37,184) of word types in our train set. The lexical context is limited to four bidirectional window because the number of possible $n$-grams increases with $n$, and their frequency decreases, which leads to higher memory requirements.

Finally, we use the Wilcoxon signed-rank test (cf. § 2.5) to ascertain whether or not the the results of the top three performing models are significantly different. However, as we have only one accuracy observation for each model (Table 4.5), we follow the approach of Spoustová et al. (2009) by dividing the development set into smaller subsets to collect numerous accuracy observations. We split our development set into 36 subsets (47 sentences each), then evaluate the 'L+MS+O', 'L+MS+M' and 'L+MS+M+O' on these subsets. The difference between 'L+MS+O' and 'L+MS+M' is statistically significant, but not between 'L+MS+M' and 'L+MS+M+O' (at significance level $p \leq 0.05$). Although the difference between the 'L+MS+M' and 'L+MS+M+O' models is not statistically significant, in the sequel, we deploy the 'L+MS+M+O' feature set to train our lexical type model (as the difference in memory requirements between the two models is relatively small).

Now that we know the best combination of features, in the following section, we evaluate the 'L+MS+M+O' model on the test set, `DeepBank` section 21, and investigate its classification errors.

### 4.3.2 Results and Error Analysis

Testing the 'L+MS+M+O' model (the lexical type model, henceforth) on `DeepBank` section 21 gives an accuracy of **92.84%**. This level of accuracy, however, is possibly not reliable enough to serve as a front-end for the parser. Therefore, we allow some lexical type ambiguity in order to attain higher accuracies. Table 4.6 presents the results of $n$-best list decoding for the lexical type model.

Table 4.6: N-best list results for the lexical type model on `DeepBank` section 21

| N | Accuracy | Tags per token |
|---|---|---|
| 1 | 92.84% | 1.00 |
| 2 | 94.21% | 1.05 |
| 3 | 95.15% | 1.10 |
| 4 | 95.64% | 1.13 |
| 5 | 96.12% | 1.17 |
| 10 | 97.06% | 1.33 |
| 20 | 97.78% | 1.58 |

The results in Table 4.6 suggest that a limited degree of ambiguity can, in fact, improve the model's accuracy. However, the decoding time does increase with increasing $n$. Hence, in practice, we are exchanging ambiguity *and* efficiency for a higher accuracy.

The error analysis reveals that 18.70% of the lexical type model's errors are unseen words. Additionally, we see two labels in the test set that never occur in the training set, but these are only two instances.

To get a deeper insight into the errors produced by the lexical type model, we manually assess the first one hundred (5%) errors of the model and recognize three types of misclassification errors:

1. `PTB` PoS tag errors: As the `PTB` PoS tags are automatically assigned, we see errors in assigning the lexical types due to mistakes in the `PTB` PoS tags. 8% of the 100 errors are of this type.

2. Inconsistency errors: Errors where the `PTB` PoS is correct but still does not correspond to the lexical type. 9% of the 100 errors assessed are of this type.

3. Classification errors: The errors that the model could not learn or failed to predict and do not fall under the aforementioned types.

In the rest of this section, we present examples of the three error types just described.

Table 4.7 shows a case where mistakenly assigned `PTB` PoS tags confuse the lexical type model. The token '`a little`', for example, is incorrectly labeled as `'d_-_prt-sgm-nag_le'` (instead of `'av_-_dg_le'`) because it was, also incorrectly, assigned the PoS tag `'DT+JJ'`. The gold PoS tag of '`a little`' is `'DT+RB'`, where `'RB'` stands for adverb, and the gold lexical type is an adverb (`'av_-_dg_le'`) as well.

From the training data we see a correlation between '`a little`' as a `'DT+JJ'` and `'d_-_prt-sgm-nag_le'` on the one hand, and between '`a little`' as a `'DT+RB'` and `'av_-_dg_le'` on the other hand. In the follow-

Table 4.7: Example where incorrectly assigned PoS tags lead to incorrect lexical types

| **Words** | **PoS** | **Gold** | **Error** |
|---|---|---|---|
| Consumers | NNS | n_pp_c-of_le | |
| may | MD | v_vp_mdl-p_le | |
| want | VB | v_vp_seq_le | |
| to | TO | cm_vp_to_le | |
| move | VB | v_np_noger_le | |
| their | PRP$ | d_-_poss-their_le | |
| telephones | NNS | n_-_c-ns_le | |
| a little | DT+JJ | av_-_dg_le | d_-_prt-sgm-nag_le |
| closer | RBR | aj_pp-pp_i-cmp_le | |
| to | TO | p_np_i-nnm_le | |
| the | DT | d_-_the_le | |
| TV | NN | n_-_mc_le | |
| set | VBD | n_-_c_le | v_np*_le |

ing sentence, for example, '`a little`' is assigned '`DT+JJ`' as a PoS tag and '`d_-_prt-sgm-nag_le`' as a lexical type.

I sprinkle a little around ...

These correlations explain why the model assigned '`d_-_prt-sgm-nag_le`' instead of '`av_-_dg_le`' to '`a little`' in Table 4.7.

Another error in Table 4.7 is '`set`' which is assigned the PoS tag '`VBD`', hence the model chooses the lexical type '`v_np*_le`', whereas the gold PoS tag is '`NN`' which, if assigned, would help the model predict the correct lexical type '`n_-_c_le`'.

In the second type of errors, what we call inconsistency errors, even though a `PTB` PoS tag is correctly predicted[8] it does not necessarily hint to the `DeepBank` lexical type. The word 'daytime', in the following sentence, is an adjective ('`JJ`') in the `PTB` gold-standard tags, but a noun in the `DeepBank` ('`n_-_c_le`'), however, the lexical type model follows the `PTB` PoS tag and assigns '`aj_-_i_le`' as a lexical type.

Two week ago, viewers of several NBC *daytime* consumer segments ...

The third type of errors, classification errors, include the unseen words. However, a few of the unseen words are in fact seen in the training data but in different forms. The errors we see suggest that removing more punctuation marks would increase the accuracy. The token '`yes`', for instance, never occurs

[8]By correct we mean correct with respect to the `PTB` gold standard, because we see cases where even the `PTB` gold standard does not seem plausible, such as the following sentence: He$_{\texttt{PRP}}$ sits$_{\texttt{VBZ}}$ down$_{\texttt{RP}}$ at$_{\texttt{IN}}$ the$_{\texttt{DT}}$ piano$_{\texttt{NN}}$ and$_{\texttt{CC}}$ plays$_{\texttt{NNS}}$. The word '`plays`' is assigned `NNS` (noun, plural) instead `VBZ` (verb, 3rd person singular present).

as a quoted word in the training set, and hence when it appears quoted in the test set it would be unseen to the model. However, one has to be very careful in removing punctuation marks; the apostrophe, for example, can serve as an individual token and is usually assigned the lexical type `‘n_-_cl-poss-pl_le’`, e.g. *The boss’ book*. Additionally, the apostrophe can be written as a closing single quotation mark, therefore one cannot blindly remove single quotation marks. Moreover, the ampersand (&) or the forward slash (/) can serve as conjunctions, and it would be incorrect to ignore them.

A recurring error, among the classification errors, is confusing transitive with ambitransitive verbs (optionally transitive verbs), `‘v_np*_le’` vs. `‘v_np_le’`.

Other cases are just difficult for the classifier to learn; consider the following sentences:

Contrasts predictably accumulate: First the music ...
“The Fourth Knee Play” an interlude from “Einstein on the Beach” ...

In the first sentence, the token ‘`First`’ is mistakenly assigned `‘n_-_pn_le’` instead of `‘av_-_i-vp_le’`. In the second sentence, however, the gold-standard lexical type of the token ‘`Fourth`’ is `‘n_-_pn_le’` (i.e. a proper noun), but the classifier assigns the lexical type `‘aj_-_i-ord-one_le’`.

For us humans, it is perhaps easy to see the difference between ‘`First`’ in the first sentence and ‘`Fourth`’ in the second sentence, but not for a machine learning model. This is a vexed question for the classifier because both tokens are ordinal words and capitalized.

In the following sentence, all the words are proper nouns, `‘n_-_pn_le’`:

“The Well-Tempered Clavier.”

The model successfully predicts the lexical types of ‘`Well-`’ and ‘`Tempered`’, but incorrectly treats ‘`The`’ as a determiner (`‘d_-_the_le’`), which is the most common lexical type of ‘`The`’. The model was, in a sense, resistant to the capitalization of ‘`The`’, but not the other words in the sentence.

A couple of errors are artifacts of repeating the same PoS tag when an initial token maps to more than one lexical token. For example, the `DeepBank`, following the `ERG` tokenization conventions, splits the initial token ‘`95-37`’ to three tokens ‘`95`’, ‘`-`’ and ‘`37`’, thus we use the PoS tag `‘CD’` for the three lexical tokens. Assigning `‘CD’` to the n-dash `‘-’` confuses the lexical type model, because `‘CD’` is a cardinal number.

The error analysis shows that some classification errors could be avoided by improving the `PTB` PoS tagging. Other classification errors seem to be quite difficult to learn. Hence, in the following sections, we assess whether using bigram features and constrained decoding would help improve the accuracy of the lexical type model.

### 4.3.3 Training with Bigram Features

Towards the end of this project, the *Abel* high-performance computing (HPC) cluster[9] was made available to us for experimentation. We examined the use of bigram features to improve the ERG lexical type assignment accuracy. However, even with one terabyte of main memory we cannot experiment freely with bigram features. The number of feature parameters grows quadratically with the label set size and linearly with the number of values a feature can take. Hence, a bigram feature like the word's surface form ($W_i$) requires 203 gigabytes of memory, because in the DeepBank train set there are 37,184 word forms and 857 labels, which leads to $37,184 \times 857 \times 857 = 27,309,751,616$ features each represented as a double floating point value (8 bytes). Moreover, to train the model with only this one bigram feature (using 12 threads), the Wapiti toolkit requires 450 gigabytes of memory.[10]

Since the cardinality of the morphosyntactic features (PoS tags) is relatively small we experiment with converting $T_i$, in Table 4.4, to bigram feature while keeping the rest as unigram features. The accuracy of the lexical type model increases to **93.06%** (compared to 92.84% with unigram features only). This result improves a bit over the original lexical type model, but with a cost of 81 hours of training and hundreds of gigabytes of main memory.

Due to time constraints, we could not carry out more experiment on using bigram features to judge whether or not additional bigram features would boost the accuracy. Nonetheless, we believe that the computational resource requirements to train CRFs with bigram features are not easily affordable, hence we rather focus on possible division of labor methods to reduce the memory requirement and possibly improve the overall accuracy (§ 4.5).

### 4.3.4 Simulation of Constrained Decoding

In some classification tasks, we might have access to information that would help constrain the choices of the classifier. In concrete terms, one can extract the possible PoS tags of a given word from some dictionary and, accordingly, constrain the PoS tagger in its training and prediction phases.

Previous studies have shown that CRFs can be constrained on two levels, the training level (parameter estimation) and decoding level (inference) (Waszczuk, 2012; Skjærholt, 2011).

Skjærholt (2011) constrained CRFs on the inference level (constrained decoding), while Waszczuk (2012) defined an extension of CRFs named Constrained Conditional Random Fields (CCRFs) which imposes constraints on the training and inference of CRFs.

[9] `http://www.uio.no/english/services/it/research/hpc/abel/`

[10] As mentioned in § 2.4.2, we are using the L-BFGS algorithm for CRF parameter estimation. In Wapiti this algorithm requires $8 \times F \times (5 + M \times 2)$ bytes, where $F$ is the number of features and $M$ is the size of the L-BFGS history, which is 5 by default in Wapiti.

Table 4.8: Constrained decoding for CRF, evaluated on DeepBank section 21

| **Model** | **Per token accuracy** |
|---|---|
| Unconstrained | 92.89% |
| Constrained | 92.99% |

The ERG lexicon provides us with valuable information that could restrict the CRF model. In this section, we simulate constrained decoding for CRFs as a post-processing step. For each lexicon entry, we extract a complete set of candidate lexical types, and for unknown words the set of constraints would be empty. Then, we walk the $n$-best lists of lexical types in decreasing order of probability, and eliminate the lists that violate the set of constraints extracted from the ERG lexicon, until we find a list that conforms to these constraints.

Table 4.8 shows the results of evaluating the constrained lexical type model on DeepBank section 21.[11]

We do not see much of an improvement in the accuracy of the constrained model. The reason, although speculative, may be that the unconstrained model already assigns very small probabilities to ineligible lexical types of a given word. Thus, constrained decoding did not gain the constrained model much in terms of accuracy. However, we believe that actual constraining of the CRF training and decoding would speed up training and inference, as suggested by the recent study of Waszczuk (2012).

### 4.3.5 Comparison with C&C

Our results are not directly comparable to those of Dridan (2009) and Ytrestøl (2012) because they used different versions of the ERG, different data sets, and different machine learning models. However, both Dridan (2009) and Ytrestøl (2012) used the Clark and Curran supertagger (C&C; Clark & Curran, 2007). Therefore, in part to relate our study to theirs and also to compare CRFs to MaxEnt models, we train and test the C&C supertagger on our data set, the DeepBank.[12] Furthermore, in order to make a fair and reliable comparison, we use the C&C feature template, provided in Clark and Curran (2010), to train a CRF-mimicry of C&C. The feature set is presented in Table 4.9 following the same notation we used in our feature ablation study Table 4.4.

We train the CRF model and the C&C supertagger on DeepBank sections $0-19$ and test them on section 21.

As shown in Table 4.10, our CRF model with a replication of the features in the C&C supertagger outperforms the MaxEnt-based C&C supertagger. However,

[11]Due to technical reasons related to ERG parsing, we couldn't extract the lexical types for the words of four sentences (in DeepBank section 21). Hence, we removed them and evaluated the unconstrained lexical type model on DeepBank section 21 without these four sentences.

[12]More accurately, we are comparing implementations of the CRFs and MaxEnt models, Wapiti and C&C.

Table 4.9: The C&C supertagger features

| **Word templates** | **PoS tag templates** |
|---|---|
| $W_i$ | $T_i$ |
| $W_{i-1}$ | $T_{i-1}$ |
| $W_{i-2}$ | $T_{i-2}$ |
| $W_{i+1}$ | $T_{i+1}$ |
| $W_{i+2}$ | $T_{i+2}$ |
| | $T_i$ & $T_{i-1}$ |
| | $T_{i-1}$ & $T_{i-2}$ |
| | $T_{i-1}$ & $T_{i+1}$ |
| | $T_i$ & $T_{i+1}$ |
| | $T_{i+1}$ & $T_{i+2}$ |

Table 4.10: The accuracy of C&C and CRF on DeepBank section 21

| **Model** | **Per token accuracy** |
|---|---|
| C&C | 90.29% |
| CRF | 91.84% |

the training time of the CRF model is remarkably longer than that of the C&C tagger. Training the CRF model takes approximately 50 hours, while the C&C can be trained in 2.5 hours. Therefore, it is possible that with massive amounts of training data the C&C model could outperform the CRF model. Ytrestøl (2012) reports that the learning curve of the C&C supertagger steadily increases with more training data, and our lexical type CRF model (§ 4.3.2) shows the same behavior in Figure 4.3. However, training a CRF model, with 857 labels, on millions of sentences, as Ytrestøl (2012) did for C&C, might be practically infeasible (in fact, as we shall see in § 4.5.2, training a CRF model on large amounts of data is doable only with a relatively small number of labels).

Finally, we train the C&C supertagger with gold-standard major syntactic categories as the PoS tag features ($T$ in Table 4.9) instead of PTB PoS tags. The motivation behind this change is to gauge the potential of an approach similar to *ensemble learning* in which a CRF model would assign major syntactic categories, then these would be used as features for the C&C supertagger. With gold-standard major syntactic categories, the accuracy of C&C increased to 91.19% on DeepBank section 20, compared to 91.06% with PTB PoS tags.[13] If the accuracy were higher, it would have been interesting to use a CRF model to assign major syntactic categories (cf. § 4.4), then employ its output as features for the C&C supertagger which, in turn, assigns lexical types.

[13]Note that the numbers reported in Table 4.10 are on DeepBank section 21.

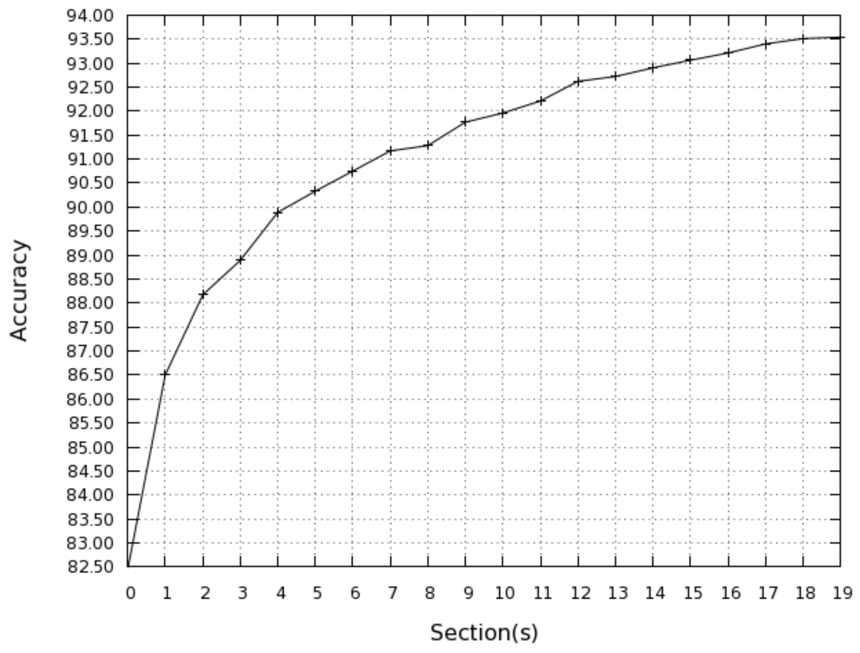


Figure 4.3: The learning curve of the lexical type model (tested on `DeepBank` section 20). Note that zero on the 'Section(s)' axis refers to `DeepBank` section number 0.

## 4.4 Major Syntactic Category Experiments

To gauge the accuracy vs. granularity balance, in this section we seek to trade some linguistic granularity for higher accuracy by choosing the major syntactic categories (MSC) as our lexical categories (tagset).

Again, we need to specify the data, label and feature sets in order to train our MSC `CRF` model. We deploy the same data split as in the lexical type experiments, `DeepBank` sections $0-19$, section 20 and section 21 for training, development and testing, respectively.

The label set consists of the eleven broad syntactic categories defined by the `ERG`, namely: **v**erb (v), **n**oun (n), **a**d**j**ective (aj), **a**d**v**erb (av), **p**reposition (p), **p**repositional **p**hrase (pp), **d**eterminer (d), **c**onjunction (c), **com**plementizer (cm), **punct**uation (pt) and miscellaneous (x).

For the feature set, we start from the features in Table 4.4, but in this experiment it is possible to use bigram features given the small number of labels. After several rounds of feature engineering experiments on the development set, we arrive at the best-performing model trained on the feature set presented in Table 4.11, where 'Punc' indicates whether or not a token ends with a punctuation mark,[14] 'Len' is the length of the token and the rest follow the

[14]Punctuation marks are identified based on the standard implementation of the function

Table 4.11: The set of features used for the major syntactic categories labeler — † Bigram feature

| **Lexical** | **Morphosyntactic** | **Morphological** | **Orthographic** |
|---|---|---|---|
| $W_i$ | $T_i$ | 5-prefix$_i$ | Cap$_i$ & $W_i$ |
| $W_{i-1}$ | $W_i$ & $T_i$† | 5-suffix$_i$ | Cap$_i$ & Cap $_{i-1}$ |
| $W_{i+1}$ | $W_{i+1}$ & $T_{i+1}$ | 4-prefix$_i$ | Hyph$_i$ |
| $W_{i-2}$ | $W_{i-1}$ & $T_{i-1}$ | 4-suffix$_i$ | $W_i$ & Len$_i$ |
| $W_i$ & $W_{i-1}$ & $W_{i-2}$ | $T_i$ & $T_{i+1}$† | 3-prefix$_i$ | |
| $W_i$ & $W_{i+1}$ & $W_{i+2}$ | $T_i$ & $T_{i-1}$† | 3-suffix$_i$ | |
| | $T_i$ & $T_{i+2}$† | 2-prefix$_i$ | |
| | $T_i$ & $T_{i-2}$† | 2-suffix$_i$ | |
| | $T_i$ & $T_{i+1}$ & $T_{i-1}$ | 1-prefix$_i$ | |
| | | 1-suffix$_i$ | |

notation used in Table 4.4.

### 4.4.1 Results and Error Analysis

The MSC model achieves a per token accuracy of **98.01%** on the test set, `DeepBank` section 21.

In the error analysis, we observe that 13.62% of the errors are unseen lexical tokens. We manually check the first one hundred (18%) of the MSC model's errors observing the same types of errors seen in the output of the lexical type model (§ 4.3.2). We find that 19% of the errors can be blamed on incorrect `PTB` PoS tags (errors of the first type). 25% of the errors are due to correctly assigned yet misleading `PTB` PoS tags (errors of the second type).

What is more, in some cases we see that giving up linguistic precision impairs the model; consider the following sentence:

. . . sponsor and a topic: On Mondays . . .

The MSC model incorrectly assigns the category `'p'` to the colon instead of `'pp'`, whereas the lexical type model correctly predicts the lexical type of the colon (hence it assigns the correct major syntactic category as well). The lexical type model assigns the lexical type `'n_pp_c-of_le'` to the word '`topic`' which selects for a `'pp'`, hence when the model assigns a label to the colon, it favors the label `'pp'` over `'p'`. This example directly corresponds to the observation of Brants (1997): "In some cases, finer grained categories of the words in the context deliver the information needed for disambiguation".

Inspired by the case just discussed, we evaluate the accuracy of the lexical type model based only on the correctness of the major syntactic category field of the lexical types it assigns. The lexical type model performs as accurate as the MSC model in assigning major syntactic categories, delivering an accuracy

`ispunct` in C.

Table 4.12: N-best list results for MSC tagging on `DeepBank` section 21

| N | Accuracy | Tags per token |
|---|---|---|
| 1 | 98.01% | 1.00 |
| 2 | 98.97% | 1.05 |
| 3 | 99.36% | 1.10 |
| 4 | 99.46% | 1.14 |
| 5 | 99.57% | 1.18 |
| 10 | 99.74% | 1.37 |
| 20 | 99.89% | 1.71 |

of **98.03%**. Furthermore, we observe that 66% of the errors made by the lexical type model, on the major syntactic category level, are also made by the MSC model, which suggests that finer grained categories might be helpful for at most one third of the errors.

Aiming for a near perfect MSC labeling, we allow limited degrees of ambiguity in assigning MSCs. The results in Table 4.12 show that rather small amounts of ambiguity lead to remarkable accuracy gains: with 5-best decoding the model attains an accuracy of 99.57%.

### 4.4.2 Derivational Morphology: Category-Changing Rules

As described in § 4.1, a lexical item in the `ERG` consists of a lexical entry and lexical rules. Among the lexical rules, there is a set of so-called *category-changing derivational rules* which change the major syntactic category of a word. Consider the following sentence:

Major$_{\texttt{aj}}$ European$_{\texttt{aj}}$ auction$_{\texttt{n}}$ houses$_{\texttt{n}}$ are$_{\texttt{v}}$ turning$_{\texttt{v}}$ increasingly$_{\texttt{av}}$ to$_{\texttt{p}}$ specialized$_{\texttt{v}}$ sales$_{\texttt{n}}$.

The word ‘`specialized`’, in the sentence above, functions as an attributive adjective but in the gold-standard `ERG` analysis it is assigned the major syntactic category `‘v’`. However, by looking at the lexical item of ‘`specialized`’, we see that it includes the derivational rule `‘v_j-nb-pas-tr_dlr’` which indicates that ‘`specialized`’ is an adjective derived from a transitive passive verb. Hence, the major syntactic category of ‘`specialized`’ is converted to `‘aj’` in the lexical parsing stage (cf. § 2.3).

In consultation with the `ERG` developers, we compiled a list of the category-changing lexical rules, and accordingly changed the major syntactic categories in the `DeepBank` in order to judge whether or not the category-changing derivational rules affect the accuracy of the MSC model.

We carry out two experiments, ‘MSC-changed’ and ‘MSC-augmented’. First, in ‘MSC-changed’, we use the basic inventory of major syntactic categories, but with looking at the lexical item (lexical entry plus applications of lexical rules) and adjusting the category when one of the derivational rules on our compiled

Table 4.13: Category changing rules with MSC tagging results (`DeepBank` section 20)

| **Model** | **Accuracy** | **# labels** |
|---|---|---|
| MSC | 98.29% | 11 |
| MSC-changed | 98.12% | 11 |
| MSC-augmented | 98.09% | 16 |

list has applied. When we see multiple applications of category-changing rules, e.g. noun to verb to adjective, we rely on the topmost rule application to determine the category of the lexical item as a whole.

Second, 'MSC-augmented', we add five derived categories to the label set of major syntactic categories, e.g. `'v-n'` verb changed to noun, to see whether such extra granularity helps or hurts in tagging accuracy.

Table 4.13 shows the results of the two experiments. The three models deliver almost the same level of accuracy, when evaluated on `DeepBank` section 20, but still the results in Table 4.13 are not easily comparable, as the labels are not exactly the same in these models. We observe, however, that 80% of the MSC model errors are shared with the MSC-changed model and 87% of the errors made by the MSC model are also made by the MSC-augmented model. Therefore, we speculate that the errors made by these models are independent of the derivational morphological information we exploited. Therefore, we speculate that majority of the errors in assigning major syntactic categories are independent of the derivational morphological information we exploited.

## 4.5 Specified Lexical Type Experiments

`CRF`s are problematically expensive to train and decode with large label sets, such as the `ERG` lexical types. Thus, in this section, we propose a divide-and-conquer strategy to overcome this problem.

Given the pattern of the `ERG` lexical types, we recognize eleven subsets of lexical types depending on the value of their major syntactic category fields. We subdivide the lexical types into eleven mutually exclusive sets each consisting of lexical types that share the same major syntactic category. We call these sets the 'specified lexical type' sets, Table 4.14 presents these sets and the number of lexical types within each, where 'v-letype', for example, denotes the set of lexical types that have a verb as major syntactic category.

For each specified lexical type set, we train a `CRF` model; which means we need to define the data and feature sets of these models. We reuse the feature templates presented in § 4.3 and § 4.4 to train the specified models; more specifically, we use the feature set in Table 4.4 to train the n-letype and v-letype models, and the features in Table 4.11 to train the rest of the specified models.

Table 4.14: The specified lexical type sets

| Specified letype | # lexical types |
|---|---|
| x-letype | 10 |
| cm-letype | 12 |
| d-letype | 40 |
| c-letype | 57 |
| pt-letype | 2 |
| pp-letype | 16 |
| av-letype | 77 |
| aj-letype | 84 |
| p-letype | 62 |
| n-letype | 231 |
| v-letype | 266 |

Table 4.15: Data representation in the specified lexical type models

| Word | letype | n-letype | v-letype |
|---|---|---|---|
| The | `d_-_the_le` | `d` | `d` |
| top | `aj_-_i-att_le` | `aj` | `aj` |
| money | `n_-_mc_le` | `n_-_mc_le` | `n` |
| funds | `n_-_c_le` | `n_-_c_le` | `n` |
| are | `v_prd_are_le` | `v` | `v_prd_are_le` |
| currently | `av_-_i-vp-x_le` | `av` | `av` |
| yielding | `v_np*_le` | `v` | `v_np*_le` |
| well | `av_-_dg-v_le` | `av` | `av` |
| over | `av_-_dg-jo-num_le` | `av` | `av` |
| 9 | `aj_-_i-crd-gen_le` | `aj` | `aj` |
| % | `n_-_c-meas-spr_le` | `n_-_c-meas-spr_le` | `n` |

The data sets, however, need to be reproduced for each specified lexical type model. For a given specified model, the training data consist of two types of training instances; (a) tokens with lexical types that have the same major syntactic category of the model, and (b) tokens with lexical types that have different major syntactic categories from that of the model. We generalize all the lexical types of the instances in (b) by using only their major syntactic categories. Hence, we add 10 labels to each specified lexical type set.

To better understand the data and label sets, Table 4.15 shows the labels of an example sentence in the data sets of the original lexical type model (letype), the noun lexical type model (n-letype) and the verb lexical type model (v-letype).

Observe that only lexical types with an `'n'` major syntactic category are used in the *n-letype* labels, the rest are major syntactic categories. Similarly, in the *v-letype* set, we see only lexical types with a `'v'` major syntactic category.

Now that we have the data, label and feature sets, we train the models

Table 4.16: Accuracy of the specified letype models on `DeepBank` section 20

| **Specified letype** | **Per token accuracy** | **Training time** |
|---|---|---|
| x-letype | 98.34% | 9 mins |
| cm-letype | 98.32% | 13 mins |
| d-letype | 98.33% | 48 mins |
| c-letype | 98.27% | 1.75 hours |
| pt-letype | 98.26% | 6 mins |
| pp-letype | 98.27% | 17 mins |
| av-letype | 98.15% | 2.58 hours |
| aj-letype | 98.21% | 2.71 hours |
| p-letype | 97.06% | 1.60 hours |
| n-letype | 96.87% | 1.88 hours |
| v-letype | 96.58% | 2.20 hours |

on `DeepBank` sections $0-19$, and use sections 20 and 21 for development and testing, respectively. Table 4.16 shows the result of the individual models each evaluated on 'its' development set.

The training times of all the models are substantially shorter than that of the lexical type model (which takes almost 20 hours); this is a distinct advantage of the specified models of which we will make use in § 4.5.2. Although the accuracies in Table 4.16 look relatively good, the the overall accuracy may not be as good, when the errors of the eleven models are combined. In other words, we cannot quantify the gain in accuracy until we combine the outputs of all the models.

When combining the outputs of the eleven specified models, two options arise. First, run all the models in parallel, then merge their outputs, which we will refer to as the 'parallel method'. Second, label the data using the MSC model (§ 4.4), then use the output of the MSC model as partially labelled inputs to the specified lexical type models, we will refer to this setup as the 'MSC-based method'.[15] Table 4.17 shows an example of how the input data would look like in the MSC-based setup.

We experiment with both methods, however, we first need to consider two side effects of such methodologies, as follows:

1. A lexical token might be assigned more than one lexical types. In this case we choose the label of the more accurate model, in the order of Table 4.16. A better policy is, of course, to consider the probabilities of the labels, but since this phenomenon was infrequent we opted for the convenient method. This case only happens in the 'parallel' setup.

[15]Achieving the MSC-based method was relatively easy, thanks to the implementation of 'forced' decoding in Wapiti.

Table 4.17: Example of inputs to the specified lexical type models in the MSC-based setup.

| | It | adds | something | to | the | market. |
|---|---|---|---|---|---|---|
| MSC output | n | v | n | p | d | n |
| n-letype | – | v | – | p | d | – |
| v-letype | n | – | n | p | d | n |
| p-letype | n | v | n | – | d | n |
| d-letype | n | v | n | p | – | n |

Table 4.18: The accuracy of combining the specified lexical type models outputs on DeepBank section 21

| **Experiment** | **Per token accuracy** |
|---|---|
| Lexical type | 92.84% |
| Parallel | 92.29% |
| MSC-based | 92.20% |

2. A lexical token might not get any lexical type, only a major syntactic category. Handling such cases depends on what we take to be our set of lexical categories. If we want pure lexical types, then we count it as an error, even if the major syntactic category is correct. However, if mixed lexical categories are allowed (lexical types and major syntactic categories), we can include the count of correctly assigned MSCs in the total accuracy.

### 4.5.1 Results and Discussion

Table 4.18 shows the results of the 'MSC-based' and 'parallel' experiments on DeepBank section 21.

The results reported do not improve over the original lexical type accuracy, however, the efficiency (memory and run time) of the specified lexical type models is substantially better than that of the lexical type model.

The lexical type model takes 4 minutes to decode DeepBank section 21, while the specified lexical type models need 69 seconds when ran sequentially. Even though the training time might not be as critical as the decoding time, in the lexical type mode, the training time is very expensive to the extent that adding more training data becomes practically unfeasible. However, the training time of all the specified lexical type models allows increasing the size of the training set, hence, potentially improve the accuracy (cf. § 4.5.2).

By investigating the outputs of the two experiments, we see that almost 15% of the 'MSC-based' errors are made by the MSC model. We also notice that 12% of the errors, in the parallel experiment, are tokens that did not get any lexical type, and only 0.3% of all tokens received more than one lexical type. Moreover, 16% of the errors in the MSC-based output get only MSC. Hence, if we consider our lexical category set to contain lexical types and major syntactic categories, the accuracies of the parallel and MSC-based methods would increase to 92.84% and 92.86%, respectively.

As the differences between the results are very small, we test for statistical significance between the lexical type model on the one hand, and the parallel and MSC-based methods on the other hand. We also test for statistical significance between the results of the two methods of combining the specified lexical type models outputs, the parallel and MSC-based methods.

We carry out three rounds of statistical significance testing (using the Wilcoxon signed-rank test, cf. § 2.5), following the approach of Spoustová et al. (2009) by dividing the test set (`DeepBank` section 21) into subsets and evaluating the models on these subsets to get numerous accuray samples. First, we split the test set into 30 subsets (47 sentences each), second, 100 subsets (14 sentences each) and third, 14 subsets (100 sentences each). In all rounds and for all the results, the *p*-values are smaller than 0.05, hence we could reject the null hypothesis that there is no statistically significant difference between the results of the models evaluated.

In the following section, we look into training the specified lexical type models on considerably larger amounts of training data.

### 4.5.2 Indirect Self-Training

Self-training is a semi-supervised machine learning method; one can 'self-train' a supervised classification model (like our lexical categorization models) on training data produced by another supervised model. In this section, we train the specified lexical type models on data generated by a syntactic parser. We refer to this setup as 'indirect self-training' just to distinguish it from self-training as defined in McClosky et al. (2006a).

Many studies investigated (indirect) self-training as a means to boost in-domain and out-of-domain lexical categorization (supertagging) and syntactic parsing (McClosky et al., 2006b; Kummerfeld et al., 2010; Ytrestøl, 2012).

Ytrestøl (2012) trained the `C&C` supertagger on 10 million automatically annotated sentences from the `WikiWoods` Treecache (Flickinger et al., 2010) observing a steady increase in supertagging accuracy. Kummerfeld et al. (2010) used self-training to speed up their parser while maintaining its accuracy. They trained a supertagger on the parser's output in order to make its decisions better suit the parser. Kummerfeld et al. (2010) report 50% increase in in-domain parsing speed and 45% in out-of-domain parsing speed.

Motivated by the relatively cheap computational cost of training the specified lexical type models, we train some of these models on 1.14 million automatically annotated sentences (22.5 million words) from the North American News Text Corpus (`NANC`; cf. § 2.6). We produce the training data by parsing the `NANC` text using the `HPSG` PET parser (Callmeier, 2000).

Table 4.19: Self-training accuracy of the specified lexical type models on `DeepBank` section 20

| | **p-letype** | **n-letype** |
|---|---|---|
| `DeepBank` | 97.06% | 96.87% |
| `NANC` | 96.65% | 95.28% |

Table 4.20: Self-training accuracy of the specified lexical type models on *cb*

| | **p-letype** | **n-letype** |
|---|---|---|
| `DeepBank` | 93.32% | 91.73% |
| `NANC` | 96.02% | 95.26% |

The feature sets remain as previously defined for all the specified lexical type models. It is worth observing, however, that the number of labels (lexical types) seen in the `NANC` training data grows to 1,000, an increase of 15% over the number of lexical types seen in `DeepBank`, which makes the classification problem more difficult.

Due to time constraints (cf. below), we trained only two specified lexical type models, n-letype and p-letype. To Train both models on 22.5 million words using 10 threads, the p-letype model takes 64 hours, while the n-letype model takes 88 hours.

Table 4.19 presents the results of evaluating the `NANC` p-letype and n-letype models on `DeepBank` section 20, together with the results of `DeepBank` models from § 4.3.

The models trained only on `NANC` data achieve lower accuracies than the ones trained only on `DeepBank`, however still not significantly lower given the fact that `NANC` data is automatically annotated. This little drop in the accuracy likely is due to the higher number of labels in the `NANC` model. Looking at the errors of the `NANC` n-letype model and the `DeepBank` n-letype model, we see that they share only 40% of the errors.

At this point, we cannot predict the overall accuracy of the eleven `NANC` specified models from the n-letype and p-letype models, especially as we see different types of errors from the ones we see in the `DeepBank`-only models (as just mentioned). In other words, it might be the case that the eleven `NANC` models have a lot of errors in common, hence the overall accuracy might be better than that of the `DeepBank` models.

We evaluate the models on out-of-domain data using a technical essay on software engineering methods, *The Cathedral and the Bazaar* (`cb`)[16], which is part of the `Redwoods` test set (cf. § 2.6). Table 4.20 presents the accuracies of the `DeepBank` n-letype and p-letype models and `NANC` n-letype and p-letype models on `cb`.

[16]`http://catb.org/esr/writings/cathedral-bazaar/`

Table 4.20 reflects an evident advantage of using large amounts of training data, even though automatically annotated. We see a sharp increase in the accuracies of both models when trained on NANC data. One explanation for these differences in the accuracy is that about 30% of the words in cb are unseen to the DeepBank models. Hence, it is plausible that with much more training data we could fill some of this lexical gap.

Our results confirm the findings of Ytrestøl (2012). Training on very large amounts of automatically annotated data can achieve competitive results in comparison to training on gold-standard data; in fact, we attained more accurate results on out-of-domain data using automatically annotated data. Moreover, our comparison was a little unfair (to the NANC models) because the number of labels in the NANC models is 1,000 whereas in the DeepBank models it is 857. One immediate suggestion would be to try constrained decoding on the NANC models, but this remains for future work.

### 4.5.3 Tiered CRFs

Towards the very end of this project, we came across a contemporaneous study by Radziszewski (2013) which presents an approach to morphosyntactic tagging for Polish that bears a lot of similarities to our specified lexical types. Radziszewski (2013) argues that performing morphosyntactic analysis for Polish, whose tagset contains over 1000 different tags (Przepiórkowski, 2005), is rather difficult with a single CRF model. Hence, he proposes using tiered CRFs, which to a large degree resembles our specified lexical type approach.

Radziszewski (2013) carried out his experiments using the National Corpus of Polish (Przepiórkowski et al., 2010), where the tags adhere to a specific pattern. Each tag consists of: (a) grammatical class (a generalization of the PoS notion) which corresponds to the major syntactic category in ERG, and (b) set of attributes depending on the value of the grammatical class. For example, the grammatical class noun is specified with attributes such as number, gender, and case. A concrete example is subst:sg:nom:m2 which denotes an animate masculine (m2), singular (sg) noun (subst) in a nominative form (nom) (Radziszewski, 2013). The first tier of CRFs selects the grammatical class. Accordingly, the subsequent models assign attributes to the grammatical class selected by the first CRF model.

As such, the tags and the strategy look very similar to the ERG lexical types and our specified lexical type approach. However, there are two major differences. First, the ERG lexical types encode syntactic information, rather than localized morphosyntactic and morphological information, making the task of predicting ERG lexical types more complex and challenging. Second, the morphosyntactic analyis in Radziszewski (2013) relies on a morphological analyzer, hence it assumes the input as a list of tokens each assigned a set of possible morphosyntactic interpretations, which is, more or less, like constraining the CRF models.

## 4.6 Summary

This chapter presented rather numerous lexical categorization experiments with different setups. Therefore, it is worthwhile summing up the numbers and tables we have shown.

We started the chapter by a theoretical introduction of lexical categories and categorization in the ERG realm. Following that, we situated our work with respect to Dridan (2009) and Ytrestøl (2012), showing novelty in our: (a) use of CRFs as a sequence labeler, (b) feature ablation studies and (c) specified lexical type approach.

Then we explored the ERG lexical categorization through three 'tracks' of experimentation based on the granularity of the lexical categories. In the lexical type experiments, we showed that using lexical, morphosyntactic, morphological and orthographic features delivers the best accuracy of assigning ERG lexical types (92.84% on DeepBank section 21). Most notably, the manual error analysis of 5% of the classification errors shows that 8% of the errors can be blamed on the automatic assignment of PTB PoS tags and 18% are unknown words. We also investigated the use of CRF bigram features in training our lexical type model, but it turned out not to be cost-effective. Then, we simulated constrained decoding for our CRF lexical type model, even though the accuracy didn't noticeably improve, we still believe that there is a potential for improvement if constrained CRFs, as presented in Waszczuk (2012), were implemented. The CRF-mimicry of the C&C supertagger proved to be more accurate than C&C, however with the cost of longer training times.

In the major syntactic category experiments, our model landed an accuracy of 98.01%; with 5-best lists decoding we attain 99.57%. We demonstrated that changing the MSCs according to the category-changing derivational rules doesn't affect the accuracy.

We presented *specified lexical types* to speed up the training process of our CRF models. In this approach, we greatly reduced the training time and memory requirements to the extent that training on 22.5 million words became feasible. We showed that training some of our specified lexical type models on automatically annotated data might improve in-domain accuracy and certainly improves out-of-domain accuracy.

All in all, when the size of the training data is relatively limited, our CRF model outperforms the MaxEnt-based C&C. By relatively limited amount of training data we mean in comparison to the massive amounts of automatically annotated training data Ytrestøl (2012) used in training C&C achieving an accuracy of about 95%. However, Ytrestøl (2012) trained the C&C supertagger for one month, hence we also have to train CRFs for one month to allow a fair comparison.

# Chapter 5

# Integration

This thesis approaches tokenization and lexical categorization as stand-alone problems. Its ultimate goal, however, is to improve subsequent NLP applications through improving tokenization and lexical categorization. In this chapter, we discuss integrating the models developed in Chapter 3 and Chapter 4 within the syntactic parsing task (parser integration).

In § 5.1 we introduce candidate approaches for parser integration. In § 5.2 we explain our experimental setup. In § 5.3 and § 5.4 we introduce the results of using our tokenization and lexical categorization models to restrict the parser search space.

## 5.1 Introduction and Related Work

In Chapter 2, we mentioned two main approaches to integrate lexical categories with parsers, *hard constraints* and *soft constraints*.

In the discipline of operations research, a hard constraint is a constraint that any solution *must* satisfy, whereas a soft constraint is a cost function that the solution seeks to maximize or minimize. In a similar spirit, we can define hard and soft constraints on syntactic parsing: the parser must choose syntactic analyses that satisfy the hard constraints and it should prefer analyses that maximize (or minimize) the probability of the soft constraints.

Previous studies have investigated using lexical categories as both hard and soft constraints on the syntactic parsing task. In the following, we review some of these studies as candidate methods to integrate information from our tokenization and lexical categorization models.

Ninomiya et al. (2006) used lexical categories (supertags) as soft constraints on parse ranking; they ranked the parse trees only in terms of supertagging probabilities. Lexical categories can also be incorporated into parse ranking and disambiguation models as auxiliary distribution features. van Noord (2007) shows that using auxiliary distribution features in a parse selection model improves parsing accuracy.

Sarkar (2007) used LTAG supertags as hard constraints to reduce syntactic lexical ambiguity, hence improve parsing efficiency. Similarly, Dridan (2009) restricted the parser search space by using supertags to decide what lexical entries can be added to the parsing chart. In both studies, parsing efficiency increases but at the cost of decreased coverage, even with $n$-best lexical categories. However, both studies suggested ways to regain the lost coverage. We return to the findings of Dridan (2009) later as they are most relevant to our work.

Clark and Curran (2004) introduced an adaptive supertagging approach for CCG parsing. The supertagger provides the parser with lexical categories based on their probabilities (within a specific threshold $\beta$). When the parser fails to find an analysis, the threshold ($\beta$) is relaxed. This approach prefers efficiency to accuracy or coverage, because the supertagger is very restrictive unless the parser fails to find an analysis. If accuracy or coverage is preferred to speed, then one would consider the approach of Auli and Lopez (2011), where they suggest reversing the adaptive supertagging approach of Clark and Curran (2004) by pruning the parse search space only when it becomes impractically large, i.e. when the size of the parse chart exceeds some predefined limit.

Dridan (2009), inter alios, presents a view on using lexical categories for handling unknown words to the parser. One downside of lexicalized parsers is the data sparsity problem; the ERG lexicon, for example, includes a finite number of lexical entries, hence there will always be unseen words. Hence, lexical categories can be exploited to leverage the lexical resources of the parser, by using them to annotate the input of the parser.

All of the reviewed approaches are applicable to the ERG parsing pipeline. In the following section, we introduce our integration strategy for lexical categories and we also gauge the benefits of using ERG token boundaries as hard constraints on the parser.

## 5.2 Experimental Setup

The previous section presented several approaches to integrate lexical information into parsing systems. In this thesis, we use the outputs from our tokenization and lexical categorization models to impose hard constraints on the ERG parsing pipeline, i.e. restricting which lexical items are added to the parsing chart. The on-going developments of the emerging version of ERG allowed integrating the output of our lexical categorization models as lexical filtering rules (cf. § 2.3) without any code changes. These developments in ERG allow using so-called micro-tokens (modified initial tokens, cf. § 3.4) as input to the parser, whereas the standard parsing setup in ERG1212 assumes an input of initial tokens.

We use the Grammar Markup Language (GML; Read et al., 2013) to annotate the input to the parser, as illustrated in Table 5.1. Then we format

Table 5.1: Data annotation with GML — (1) raw text, (2) token boundaries, (3) token boundaries and MSCs and (4) ambiguous MSCs

| | |
|---|---|
| (1) | Ms. McCraw says the magazine is fighting back. |
| (2) | ⌊ ⌋Ms.⌊ ⌋McCraw⌊ ⌋says⌊ ⌋the⌊ ⌋magazine⌊ ⌋is⌊ ⌋fighting⌊ ⌋back.⌊ ⌋ |
| (3) | ⌊ ⌋Ms.⌊ ¦n⌋McCraw⌊ ¦n⌋says⌊ ¦v⌋the⌊ ¦d⌋magazine⌊ ¦n⌋is⌊ ¦v⌋fighting⌊ ¦v⌋back.⌊ ¦pp⌋ |
| (4) | ⌊ ⌋Ms.⌊ ¦n⌋McCraw⌊ ¦n⌋says⌊ ¦v⌋the⌊ ¦d⌋magazine⌊ ¦n⌋is⌊ ¦v⌋fighting⌊ ¦v⌋back.⌊ ¦pp\|n⌋ |

our data files in compliance with the so-called YY input format of the PET parser.[1]

The filtering of token boundaries is somewhat 'conservative', meaning that we should provide the left and right boundaries of any token retained for parsing, but extra token boundaries are also allowed, e.g. adding a token boundary inside a multi-word lexical entry (⌊ ⌋Mountain⌊ ⌋View⌊ ⌋) would still allow the multi-word expression to be recognized as a single token. This conservative filtering enables $n$-best ambiguous token boundaries, hence help reduce loss in coverage.

In lexical category filtering, for each token, if its right token boundary carries major syntactic category or lexical type annotation, then the corresponding lexical items can only be added to the parsing chart if their category is compatible with the annotation. In concrete terms, in line (2) Table 5.1, none of the token boundaries carries lexical category annotation, hence all corresponding lexical items would be considered in parsing. However, in line (3) Table 5.1, each right token boundary contains an MSC annotation which is used in lexical pruning, e.g. the token '`back`' is annotated with a '`pp`' major syntactic category, hence only lexical items with '`pp`' would be added to the parsing chart.

Our integration choice is primarily motivated by the fact that more efficient `ERG` parsing is needed. Today, the `ERG` parser (PET) uses 20 seconds, on average, to analyze a sentence (cf. following sections); `ERG` is continuously enriched to achieve higher accuracy and coverage scores, making the parsing task more computationally expensive.

In the following experiments, we try to quantify the benefits of using lexical categories and token boundaries to the `ERG` parser in terms of coverage, efficiency and accuracy. Coverage is the proportion of sentences which receive at least one syntactic analysis. Efficiency is the average parsing time per sentence. Accuracy is evaluated in two ways: (a) exact matches, the number of parse analyses that exactly match the gold-standard ones, and (b) PARSEVAL metric (Black et al., 1991) which, in simple terms, splits the output and gold parse trees into two sets of constituents (labeled brackets) and computes the overlap between these sets.

The coverage, efficiency and accuracy results reported in this chapter were measured using the `[incr tsdb()]` performance profiling tool (Oepen &

[1]`http://moin.delph-in.net/PetInput`

Table 5.2: Parsing efficiency, coverage and accuracy using ambiguous token boundaries, gold-standard token boundaries and automatically assigned token boundaries.

| | **Efficiency** | **Coverage** | **Accuracy** | |
|---|---|---|---|---|
| | Seconds | % | Exact matches | PARSEVAL |
| All | 20.61 | 97.3 | 339 | 87.2 |
| Gold TB | 19.00 | 97.6 | 345 | 87.8 |
| TB | 18.81 | 97.3 | 339 | 87.5 |

Flickinger, 1998; Oepen & Callmeier, 2000).[2]

Finally, all of the experiments in the following sections are evaluated on `DeepBank` section 21, totalling 1,389 sentences.[3] In addition, we run the parser in the treebank development mode, meaning that the parser would time-out after four minutes if it failed to find an analysis.

## 5.3 Token Boundaries Integration

As we explained in Chapter 3, all of the recent studies on `ERG` lexical categorization and parsing either assume gold-standard `ERG` token boundaries or operate off an ambiguous lattice of token boundaries. The main goal of our `ERG` tokenization model (§ 3.4) is to enable `ERG` lexical categorization. However, in this section we investigate whether or not the `ERG` tokenization model would also help improve parsing.

We compare the accuracy, efficiency and coverage of the parser in two configurations. First, unrestricted parsing with fully ambiguous token boundaries (i.e. with all possible token boundaries), and second, restricted parsing with token boundaries provided by our `ERG` tokenization model. The upper bound for this experiment is parsing with gold-standard token boundaries. Note that this upper bound is only for coverage and accuracy, not efficiency, because the automatically assigned token boundaries might be more restrictive than the gold-standard ones, hence they can achieve higher efficiency than gold-standard token boundaries.

Table 5.2 shows the efficiency, coverage and accuracy of the `ERG` parser using ambiguous token boundaries (All), gold-standard token boundaries (Gold TB) and automatically assigned token boundaries (TB).

With ambiguous token boundaries, the parser takes 20.68 seconds, on average, to analyze a sentence. Hence, the efficiency numbers in Table 5.2 can be translated as reductions of parsing time by: 7.8% with gold-standard token

[2]`http://www.delph-in.net/itsdb/`

[3]Note that `DeepBank` section 21 contains 1,414 sentences of which we excluded 25 because their gold-standard analyses in the current version of `ERG` were unavailable in our experimentation setup where the input of the parser consists of micro-tokens instead of initial tokens.

boundaries and 8.7% with automatically assigned token boundaries. There is no noticeable differences in coverage between the three configurations. The differences in accuracy are also very small.

The results in Table 5.2 suggest that: (a) there is small gains in parsing efficiency from integrating ERG token boundaries with parsing, and (b) these gains are similar when using gold-standard and automatically assigned token boundaries. These results prove that, in practice, our ERG tokenization model is good enough to enable ERG lexical categorization and parsing. Possibly, more improvements in parsing can be attained by using *n*-best token boundaries, however, this shall be investigated in future work.

In the rest of this chapter, we focus on integrating lexical categories with parsing using gold-standard ERG token boundaries because our lexical categorization models were trained on gold-standard ERG token boundaries.

## 5.4 Lexical Categories Integration

Dridan (2009) investigated using ERG lexical categories to impose hard constraints on the parser. She used lexical categories, from the eight degrees of granularities (cf. Table 4.1), to improve parsing efficiency, accuracy and coverage. Furthermore, she experimented with using single-, multiple- and selective-taggers (cf. § 2.2.3).

Dridan (2009) reports that POS[4] single tags led to a three-fold speed-up in parsing time but at the cost of 8% in coverage. She also finds that multiple le-type tags can achieve very high speed-up, but multiple POS tags give the best absolute coverage with relatively minor speed increases. In selective tagging, she shows that le-types can lead to a 12-fold speed-up but at the cost of losing up to 30% in coverage. However, selective POS+morph tags can increase efficiency by 50% while maintaining coverage and accuracy. Dridan (2009) concludes that modest efficiency increases are possible using lexical restriction.

In this section we show that significant efficiency gains with lexical restriction are actually attainable. Our results, however, are not directly comparable to Dridan (2009) because she: (a) used a different version of ERG, (b) trained and tested on different data sets, (c) used initial tokens (PTB-like) instead of lexical tokens (ERG-like).

We experiment with two configurations of using lexical categories to restrict the parser search space. First, using the major syntactic categories assigned by the model we developed in § 4.4. Second, using the lexical types assigned by the model we developed in § 4.3. In all configurations our comparison reference point is the unrestricted parser with gold-standard ERG token boundaries, because our lexical categorization models use ERG token boundaries. To define upper bounds for our experiments, we restrict the parser with gold-standard

[4]Note that Dridan (2009) used the term ‘POS’ for major syntactic categories (MSC).

Table 5.3: Parsing efficiency, coverage and accuracy using gold-standard major syntactic categories and lexical types.

| | **Efficiency** | **Coverage** | **Accuracy** | |
|---|---|---|---|---|
| | Seconds | % | Exact matches | PARSEVAL |
| Unrestricted | 19 | 97.6 | 345 | 87.8 |
| Gold MSC | 4.31 | 99.9 | 400 | 90.1 |
| Gold le-type | 0.87 | 100 | 585 | 93.4 |

Table 5.4: Parsing efficiency, coverage and accuracy with $n$-best major syntactic categories

| | **Efficiency** | **Coverage** | **Accuracy** | |
|---|---|---|---|---|
| | Seconds | % | Exact matches | PARSEVAL |
| Unrestricted | 19.00 | 97.6 | 345 | 87.8 |
| 1-best | 4.00 | 91.6 | 305 | 84.0 |
| 2-best | 4.67 | 95.5 | 333 | 86.3 |
| 5-best | 6.59 | 98.3 | 352 | 87.3 |

major syntactic categories and lexical types. Table 5.3 shows the efficiency, coverage and accuracy attained by using gold-standard MSCs and le-types.

Both MSCs and le-types promise noticeable improvements on the three dimensions of parsing evaluation, accuracy, coverage and efficiency. In the following two sections, we assess how close to the upper bounds we land using automatically assigned major syntactic categories and lexical types.

### 5.4.1 Major Syntactic Categories Integration

In § 4.4, we developed a `CRF` model to assign major syntactic categories that achieves an accuracy of 98.01% (on `DeepBank` section 21). Now, we shall determine the usefulness of such a model for the parsing task, i.e. investigate how an MSC accuracy of 98.01% per token would be reflected on parsing efficiency, coverage and accuracy.

Table 5.4 shows the results of restricting the parser search space using 1-best, 2-best and 5-best lists of major syntactic categories assigned by the MSC model.

With 1-best MSCs, we see a reduction of 79% in parsing time but at the cost of 6 points coverage and 3.8 points in PAREVAL (labeled brackets). However, with 5-best lists, the parsing time is reduced by 65% and coverage is increased by 0.7 point, at the cost of 0.5 point in PARSEVAL accuracy, but it is worth observing that the number of exact matches increases.

In § 4.4.1, we reported that the lexical type model (le-type) achieves an accuracy just as high as the MSC model when evaluated based only on the correctness of the MSC fields of the lexical types it assigns. As those two models

Table 5.5: Parsing efficiency, coverage and accuracy with major syntactic categories of $n$-best lexical types

| | **Efficiency** | **Coverage** | **Accuracy** | |
|---|---|---|---|---|
| | Seconds | % | Exact matches | PARSEVAL |
| Unrestricted | 19.00 | 97.6 | 345 | 87.8 |
| 1-best | 4.01 | 90.5 | 300 | 83.1 |
| 2-best | 4.19 | 92.6 | 317 | 84.5 |
| 5-best | 6.64 | 95.3 | 337 | 86.1 |

achieve the same level of accuracy, we compare their effects on parsing to see whether they also achieve the same level of improvement on parsing. Hence, we evaluate using the MSCs of the lexical types assigned by the le-type model (LE-MSC, henceforth) to restrict the parser search space. Table 5.5 shows the results of experimenting with 1-best, 2-best and 5-best lists of LE-MSCs.

From Table 5.5 we see that 1-best LE-MSC achieves the same level of efficiency as 1-best MSC, however at the cost of 8 points in coverage and 4.7 points in accuracy. Furthermore, 5-best LE-MSC lists achieves almost the same level of efficiency as 5-best MSC but at higher costs of coverage and accuracy. In fact, 5-best LE-MSC leads to the coverage and accuracy of 2-best MSC; the reason is that when we generate the $n$-best lists in the le-type model, most lexical types would differ in the subcategorization and description fields but not in the MSC field. Hence, the ambiguity in the 5-best LE-MSC lists is lower than that in the 5-best MSC lists. In concrete terms, in the 5-best lists of LE-MSC each word is assigned 1.01 MSCs, whereas in the 5-best lists of MSCs each token receives 1.18 MSCs.

Given the results in Table 5.4 and Table 5.5, we can conclude: in practice the MSC model is better than the le-type model in assigning major syntactic categories.

A common method to restrict parsing systems with imperfect taggers or supertaggers is so-called *selective tagging*; in this method, only lexical categories with a confidence (or probability) level higher than a predefined threshold ($\beta$) would be used in restricting the parser search space. We experiment with using selective tagging for major syntactic categories with various thresholds ($\beta$).

Table 5.6 shows that selective MSCs with $\beta = 0.95$ can improve parsing efficiency and coverage with no loss in accuracy. In addition, selective MSCs with $\beta = 0.95$ achieves the same level of efficiency as 5-best lists MSCs and with slightly higher scores in coverage and accuracy. Selective tagging, moreover, has a major advantage over multiple tagging ($n$-best lists) because the latter is computationally expensive to generate, whereas selective tagging is just a matter of pruning the 1-best list of MSCs.

In Figure 5.1 we draw the relations coverage vs. efficiency and accuracy vs. efficiency, using selective MSCs with different thresholds from $\beta = 0.50$ up to

Table 5.6: Parsing efficiency, coverage and accuracy with selective major syntactic categories

| | **Efficiency** | **Coverage** | **Accuracy** | |
|---|---|---|---|---|
| | Seconds | % | Exact matches | PARSEVAL |
| Unrestricted | 19 | 97.6 | 345 | 87.8 |
| $\beta$=0.80 | 4.87 | 96.4 | 340 | 86.9 |
| $\beta$=0.85 | 5.11 | 97.0 | 340 | 87.0 |
| $\beta$=0.90 | 5.39 | 97.6 | 349 | 87.4 |
| $\beta$=0.95 | 6.34 | 98.6 | 351 | 87.8 |

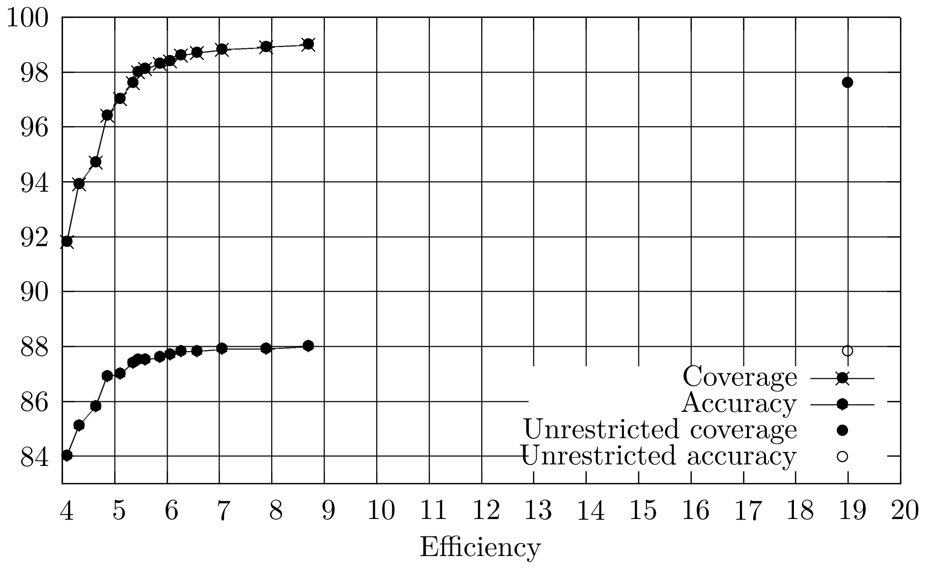


Figure 5.1: Efficiency vs. coverage and efficiency vs. accuracy with selective MSCs. Note that each bullet (•) on the curves marks a $\beta$ value as follows: 0.5,0.6,0.7,0.8,0.85,0.90,0.91. . . 0.98,0.99.

$\beta = 0.99$.

From Figure 5.1 we see that significant improvements in efficiency are achievable with no losses in accuracy and coverage. However, if less than 6 seconds is preferred for parsing a sentence, then one would have to give up some coverage and accuracy in return.

Finally, Table 5.7 presents detailed parsing efficiency results with selective MSCs ($\beta = 0.95$).

From Table 5.7 we see that the reduction in parsing time is proportional to the sentence length, which is exactly what we wish parser restriction would do. This result is particularly interesting as parsing time-outs usually happen when the sentence is relatively long (cf. Figure 5.3).

Table 5.7: Detailed parsing efficiency with selective major syntactic categories $\beta$=0.95

| **Aggregate** | **Unrestricted** Seconds | **$\beta$=0.95** Seconds | **Reduction** % |
|---|---|---|---|
| $40 \leq \textit{length} < 60$ | 146.15 | 41.52 | 71.6 |
| $20 \leq \textit{length} < 40$ | 36.11 | 11.81 | 67.3 |
| $0 \leq \textit{length} < 20$ | 1.83 | 1.01 | 44.5 |
| **Total** | **19.00** | **6.34** | **66.6** |

Table 5.8: Parsing efficiency, coverage and accuracy with selective lexical types

| | **Efficiency** | **Coverage** | **Accuracy** | |
|---|---|---|---|---|
| | Seconds | % | Exact matches | PARSEVAL |
| Unrestricted | 19.00 | 97.6 | 345 | 87.8 |
| $\beta$=0.80 | 1.35 | 89.3 | 366 | 84.2 |
| $\beta$=0.85 | 1.54 | 92.2 | 374 | 85.7 |
| $\beta$=0.90 | 1.97 | 94.7 | 383 | 86.8 |
| $\beta$=0.95 | 3.01 | 97.8 | 395 | 88.3 |

### 5.4.2 Lexical Types Integration

In this section we study the effect of lexical types assigned by our lexical type model (le-type model; § 4.3) on parsing efficiency, coverage and accuracy.

From the previous section, we learnt that selective tagging is the best choice for integrating imperfect lexical categorization models with parsers. Furthermore, the accuracy of the le-type model (92.84% on `DeepBank` section 21) is far less that of the MSC model and the computational cost for generating $n$-best lists of lexical types is quite expensive. Therefore, in this section we only experiment with selective lexical types.

Table 5.8 shows the results of restricting the parser using selective lexical types with several thresholds.

Selective lexical types with $\beta = 0.95$ achieves the highest reduction in parsing time while maintaining coverage and accuracy, if not increasing them as well. It reduces parsing time by 84% and increases coverage by 0.2 point and accuracy by 0.5 point. The number of exact matches also increases from 345 to 395.

In Figure 5.2 we draw the changes of accuracy and coverage with efficiency using selective lexical types with $\beta$ from 0.5 to 0.99.

From Figure 5.2 we see that rather high increases in efficiency are achievable without any losses in accuracy or coverage. We can also see that giving up efficiency until we reach 4.5 seconds per sentence helps improve accuracy and coverage, but thereafter the accuracy and coverage curves flatten out. The reason behind this behavior is that the last bullets on the curves correspond to

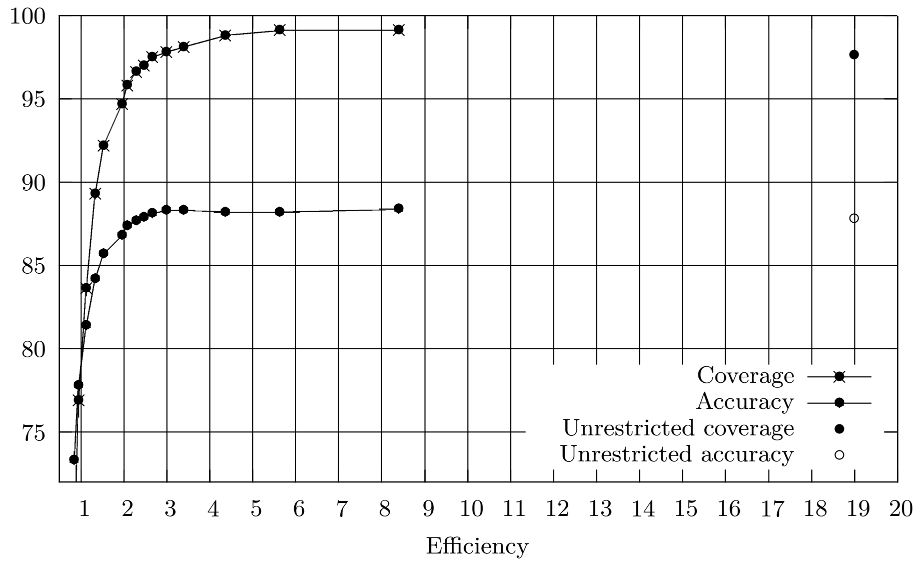


Figure 5.2: Efficiency vs. coverage and efficiency vs. accuracy with selective lexical types. Note that each bullet (•) on the curves marks a $\beta$ value as follows: 0.5,0.6,0.7,0.8,0.85,0.90,0.91. . . 0.98, 0.99.

high confidence thresholds (0.97, 0.98 and 0.99), hence the number of lexical types selected to restrict the parser significantly decreases.

Finally, we believe that the potentials of parsing improvement are even higher with the standard parsing mode which only allows 60 seconds for the parser to time-out. As mentioned in § 5.2, we run all the integration experiments using the development mode, hence the parser is given four minutes to analyze a sentence.

Figure 5.3 and Figure 5.4 illustrate the parsing times of all sentences in our test set using the unrestricted parser and restricted parser with selective lexical types ($\beta = 0.95$).

From Figure 5.3 and Figure 5.4 we can easily see the significance of restricting the parser search space in the standard mode.

## 5.5 Summary

In this chapter we studied integrating our tokenization and lexical categorization models with ERG parsing, using their outputs as hard constraints to restrict the parser search space. We showed that using ERG token boundaries leads to minor improvements in parsing efficiency without any losses in coverage and accuracy.

In lexical categories integration, we found that selective tagging with both major syntactic categories and lexical types delivers better results than single

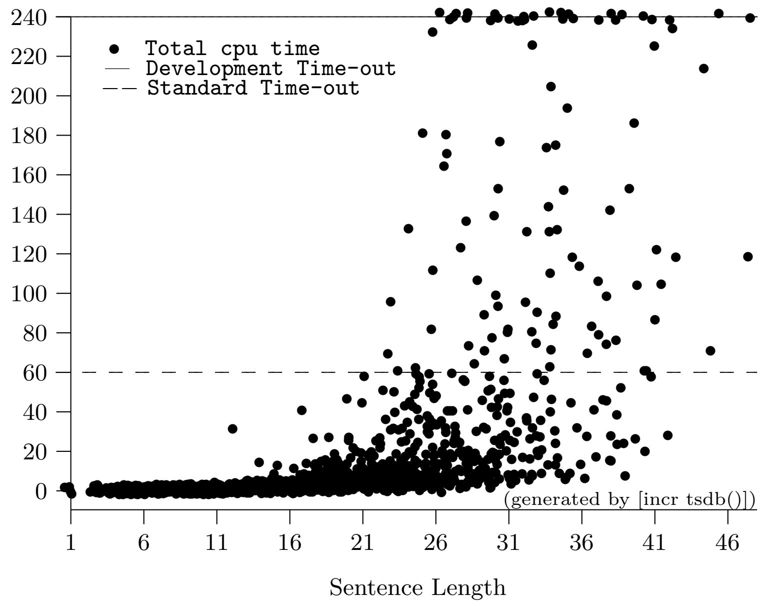


Figure 5.3: Parsing times using the unrestricted parser

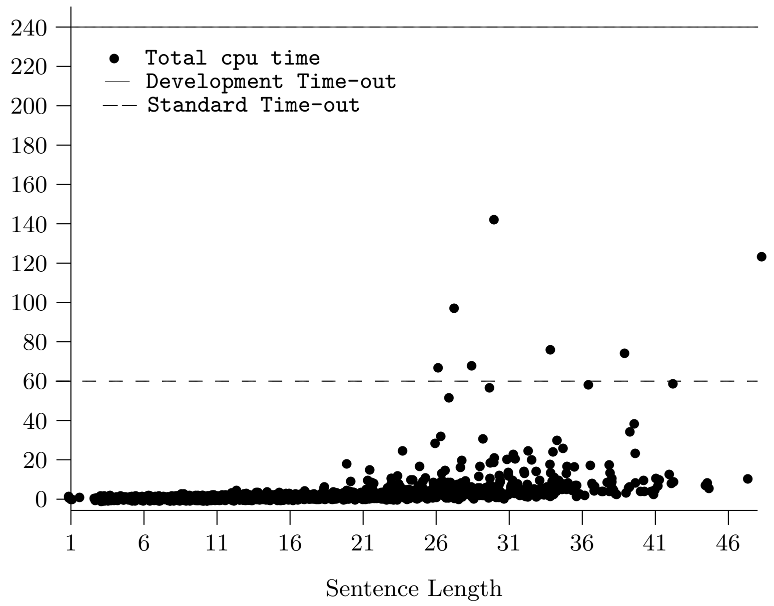


Figure 5.4: Parsing times with selective lexical types $\beta = 0.95$

and multiple tagging, which is in line with the findings of Dridan (2009).

Our experiments prove that major syntactic categories and lexical types can significantly improve parsing efficiency. Selective major syntactic categories ($\beta = 0.95$) reduces parsing time by 66% and increases coverage by 1 point while maintaining accuracy. Selective lexical types ($\beta = 0.95$), on the other hand, reduces parsing time by 84% with small improvements in coverage and accuracy, showing that sometimes there is a 'free lunch' in machine learning.[5]

All in all, our parser integration experiments showed pronounced improvements over what Dridan (2009) reported, which can be attributed to: (a) difference in the experimental setup, e.g. we didn't make any code changes to the PET parser while Dridan (2009) did, (b) differences in the tokenization schemes, i.e. the use of lexical tokens vs. initial tokens, (c) the mapping from generic to native lexical types (cf. § 4.3), (d) more sophisticated machine learning algorithms (`CRFs` vs. `MaxEnt` and `HMM`), e.g. Dridan (2009) found that the `C&C` supertagger gives better results than the `TnT` tagger when she used the underlying probability distribution of each model to guide tag assignment, hence it might be that `CRFs` perform quite well in assigning tag probabilities.

[5]No Free Lunch Theorems (Wolpert & Macready, 1997).

# Chapter 6

# Conclusion

The HPSG ERG—being a broad-coverage grammar in the context of detailed syntacto-semantic analyses—defines tokenization and lexical categorization regimes that diverge from the classic conceptions of such tasks. Both ERG tokenization and lexical categorization, to varying degrees, blend aspects of syntactic analysis. ERG tokenization recognizes multi-word expressions, hence the whitespace is no longer a sufficient or necessary word separator. ERG lexical categories more overtly encode syntactic properties in addition to morphological and lexical information.

In this thesis, we investigated the use of Conditional Random Fields (CRFs), a somewhat recent 'trend' in sequence labeling, to model the rather challenging ERG conception of tokenization and lexical categorization.

In Chapter 2, we presented a theoretical background on sequence labeling in general and CRFs in particular. We briefly introduced some aspects of the ERG parsing pipeline and reviewed previous related work.

The subtleties of the tokenization task, variability in common tokenization schemes, and their potential impact on the downstream processing inspired our investigation of a data-driven approach towards tokenization in Chapter 3. We showed that tokenization comprises a range of challenges and opportunities that should be equally amenable to careful engineering and experimentation as other NLP tasks—especially if one to consider the far more challenging view on tokenization of ERG.

We reported state-of-the-art results in PTB tokenization and showed that domain-adaptable tokenization models in a sequence labeling approach can achieve very high accuracies, and generally outperform state-of-the-art rule-based systems. Further, we presented a tokenization model for ERG, that even though didn't achieve the same level of accuracy as our PTB tokenization model, it was accurate enough to serve as a front-end for the parser, and even to improve its efficiency.

Our tokenization toolkit with pre-trained PTB and ERG models will be released within the DELPH-IN open source repository, either for stand-alone use (PTB tokenization) or for integration into the ERG parsing pipeline or the parsers of Zhang and Krieger (2011); Ytrestøl (2011); Evensberget (2012).

The intricacy of ERG lexical categorization on the one hand, and the superiority of CRFs in different NLP tasks on the other hand, motivated our research on using CRFs to model the task of ERG lexical categorization in Chapter 4.

We studied ERG lexical categorization on two levels of linguistic granularity, major syntactic categories and lexical types. On the lexical type level, we presented a detailed study on candidate features to learn ERG lexical types, and reported the accuracy, main memory requirement and training time for eleven models trained with different combinations of these features. We also showed that the use of a few CRF bigram features to learn the ERG lexical types, is very expensive and leads to minor improvements in accuracy. Although our simulation of constrained decoding for the CRF lexical type model didn't substantially improve accuracy, we still believe that constrained CRFs might help reduce the training and decoding times of CRFs (cf. below). In comparing our CRF model to the C&C supertagger, we found that the CRF-mimicry of the C&C supertagger is significantly more accurate than C&C, however at the cost of longer training times.

In the major syntactic category experiments, our CRF model delivered high accuracy with 1-best decoding and near-perfect accuracy with 5-best lists of major syntactic categories. We showed that applying the ERG derivational morphological rules to change major syntactic categories doesn't lead to improvement in accuracy, which suggested that most of the MSC errors are independent of derivational morphological properties.

We presented the specified lexical types approach by dividing the set of lexical types into eleven mutually exclusive but interacting sets. In this approach, we substantially reduced the training time and memory requirements to the extent that training on more than 22 million words became feasible. We also confirmed the findings of Ytrestøl (2012) that training only on very large amounts of automatically annotated data can achieve relatively competitive accuracies, especially on out-of-domain data, in comparison to training on manually annotated data.

The ultimate aim of improving ERG parsing was discussed in Chapter 5. We experimented with using our tokenization and lexical categorization models to prune the ERG parser search space. We saw the classic tradeoff, accuracy vs. efficiency, but also selective tagging models offered us a 'free lunch' by improving parsing efficiency, accuracy and coverage all at once.

We showed that using ERG token boundaries leads to about 8% reduction in parsing time without any losses in coverage and accuracy. The integration of lexical categories, however, led to more pronounced efficiency improvements, with selective lexical types reducing parsing time by 84% and slightly increasing coverage and accuracy. We also showed that richer linguistic granularity can lead to greater gains in parser improvement, evidenced by the results of using of major syntactic categories vs. lexical types to prune the parser search space.

## 6.1 Future Work

While a certain degree of feature engineering has been invested during our tokenization experiments, we believe that further (empirical) improvements can be made in the design of the sequence labeler, e.g. to copy the period following certain abbreviations (if strict compliance to the WSJ section of the PTB were the goal). Also, extra labels could be introduced in the sequence labeler to achieve certain rewriting operations (in a sense similar to that of a finite-state transducer).

The ERG lexicon provides us with valuable information that can be exploited to constraint the ERG tokenization model, i.e. extract a set of multi-word tokens from the ERG lexicon. Given this information, one can constrain CRFs in a way similar to our simulation of constrained decoding for lexical categories or in an actual implementation of constrained CRFs.

In addition, examining the generalizations of our tokenization models and mismatches against gold-standard annotations, already has proven a useful technique in the identification of inconsistencies and errors within existing resources. Hence, one can investigate the benefit of our models as a feedback mechanism to resource creation, i.e. error detection in annotated corpora.

In the specified lexical type setup, due to time constraints, we trained only two specified lexical type models on the NANC data (§ 4.5.2). Training all of the specified lexical type models on NANC data needs to be done in future work and possibly augmented with a constrained decoding strategy. Moreover, training our specified lexical type models on 22.5 million words took 3.6 days, one can also try training up to one month, as Ytrestøl (2012) did, to define an upper limit for CRFs scalability and possible accuracy gains from training on massive amounts of automatically generated training data.

The pattern of the ERG lexical types allows the use of ensemble learning approaches; while we touched upon this idea in § 4.3.5, we believe it still needs comprehensive and careful experimentation.

In combining the outputs of the specified lexical type models, one can also generate the $n$-best lists of all models, then exploit the probabilities (confidence scores) associated with these outputs in order to aggregate the final output.

# Appendix A

# Generic to Native Mapping

## Generic to Native Lexical Types Mapping Rules

The DeepBank was created by automatically parsing the WSJ text, and then manually correcting or disambiguating the output of the parser. One side effect of this methodology in the ERG realm is that unknown words would be assigned so-called 'generic lexical entries' (in contrast to known words which are assigned 'native lexical entries'). The generic lexical entries are essentially underspecified lexical entries instantiated to fill the gaps in the lexical chart, i.e. fill the input positions for which no native entries were found.

In the following, Table A.1, we present the mapping rules we used to convert all generic lexical types in DeepBank to their corresponding native ones.

Table A.1: Generic to native lexical type mapping rules

| **Generic Lexical Type** | **Native Lexical Type** |
|---|---|
| aj_-_i-cmp-unk_le | aj_pp_i-cmp_le |
| aj_-_i-crd-gen_le | aj_-_i-crd-two_le |
| aj_-_i-crd-unk_le | aj_-_i-crd-two_le |
| aj_-_i-frct-gen_le | aj_-_i-frct_le |
| aj_-_i-ord-gen_le | aj_-_i-ord-two_le |
| aj_-_i-sup-unk_le | aj_-_i-sup_le |
| aj_-_i-unk_le | aj_-_i_le |
| aj_np_i-crd-gen_le | aj_np_i-crd-nsp_le |
| av_-_dc-like-unk_le | av_-_dc-like-pr_le |
| av_-_i-unk_le | av_-_i-vp_le |
| n_-_c-pl-gen_le | n_pl_olr |
| n_-_c-pl-unk_le | n_pl_olr |
| n_-_day-crd-gen_le | n_-_c-day_le |
| n_-_mc-ns-g_le | n_-_mc-ns_le |
| n_-_mc-unk_le | n_-_mc_le |
| n_-_meas-n-gen_le | n_-_c-meas_le |
| n_-_pn-dom-e-gen_le | n_-_pn-dom-euro_le |
| n_-_pn-dom-gen_le | n_-_pn-dom-card_le |
| n_-_pn-dom-o-gen_le | n_-_pn-dom-ord_le |
| n_-_pn-gen_le | n_-_pn_le |
| n_-_pn-pl-unk_le | n_-_pn-pl_le |
| n_-_pn-unk_le | n_-_pn_le |
| n_np_pn-hour-gen_le | n_-_pn-hour_le |
| v_-_pas-unk_le | v_-_psv_le |
| v_np*_bse-unk_le | v_n3s-bse_ilr |
| v_np*_pa-unk_le | v_pst_olr |
| v_np*_pr-3s-unk_le | v_3s-fin_olr |
| v_np*_pr-n3s-unk_le | v_n3s-bse_ilr |
| v_np*_prp-unk_le | v_prp_olr |
| v_np*_psp-unk_le | v_psp_olr |
| v_np*_unk_le | v_np*_le |

# Bibliography

Adolphs, P., Oepen, S., Callmeier, U., Crysmann, B., Flickinger, D., & Kiefer, B. (2008). Some fine points of hybrid natural language parsing. In *Proceedings of the 6th International Conference on Language Resources and Evaluation.* Marrakech, Morocco.

Alpaydin, E. (2004). *Introduction to machine learning.* MIT press.

Auli, M., & Lopez, A. (2011, June). A Comparison of Loopy Belief Propagation and Dual Decomposition for Integrated CCG Supertagging and Parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies* (pp. 470–480). Portland, Oregon, USA: Association for Computational Linguistics.

Black, E., Abney, S., Flickenger, D., Gdaniec, C., Grishman, C., Harrison, P., . . . Strzalkowski, T. (1991). Procedure for quantitatively comparing the syntactic coverage of english grammars. In *Proceedings of the workshop on speech and natural language* (pp. 306–311). Stroudsburg, PA, USA: Association for Computational Linguistics.

Bloomfield, L. (1935). *Language.* London, UK: George Allen and Unwin.

Bouma, G., Van Noord, G., & Malouf, R. (2001). Alpino: Wide-coverage computational analysis of Dutch. *Language and Computers*, *37*(1), 45–59.

Brants, T. (1997). Internal and External Tagsets in Part-of-Speech Tagging. In *Proceedings of Eurospeech.* Rhodes, Greece.

Brants, T. (2000). TnT – A Statistical Part-of-Speech Tagger. In *Proceedings of the Sixth Conference on Applied Natural Language Processing* (pp. 224–231). Seattle, Washington, USA.

Briscoe, T., & Carroll, J. (1995). Developing and Evaluating a Probabilistic LR Parser of Part-of-Speech and Punctuation Labels. In *Proceedings of the 4th ACL/SIGPARSE International workshop on Parsing Technologies* (pp. 48–58). Prague, Czech Republic.

Brown, R. W. (1957). Linguistic Determinism and the Part of Speech. *The Journal of Abnormal and Social Psychology*, *55*(1).

Callmeier, U. (2000). PET — a platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering*, *6*(1), 99–107.

Chen, J., & Rambow, O. (2003). Use of Deep Linguistic Features for the Recognition and Labeling of Semantic Arguments. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing* (pp. 41–48).

Chen, X., & Kit, C. (2011). Improving Part-of-speech Tagging for Context-free Parsing. In *Proceedings of 5th International Joint Conference on Natural Language Processing* (pp. 1260–1268). Chiang Mai, Thailand.

Chiarcos, C., Ritz, J., & Stede, M. (2009, August). By all these lovely tokens... merging conflicting tokenizations. In *Proceedings of the Third Linguistic Annotation Workshop* (p. 35 – 43). Suntec, Singapore: Association for Computational Linguistics.

Clark, S. (2002). Supertagging for Combinatory Categorial Grammar. In *Proceedings of the 6th International workshop on Tree Adjoining Grammars and Related Frameworks* (pp. 19–24). Venice, Italy.

Clark, S., & Curran, J. R. (2004, August). The Importance of Supertagging for Wide-Coverage CCG Parsing. In *Proceedings of the 20th International Conference on Computational Linguistics* (pp. 282–288). Geneva, Switzerland: COLING.

Clark, S., & Curran, J. R. (2007). Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, *33*(4), 493–552.

Clark, S., & Curran, J. R. (2010). Supertagging for Efficient Wide-Coverage CCG Parsing. *Supertagging: Using Complex Lexical Descriptions in Natural Language Processing*, 221–233.

Collins, M. (2002). Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing* (pp. 1–8). Philadelphia, PA, USA.

Curran, J. R., Clark, S., & Vadas, D. (2006, July). Multi-tagging for lexicalized-grammar parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Meeting of the Association for Computational Linguistics* (p. 697 – 704). Sydney, Australia: Association for Computational Linguistics.

Dalrymple, M. (2006). How much can part-of-speech tagging help parsing? *Natural Language Engineering*, *12*, 373–389.

Davidson, T. (1874). The grammar of Dionysios Thrax. *The Journal of Speculative Philosophy*, *8*(4), 326–339.

Denis, P., & Sagot, B. (2009). Coupling an annotated corpus and a morphosyntactic lexicon for state-of-the-art POS tagging with less human effort. In *Pacific Asia Conference on Language, Information and Computation.* Hong Kong, Chine.

Dridan, R. (2009). *Using lexical statistics to improve HPSG parsing.* PhD thesis, Saarland University.

Dridan, R., & Oepen, S. (2012, July). Tokenization. Returning to a long solved problem. A survey, contrastive experiment, recommendations, and toolkit. In *Proceedings of the 50th Meeting of the Association for Computational Linguistics* (p. 378 – 382). Jeju, Republic of Korea.

Evensberget, J. B. (2012). *Context-free approximation of large unification grammars: A walk in the forest.* Master's thesis, University of Oslo.

Fares, M., Oepen, S., & Zhang, Y. (2013). Machine Learning for High-Quality Tokenization Replicating Variable Tokenization Schemes. In *Computational Linguistics and Intelligent Text Processing* (Vol. 7816, pp. 231–244). Samos, Greece: Springer Berlin Heidelberg.

Flickinger, D. (2000). On building a more efficient grammar by exploiting types. *Natural Language Engineering*, *6 (1)*, 15 – 28.

Flickinger, D., Oepen, S., & Ytrestøl, G. (2010, May). Wikiwoods: Syntacto-semantic annotation for English Wikipedia. In *Proceedings of the 7th conference on international language resources and evaluation.* Valletta, Malta.

Flickinger, D., Zhang, Y., & Kordoni, V. (2012). DeepBank: A dynamically annotated treebank of the Wall Street Journal. In *Proceedings of the 11th International Workshop on Treebanks and Linguistic Theories (TLT-11).* Lisbon, Portugal.

Forst, M., & Kaplan, R. M. (2006). The importance of precise tokenizing for deep grammars. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006)* (pp. 369–372).

Foster, J. (2010, June). "cba to check the spelling": Investigating parser performance on discussion forum posts. In *Human Language Technology Conference: The 2010 annual conference of the north american chapter of the association for computational linguistics* (p. 381 – 384). Los Angeles, California: Association for Computational Linguistics.

Francis, W. N. (1964). A standard sample of present-day English for use with digital computers.

Graff, D. (1995). *North American News Text Corpus.* Linguistic Data Consortium. LDC95T21.

Green, S., de Marneffe, M.-C., Bauer, J., & Manning, C. D. (2011, July). Multiword expression identification with tree substitution grammars: A parsing tour de force with french. In *Proceedings of the 2011 conference on empirical methods in natural language processing* (p. 725 – 735). Edinburgh, Scotland, UK.: Association for Computational Linguistics.

Greene, B. B., & Rubin, G. M. (1971). *Automatic grammatical tagging of English.* Department of Linguistics, Brown University.

Hassan, H., Sima'an, K., & Way, A. (2007). Supertagged Phrase-Based Statistical Machine Translation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics* (pp. 288–295). Prague, Czech Republic: Association for Computational Linguistics.

Hockenmaier, J. (2003). Data and models for statistical parsing with Combinatory Categorial Grammar.

Hopper, P. J., & Thompson, S. A. (1984). The discourse basis for lexical categories in universal grammar. *Language*, 703–752.

Hovy, E., Marcus, M., Palmer, M., Ramshaw, L., & Weischedel, R. (2006, June). Ontonotes. The 90% solution. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics* (p. 57 – 60). New York City, USA.

Joachims, T., Finley, T., & Yu, C.-N. J. (2009). Cutting-plane training of structural SVMs. *Machine Learning*, *77*(1), 27–59.

Joshi, A. K., & Srinivas, B. (1994). Disambiguation of super parts of speech (or supertags): almost parsing. In *Proceedings of the 15th Conference on Computational Linguistics - volume 1* (pp. 154–160). Kyoto, Japan.

Jurafsky, D., & Martin, J. H. (2008). *Speech and language processing* (2nd ed.). Upper Saddle River: Prentice Hall.

Kaplan, R. M. (2005). A method for tokenizing text. Festschrift for Kimmo Koskenniemi on his 60th birthday. In A. Arppe et al. (Eds.), *Inquiries into words, constraints and contexts* (p. 55 – 64). Stanford, CA: CSLI Publications.

Kim, J.-D., Ohta, T., Teteisi, Y., & Tsujii, J. (2003). GENIA corpus — a semantically annotated corpus for bio-textmining. *Bioinformatics*, *19*, i180 – i182.

King, T. H., Crouch, R., Riezler, S., Dalrymple, M., & Kaplan, R. M. (2003). The PARC 700 Dependency Bank. In *In Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora (LINC-03)* (pp. 1–8).

Kulick, S., Bies, A., Liberman, M., Mandel, M., McDonald, R., Palmer, M., . . . White, P. (2004). Integrated annotation for biomedical information extraction. In *Proceedings of the Human Language Technology Conference and the Annual Meeting of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL)* (pp. 61–68).

Kummerfeld, J. K., Roesner, J., Dawborn, T., Haggerty, J., Curran, J. R., & Clark, S. (2010, July). Faster Parsing by Supertagger Adaptation. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics* (pp. 345–355). Uppsala, Sweden: Association for Computational Linguistics.

Lafferty, J. D., McCallum, A., & Pereira, F. C. N. (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the Eighteenth International Conference on Machine Learning* (pp. 282–289). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Lapponi, E., Velldal, E., Øvrelid, L., & Read, J. (2012, June). UiO2: sequence-labeling negation using dependency features. In *Proceedings of the 1st Joint Conference on Lexical and Computational Semantics* (p. 319 – 327). Montréal, Canada.

Lavergne, T., Cappé, O., & Yvon, F. (2010, July). Practical very large scale CRFs. In *Proceedings of the 48th Meeting of the Association for Computational Linguistics* (p. 504 – 513). Uppsala, Sweden.

Lease, M., & Charniak, E. (2005). Parsing biomedical literature. In *Proceedings of the Second International Joint Conference on Natural Language Processing* (pp. 58–69). Jeju Island, Korea.

Manning, C. D., & Schütze, H. (1999). *Foundations of statistical natural language processing.* MIT press.

Marcus, M., Santorini, B., & Marcinkiewicz, M. A. (1993). Building a large annotated corpora of English: The Penn Treebank. *Computational Linguistics*, *19*, 313 – 330.

Marshall, I. (1983). Choice of grammatical word-class without global syntactic analysis: Tagging words in the lob corpus. *Computers and the Humanities*, *17*, 139-150.

Maršík, J., & Bojar, O. (2012, September). TrTok: A Fast and Trainable Tokenizer for Natural Languages. *Prague Bulletin of Mathematical Linguistics*, *98*, 75 – 85.

McArthur, T. (1992). *The Oxford companion to the English language.* Oxford University Press.

McClosky, D., Charniak, E., & Johnson, M. (2006a, June). Effective Self-Training for Parsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference* (pp. 152–159). New York City, USA: Association for Computational Linguistics.

McClosky, D., Charniak, E., & Johnson, M. (2006b). Reranking and Self-Training for Parser Adaptation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics* (pp. 337–344). Sydney, Australia.

Ninomiya, T., Matsuzaki, T., Tsuruoka, Y., Miyao, Y., & Tsujii, J. (2006). Extremely Lexicalized Models for Accurate and Fast HPSG Parsing. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing* (pp. 155–163). Sydney, Australia.

Oepen, S., & Callmeier, U. (2000). Measure for Measure: Parser Cross-Fertilization. Towards Increased Component Comparability and Exchange. In *Proceedings of the 6th international workshop on parsing technologies (iwpt'2000)* (pp. 183–194).

Oepen, S., Flickinger, D., Toutanova, K., & Manning, C. D. (2004). LinGO Redwoods. A rich and dynamic treebank for HPSG. *Research on Language and Computation*, *2*(4), 575 – 596.

Oepen, S., & Flickinger, D. P. (1998). Towards Systematic Grammar Profiling. Test Suite Technology Ten Years After. *Journal of Computer Speech and Language*, *12*(4), 411–436.

Øvrelid, L., Velldal, E., & Oepen, S. (2010). Syntactic scope resolution in uncertainty analysis. In *Proceedings of the 23rd International Conference* *on Computational Linguistics* (p. 1379 – 1387). Stroudsburg, PA, USA: Association for Computational Linguistics.

Petrov, S., Das, D., & McDonald, R. (2012, May). A Universal Part-of-Speech Tagset. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)* (pp. 2089–2096). Istanbul, Turkey: European Language Resources Association (ELRA).

Petrov, S., & Klein, D. (2007, April). Improved Inference for Unlexicalized Parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference* (pp. 404–411). Rochester, New York: Association for Computational Linguistics.

Petrov, S., & McDonald, R. (2012). Overview of the 2012 shared task on parsing the web. In *Notes of the first workshop on syntactic analysis of non-canonical language (sancl).*

Plank, B., & van Noord, G. (2008). Exploring an Auxiliary Distribution Based Approach to Domain Adaptation of a Syntactic Disambiguation Model. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation* (pp. 9–16). Manchester, UK.

Pollard, C., & Sag, I. A. (1994). *Head-Driven Phrase Structure Grammar.* Chicago, IL, USA: The University of Chicago Press.

Prins, R., & van Noord, G. (2003). Reinforcing Parser Preferences through Tagging. *Traitement Automatique des Langues*, *44*(3), 121–139.

Przepiórkowski, A. (2005). The IPI PAN Corpus in numbers. In *Proceedings of the 2nd Language & Technology Conference, Poznan, Poland* (pp. 27–31).

Przepiórkowski, A., Górski, R. L., Łazinski, M., & Pezik, P. (2010). Recent Developments in the National Corpus of Polish. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation LREC'10.* Valletta, Malta: European Language Resources Association (ELRA).

Pyysalo, S., Ginter, F., Heimonen, J., Björne, J., Boberg, J., Järvinen, J., & Salakoski, T. (2007). Bioinfer: a corpus for information extraction in the biomedical domain. *BMC Bioinformatics*, *8*(1), 50.

Radziszewski, A. (2013). A Tiered CRF Tagger for Polish. In *Intelligent tools for building a scientific information platform* (Vol. 467, pp. 215–230). Springer Berlin Heidelberg.

Read, J., Dridan, R., & Oepen, S. (2013, May). Simple and Accountable Segmentation of Marked-up Text. In *Proceedings of the 19th nordic conference on computational linguistics.* Oslo, Norway.

Rimell, L., & Clark, S. (2009). Porting a lexicalized-grammar parser to the biomedical domain. *Journal of Biomedical Informatics*, *4*, 852–865.

Sag, I., Baldwin, T., Bond, F., Copestake, A., & Flickinger, D. (2002). Multiword expressions: A pain in the neck for nlp. *Computational Linguistics and Intelligent Text Processing*, 189–206.

Sarkar, A. (2007). Combining Supertagging and Lexicalized Tree-Adjoining Grammar Parsing. *Complexity of Lexical Descriptions and its Relevance to Natural Language Processing: A Supertagging Approach.*

Schabes, Y., & Joshi, A. K. (1990). Parsing with lexicalized tree adjoining grammar. *Technical Reports (CIS).*

Schmid, H. (1994). Probabilistic part-of-speech tagging using decision trees. In *Proceedings of International Conference on New Methods in Language Processing* (Vol. 12, pp. 44–49).

Seshadri, N., & Sundberg, C.-E. (1994). List Viterbi decoding algorithms with applications. *IEEE Transactions on Communications*, *42*(234), 313–323.

Sha, F., & Pereira, F. (2003). Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1* (pp. 134–141). Edmonton, Canada: Association for Computational Linguistics.

Shen, L., & Joshi, A. K. (2003). A SNoW Based Supertagger with Application to NP Chunking. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics* (pp. 505–512). Sapporo, Japan: Association for Computational Linguistics.

Skjærholt, A. (2011). *Ars flectandi: Automated morphological analysis of Latin.* Master's thesis, University of Oslo.

Søgaard, A. (2010). Simple semi-supervised training of part-of-speech taggers. In *Proceedings of the ACL 2010 Conference Short Papers* (pp. 205–208). Uppsala, Sweden.

Spoustová, D. j., Hajič, J., Raab, J., & Spousta, M. (2009). Semi-Supervised Training for the Averaged Perceptron POS Tagger. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)* (pp. 763–771). Athens, Greece.

Surdeanu, M., Johansson, R., Meyers, A., Màrquez, L., & Nivre, J. (2008). The CoNLL 2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of the 12th Conference on Natural Language Learning* (p. 159 – 177). Manchester, England.

Sutton, C., & McCallum, A. (2006). *An introduction to conditional random fields for relational learning.* Introduction to statistical relational learning. MIT Press.

Tomanek, K., Wermter, J., & Hahn, U. (2007). Sentence and token splitting based on conditional random fields. In *Proceedings of the 10th Conference of the Pacific Association for Computational Linguistics* (p. 49 – 57). Melbourne, Australia.

Toutanova, K., Klein, D., Manning, C. D., & Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1* (pp. 173–180). Edmonton, Canada.

Toutanova, K., Manning, C., Shieber, S., Flickinger, D., & Oepen, S. (2002). Parse Disambiguation for a Rich HPSG Grammar. In *First Workshop on Treebanks and Linguistic Theories (TLT2002)* (pp. 253–263). Sozopol, Bulgaria.

Tsuruoka, Y., Tateishi, Y., Kim, J.-D., Ohta, T., McNaught, J., Ananiadou, S., & Tsujii, J. (2005). Developing a robust part-of-speech tagger for biomedical text. In *Proceedings of the 10th Panhellenic conference on Advances in Informatics* (p. 382 – 392). Berlin, Heidelberg: Springer-Verlag.

van Noord, G. (2007, June). Using Self-Trained Bilexical Preferences to Improve Disambiguation Accuracy. In *Proceedings of the Tenth International Conference on Parsing Technologies* (pp. 1–10). Prague, Czech Republic: Association for Computational Linguistics.

Velldal, E. (2008). *Empirical Realization Ranking.* PhD thesis, University of Oslo, Department of Informatics.

Waszczuk, J. (2012). Harnessing the CRF Complexity with Domain-Specific Constraints. The Case of Morphosyntactic Tagging of a Highly Inflected Language. In *Proceedings of COLING 2012* (pp. 2789–2804). Mumbai, India.

Watson, R. (2006). Part-of-speech Tagging Models for Parsing. In *Proceedings of the 9th Conference of Computational Linguistics in the UK (CLUK'06).* Milton Keynes, UK.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics*, *1*(6), 80–83.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, *1*(1), 67–82.

Yoshida, K., Tsuruoka, Y., Miyao, Y., & Tsujii, J. (2007). Ambiguous part-of-speech tagging for improving accuracy and domain portability of syntactic parsers. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence* (p. 1783 – 1788). Hyderabad, India: Morgan Kaufmann Publishers Inc.

Ytrestøl, G. (2011). Cuteforce. Deep deterministic HPSG parsing. In *Proceedings of the 12th International Conference on Parsing Technologies* (p. 186 – 197). Dublin, Ireland.

Ytrestøl, G. (2012). *Transition-Based Parsing for Large-Scale Head-Driven Phrase Structure Grammars.* PhD thesis, University of Oslo, Department of Informatics.

Zhang, Y., & Krieger, H.-U. (2011). Large-scale corpus-driven PCFG approximation of an HPSG. In *Proceedings of the 12th International Conference on Parsing Technologies* (p. 198 – 208). Dublin, Ireland.