Probabilistic Tree Transducers for Grammatical Error Correction

by

Jan Moolman Buys

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Computer Science in the Faculty of Science at Stellenbosch University

Computer Science Division,
Department of Mathematical Sciences,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Supervisor: Prof. A.B. van der Merwe

December 2013

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: . . . .

Abstract

We investigate the application of weighted tree transducers to correcting grammatical errors in natural language. Weighted finite-state transducers (FST) have been used successfully in a wide range of natural language processing (NLP) tasks, even though the expressiveness of the linguistic transformations they perform is limited. Recently, there has been an increase in the use of weighted tree transducers and related formalisms that can express syntax-based natural language transformations in a probabilistic setting.

The NLP task that we investigate is the automatic correction of grammar errors made by English language learners. In contrast to spelling correction, which can be performed with a very high accuracy, the performance of grammar correction systems is still low for most error types. Commercial grammar correction systems mostly use rule-based methods. The most common approach in recent grammatical error correction research is to use statistical classifiers that make local decisions about the occurrence of specific error types. The approach that we investigate is related to a number of other approaches inspired by statistical machine translation (SMT) or based on language modelling. Corpora of language learner writing annotated with error corrections are used as training data.

Our baseline model is a noisy-channel FST model consisting of an n-gram language model and a FST error model, which performs word insertion, deletion and replacement operations. The tree transducer model we use to perform error correction is a weighted top-down tree-to-string transducer, formulated to perform transformations between parse trees of correct sentences and incorrect sentences. Using an algorithm developed for syntax-based SMT, transducer rules are extracted from training data of which the correct version of sentences have been parsed. Rule weights are also estimated from the training data. Hypothesis sentences generated by the tree transducer are reranked using an n-gram language model.

We perform experiments to evaluate the performance of different configurations of the proposed models. In our implementation an existing tree transducer toolkit is used. To make decoding time feasible sentences are split into clauses and heuristic pruning is performed during decoding. We consider different modelling choices in the construction of transducer rules. The evaluation of our models is based on precision and recall. Experiments are performed to correct various error types on two learner corpora. The results show that our system is competitive with existing approaches on several error types.

Uittreksel

Ons ondersoek die toepassing van geweegde boomoutomate om grammatikafoute in natuurlike taal outomaties reg te stel. Geweegde eindigetoestand outomate word suksesvol gebruik in 'n wye omvang van take in natuurlike taalverwerking, alhoewel die uitdrukkingskrag van die taalkundige transformasies wat hulle uitvoer beperk is. Daar is die afgelope tyd 'n toename in die gebruik van geweegde boomoutomate en verwante formalismes wat sintaktiese transformasies in natuurlike taal in 'n probabilistiese raamwerk voorstel.

Die natuurlike taalverwerkingstoepassing wat ons ondersoek is die outomatiese regstelling van taalfoute wat gemaak word deur Engelse taalleerders. Terwyl speltoetsing in Engels met 'n baie hoë akkuraatheid gedoen kan word, is die prestasie van taalregstellingstelsels nog relatief swak vir meeste fouttipes. Kommersiële taalregstellingstelsels maak oorwegend gebruik van reël-gebaseerde metodes. Die algemeenste benadering in onlangse navorsing oor grammatikale foutkorreksie is om statistiese klassifiseerders wat plaaslike besluite oor die voorkoms van spesifieke fouttipes maak te gebruik. Die benadering wat ons ondersoek is verwant aan 'n aantal ander benaderings wat geïnspireer is deur statistiese masjienvertaling of op taalmodellering gebaseer is. Korpora van taalleerderskryfwerk wat met foutregstellings geannoteer is, word as afrigdata gebruik.

Ons kontrolestelsel is 'n geraaskanaal eindigetoestand outomaatmodel wat bestaan uit 'n n-gram taalmodel en 'n foutmodel wat invoegings-, verwyderings- en vervangingsoperasies op woordvlak uitvoer. Die boomoutomaatmodel wat ons gebruik vir grammatikale foutkorreksie is 'n geweegde bo-na-onder boom-na-string omsetteroutomaat geformuleer om transformasies tussen sintaksbome van korrekte sinne en foutiewe sinne te maak. 'n Algoritme wat ontwikkel is vir sintaksgebaseerde statistiese masjienvertaling word gebruik om reëls te onttrek uit die afrigdata, waarvan sintaksontleding op die korrekte weergawe van die sinne gedoen is. Reëlgewigte word ook vanaf die afrigdata beraam. Hipotese-sinne gegenereer deur die boomoutomaat word herrangskik met behulp van 'n n-gram taalmodel.

Ons voer eksperimente uit om die doeltreffendheid van verskillende opstellings van die voorgestelde modelle te evalueer. In ons implementering word 'n bestaande boomoutomaat sagtewarepakket gebruik. Om die dekoderingstyd te verminder word sinne in frases verdeel en die soekruimte heuristies besnoei. Ons oorweeg verskeie modelleringskeuses in die samestelling van outomaatreëls. Die evaluering van ons modelle word gebaseer op presisie en herroepvermoë. Eksperimente word uitgevoer om verskeie fouttipes reg te maak op twee leerderkorpora. Die resultate wys dat ons model kompeterend is met bestaande benaderings op verskeie fouttipes.

Acknowledgements

I would like to express my gratitude to the following people and organisations:

- My supervisor, Prof Brink van der Merwe, for his guidance and advice throughout this study.
- The MIH Media Lab, for financial support and for providing a stimulating research environment.
- Prof Lynette van Zijl and the Computer Science Division, for the opportunity to attend the 2012 International Winter School in Speech and Language Technologies.
- The management of the Stellenbosch High Performance Computer, which was used to perform some of the experiments in this thesis.
- The Canticum Novum choir, for providing necessary distractions from my studies.
- My parents, for their continued support.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms and Abbreviations

CLC	Cambridge Learner Corpus
CNF	Chomsky normal form
CoNLL	Conference for Natural Language Learning
EM	Expectation maximization
FCE	First Certificate in English
FSA	Finite-state automaton
FST	Finite-state transducer
GEC	Grammatical error correction
HMM	Hidden Markov model
HOO	Helping Our Own shared task
LHS	Left hand side
LM	Language model
MLE	Maximum likelihood estimate
NUCLE	National University of Singapore corpus of learner English
(P)CFG	(Probabilistic) context-free grammar
PGM	Probabilistic graphical model
POS	Part-of-speech
PTB	Penn Treebank
RHS	Right hand side
RTG	Regular tree grammar
SCFG	Synchronous context-free grammar
(S)MT	(Statistical) machine translation
(S)TSG	(Synchronous) tree-substitution grammar
SVA	Subject-verb agreement
SVM	Support vector machine
WSJ	Wall Street Journal
(x)(L)(N)T	(Extended) (linear) (nondeleting) top-down tree transducer
(x)(L)(N)TS	(Extended) (linear) (nondeleting) top-down tree-to-string transducer

Chapter 1

Introduction

A central limitation of many practical natural language processing (NLP) systems today lies in their inability to fully model the syntax of natural language. A range of NLP tasks can be performed surprisingly well by using models based only on local word context. Yet, to handle more complex structures that occur in language, and to move closer to the way that humans process language, we need to incorporate models of syntax.

In this thesis we investigate the application of syntax-based tree transducer models to grammatical error correction. Grammatical error correction is an important, yet under-explored problem in NLP. It is especially important to those learning and using English as a second or foreign language. We use a class of models called weighted tree transducers to model syntax-based text transformations in a probabilistic setting. As the weights of these models are used to represent probability distributions, we also refer to this class as probabilistic tree transducers. Constructing and applying our models can also be seen as a machine learning problem.

1.1 Background

Automata theory (Sipser, 2006) is rooted in Turing's models of computation developed in the 1930s. Shannon (1948) was the first to use a Markov chain (which is a kind of weighted finite-state automaton) as a simple model for natural language. The noted linguist Noam Chomsky proposed different classes of automata as formal models for natural language (Chomsky, 1956). In particular, Chomsky argued that finite-state models are inadequate to describe language, and argued that (at least) context-free grammars are needed. Chomsky also introduced the use of trees to describe the syntax of sentences. In his linguistic theory of transformational grammar, a kernel of simple sentences is generated by a context-free grammar. All other sentences are constructed by repeated transformations on the syntax trees of kernel sentences. The idea of a tree transducer, a finite-state machine that performs these transformations, was inspired by transformational grammar. The tree transducer was formalised independently by Rounds (1970) and Thatcher (1970).

Chomsky was strongly opposed to empirical linguistics in general, and to probabilistic models in particular. He argued that "There is no general relation between the frequency of a string (or its component parts) and its grammaticalness." (Chomsky, 1956). He used the now famous sentence *Colourless green ideas sleep furiously* as an example of a sentence that is grammatically correct though without any meaning, and therefore unlikely to occur as a written sentence. Chomsky's influence led to a shift away from data-driven methods in linguistics. In the field of Artificial Intelligence, that developed out of Computer Science in the late 1950s, the focus was also primarily on symbolic methods. Only in the field of electrical engineering work was done in the stochastic paradigm, building on Shannon's models (Jurafsky and Martin, 2009, chap. 1). These stochastic methods became successful in automatic speech recognition, especially with the use of the hidden Markov model (HMM) (Rabiner and Juang, 1986). The success of HMMs led to the use of probabilistic finite-state models in many applications in natural language processing, including part-of-speech tagging, syntactic parsing and machine translation. A good example of this development is in machine translation. Early machine translation systems were based on large hand-crafted grammars. However, their success was limited due to the prevalence of ambiguity and irregularities in language. With the advent of statistical machine translation (SMT) (Brown *et al.*, 1993), it was shown that statistical methods with simpler underlying models could perform as well as or better than rule-based systems, without the cost of manual rule-construction.

In this past decade, the development of probabilistic data-driven methods accelerated (Jurafsky and Martin, 2009, chap. 1). The first reason for this was that large text corpora and annotated linguistic datasets became widely available. These resources led to competitive, standardized evaluations of different approaches. Secondly, the spread of high-performance computing allowed the development of models with computational and memory requirements simply not available earlier. Finally, there was a greater interplay between the fields of machine learning and NLP.

Though these advances improved the performance of finite-state methods, there are still limitations to their abilities. An example of their deficiencies can be seen in the output of SMT systems: Often, it broadly conveys the meaning of the sentence in the source language, but not as a well-formed sentence in the target language. Therefore some researchers started to use more expressive models, such as those used in rule-based systems, in a probabilistic setting. An example of this is syntax-based SMT (Yamada and Knight, 2001; Graehl *et al.*, 2008), where the translation is based on a transformation involving the syntactic structure of the source or target sentence. These probabilistic, syntax-based models were formalized with weighted tree transducers. However, simply adding syntax does not automatically improve the performance of a model. Several issues regarding parameterization and decoding should be addressed to make syntax-based models successful.

The main theoretical models that we study in this thesis are weighted tree automata and weighted tree transducers. Tree automata theory (Gécseg and Steinby, 1984; Comon *et al.*, 2007) was developed as a generalization of classical (string) automata theory. The main application areas are in compiler theory and in natural language processing. Weighted algorithms developed for finite-state transducers (Mohri, 2009) have been extended to weighted tree transducers (May, 2010). In contrast to the string case, where weighted finite-state transducers (FSTs) are a standardized model, there are a multitude of different tree transducer formalisms, which differ in their expressive power (Maletti *et al.*, 2009). Furthermore, there are transducers that perform tree-to-tree transformations as well as transducers for tree-to-string transformations. In the implementation of our tree transducer models we use the tree automata toolkit Tiburon (May and Knight, 2006).

1.2 Problem description

English is arguably the most widely used language in the world. It has more than 400 million native speakers and a similar number of bilingual speakers (Huddleston and Pullum, 2005, chap. 1). Many more people are learning English, or have some degree of proficiency in it. English is the most widely used language on the internet, and is the primary language used for international communication. We follow the terminology of Leacock *et al.* (2010) by using the term *language learner* to refer to people learning English both in predominantly English-speaking countries and in places where the dominant language is not English.

There is a growing need for tools to assist people speaking English as a second or foreign language in using the language correctly. Central to such tools are methods to automatically detect and correct (or suggest possible corrections to) grammatical errors. Spell checking is a well-studied problem: Today spelling mistakes can be detected automatically with a very high accuracy, and good corrections can be suggested in most cases. However, grammar correction is a more difficult problem, and it is an under-explored task in NLP. Word processing systems can detect some grammatical errors. However, they usually focus on first language speakers, while there are significant differences between the grammatical errors that first language speakers and language learners make (Leacock *et al.*, 2010, chap. 3).

By grammatical error correction (GEC) we mean the automated correction of grammatical errors that people make in writing. We constrain our study to that of errors made in English, due to the availability of large quantities of linguistic resources, and the significance of English as the language used most often as a second or additional language. However, the methods that we study are also applicable to other languages. Although a practical grammar correction system for language learners should include a component for spelling correction, we do not study spelling correction in this thesis. Our models can be applied to correct bad language on the internet (Eisenstein, 2013). However, we are not concerned with the specific characteristics of non-standard language usage on social media text.

Modern NLP has made it possible to build systems that are capable of detecting and correcting a reasonable subset of errors made by language learners (Leacock *et al.*, 2010, chap. 1). However, performance is still low in comparison to widely-studied NLP problems such as parsing, word sense disambiguation and information extraction. The use of statistical methods to perform linguistic analysis is not yet as widespread in practical GEC systems as for other NLP tasks such as machine translation.

Recently, shared tasks for GEC has led to an increase in attention in the NLP research community. The Helping Our Own (HOO) task was introduced by Dale and Kilgarriff (2010) with the intention of correcting errors in computational linguistics papers made by non-native English speakers. The 2011 task (Dale and Kilgarriff, 2011) used a dataset of text fragments extracted from computational linguistics papers, annotated with error types and suggested corrections. The 2012 task (Dale *et al.*, 2012) focussed on preposition and determiner correction in learner text, using a larger annotated dataset extracted from the Cambridge learner corpus. In 2013, GEC was the shared task for the high-profile Computational Natural Language Learning (CoNLL) conference (Ng *et al.*, 2013), focussing on a set of five common language learner errors. We entered a system based on the models in this thesis for the shared task, obtaining promising results (Buys and van der Merwe, 2013).

1.3 Approach

Grammatical error correction can be viewed as either a text classification or a text transformation problem. In the classification approach, words or phrases in the text are classified as either grammatical or ungrammatical. At incorrect positions, a correction is chosen from a finite set of alternatives and applied to the original text.

In the transformational approach, which we follow in this thesis, possibly incorrect sentences are transformed to corresponding correct sentences. Though most of the words are left unchanged, incorrect phrases are transformed to correct phrases. A criterion of these transformations is that they should perform the minimal number of edits to obtain a correct sentence. In this thesis we develop models based on weighted tree-to-string transducers to perform these transformations.

The most important modelling choice in constructing GEC models concerns the context taken into consideration when making classification decisions or performing phrasal transformations. Many grammar errors can be seen as the result of the incorrect use of the syntax of a language. Therefore we construct models that perform transformations based on syntactic context.

A tree-to-tree or tree-to-string transducer can be used to perform grammar correction. A problem with the tree-to-tree model is that to decode an incorrect sentence it should be parsed first. Parsing ungrammatical sentences is less accurate than parsing grammatical sentences, as the incorrect words in the sentence may lead the parser to make incorrect parsing decisions about other parts of the sentence. Therefore we rather use tree-to-string transducers. These transducers are formulated to transform the parse trees of correct sentences into corresponding incorrect sentences. During decoding a given (incorrect) sentence is transformed to a correct sentence, and parsed simultaneously.

As a baseline, we implement a FST model based on a n-gram language model and a word transformation model that does not consider any additional context. The main challenges in our tree transducer models lie in parameterization and decoding. Finding minimum edit distances is much more computationally expensive when working with trees than with strings. Therefore the rule set of the transducer should be constructed carefully. As the model takes more context into consideration, there are more parameters to be estimated from a limited amount of training data. The search space of the model is very large, so pruning should be performed to make decoding feasible, but without undermining the advantages of taking more context into consideration.

1.4 Objectives

The objectives of this study are

- to study the application of weighted tree transducers to natural language processing;
- to study weighted tree transducers as a class of probabilistic models;
- to investigate syntax-based models for grammatical error correction; and
- to develop a novel practical NLP system, based on probabilistic tree transducers, for grammatical error correction.

1.5 Thesis outline

In this section we provide a brief outline of the structure of this thesis. In Chapter 2 we discuss English grammar and grammar errors, and review existing models of grammatical error correction. The theoretical models (automata, grammars and transducers) used in this thesis are defined in Chapter 3, with examples of how they are applied to model natural language. We present representation, inference and training algorithms for probabilistic models in Chapter 4. Language models, syntactic parsing, and probabilistic models for the automata theoretic models are discussed.

The experimental setup of the models that we develop is set out in Chapter 5. A baseline FST model is also given. We present the tree transducer models that we propose for grammatical error correction in Chapter 6. Results of our model are given and analysed in Chapter 7. We conclude this work by summarizing our contributions and suggesting future work, in Chapter 8.

Chapter 2

Grammatical Error Correction

In this chapter we present background to the problem of grammatical error correction. We review English grammar and different types of grammar errors. Training and testing data used to develop GEC systems, as well as GEC evaluation metrics are discussed. Then we review the main categories of GEC models: Statistical classifiers, rule-based models, language modelling approaches and SMT-inspired approaches. Leacock *et al.* (2010) give a recent, comprehensive review of grammatical error correction. In the presentation of this chapter we often follow their terminology and analysis.

2.1 English grammar

In this section we review some relevant concepts in English grammar. Firstly, the goals of grammar and varieties in language are discussed. Then we present the main components of syntactic structure.

2.1.1 Goals of grammar

A *grammar* of a language describes the principles and rules governing the form and meaning of words, phrases, clauses, and sentences (Huddleston and Pullum, 2002). *Usage* refers to the choice of words in a given context. Though multiple word choices may be grammatically correct and have the same meaning, there is often a preference among native speakers for some word choice above others. Though usage is not strictly seen as part of the grammar of a language, it plays an important role in NLP models. In the field of grammatical error correction, usage errors are usually included in the kind of errors to be corrected (Leacock *et al.*, 2010, chap. 1). In most cases there is little difference among native speakers of a language regarding judgements of pure grammaticality. However, there may be greater variation in judgements of usage preferences.

There are many varieties of English. We mention some distinctions. What is known as *standard English* is the international norm, with a few points of difference between American and British English. There are also *non-standard* forms of usage (Huddleston and Pullum, 2005, chap. 1). Furthermore a distinction is made between *formal* and *informal styles*, which are used in different contexts. The type of language used on social media can also be seen as a rapidly developing non-standard form of English. Social media text has non-standard punctuation, capitalization, spelling, vocabulary, and syntax (Eisenstein, 2013).

Example 2.1.1 In the sentences below, (1a) is an example of non-standard usage, while (1b) is the corresponding standard form. Sentence (2a) is an example of informal style. The formal equivalent in (2b) needs to be used only in very formal contexts.

1. a) *I ain't told nobody nothing.*
   b) *I haven't told anybody anything.*
2. a) *He was the one she worked with.*
   b) *He was the one with whom she worked.*

A grammar can have either *descriptive* or *prescriptive* goals. Descriptive grammar describes a language as it is used by people who speak and write the language, while prescriptive grammar prescribes how the language should be used. Prescriptive grammars often do not distinguish between informal style and ungrammatical usage. For example, prescriptive grammar may judge the sentence *It is me* to be ungrammatical, even though it is widely used by native speakers.

A variation in language that does not meet some prescriptive standard should not *per se* be seen as ungrammatical. The goal of grammatical error correction should not be to enforce strict prescriptive constraints on how a language should be used; it should rather be to correct mistakes where written constructions do not conform to standard usage of the language.

Grammar can be divided into two components: *Morphology* and *syntax*. Morphology is concerned with the internal structure of words, while syntax is concerned with the way that words combine to form sentences. Grammar also interacts with other levels of description of language (Huddleston and Pullum, 2005, chap. 1). *Semantics* deals with the principles of how sentences are associated with their (literal) meanings. *Pragmatics* refers to the use and interpretation of sentences as they are used in particular contexts.

Although the focus of our models is on syntax, we briefly mention one relevant concept from morphology. Different words are associated with the same *lexeme* if they are different forms of essentially the same word. For example, *cat* and *cats* are singular and plural forms of the same noun lexeme, while *kick*, *kicks* and *kicked* are different forms of the same verb lexeme. Different forms of the same lexeme are called *inflections*. An inflection is usually indicated by a suffix appended to the base form of the lexeme.

2.1.2 Syntactic structure

We give a brief overview of the most important elements of English syntax, mostly following the terminology of Huddleston and Pullum (2005).

Two basic kinds of grammatical elements are *functions* and *categories*. A function is a relational concept, indicating a relation between a word or phrase and another

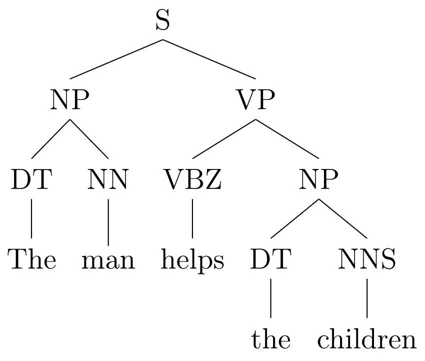


Figure 2.1: A constituency parse tree.

word or phrase. A category is a class of words or phrases that are grammatically alike. A function may be realized by multiple categories, while a category may perform different functions in different contexts.

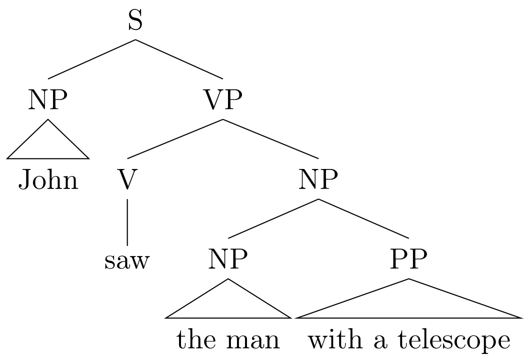
Fundamental to phrase structure (or constituency) grammar is the idea that a phrase labeled with a category, called a *constituency phrase*, can behave as a single unit. The most important word in a constituency phrase, that usually determines the phrase category, is called the *head* of the phrase. The other words in the phrase are called *dependents*. Constituency structure is hierarchical: Two constituency phrases are either disjoint, or one phrase is a subphrase of the other. Consequently, constituency structure can be represented as a tree, called a *parse tree*. Chomsky (1957) first proposed phrase structure grammars to formalize English grammar. A parse tree for the sentence *The man helps the children* is shown in Figure 2.1.

The term *part-of-speech* (POS) refers to a category of words. A standard list of parts-of-speech is noun, verb, adjective, adverb, determiner, preposition and conjunction. However, lists of POS categories (also called *tagsets*) differ in level of granularity. The Penn Treebank POS tagset (Santorini, 1990), which we use in our models, has 36 tags. This tagset is given in Appendix A.

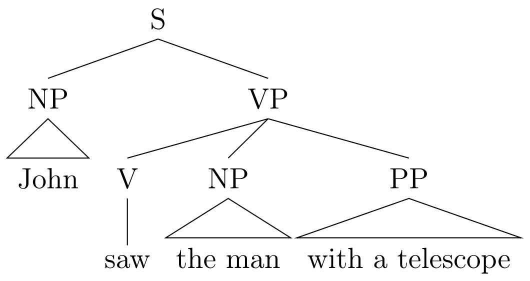
Parts-of-speech can be classified into *closed* and *open* class categories. Closed classes have a relatively fixed membership, while new words are continuously being added to open classes. The open POS classes are nouns, verbs, adjectives and adverbs. Many words have multiple possible POS tags. For example, the words *map* and *drink* can be nouns or verbs, depending on the context they are used in. *POS tagging* is the problem of assigning a POS tag to each word in a sentence. State-of-the-art taggers such as the Stanford POS tagger (Toutanova *et al.*, 2003) can achieve accuracies of around 97%. A syntactic phrase is usually labeled with the POS of its head word. The main phrase categories are noun, verb, adjective, adverb and prepositional phrases.

A basic category in the syntax of sentences is the *clause*, which usually consists of a phrase with a *subject* function, followed by a phrase with a *predicate* function. A sentence that has the form of a clause is a *clausal* sentence. Sentences often consist of two or more clauses. Later in this thesis, we present an algorithm to split sentences heuristically into clauses.

The basic clause types are declarative, interrogative and imperative clauses.



(a)



(b)

Figure 2.2: An example of prepositional phrase attachment ambiguity.

Characteristically, declarative clauses make statements, interrogatives ask questions, and imperatives issue directives, including requests, commands and instructions.

Example 2.1.2 Sentences below illustrate these concepts. In the clausal sentence (1), the subject is the noun *things* and the predicate is the verb *change*. Sentence (2) is also a clausal sentence, with a noun phrase subject *the man* and a verb phrase object *kicks the ball*. In this sentence, the noun phrase *the ball* is an *object*. Sentence (3) is a compound sentence with two clauses. The clauses in the first three sentences are declarative. Sentence (4) is an interrogative clause, while (5) is an imperative.

1. *Things change.*
2. *The man kicks the ball.*
3. *He kicks the ball, and she catches it.*
4. *Did he kick the ball?*
5. *Kick the ball!*

Just as words can have multiple POS tags, sentences can have multiple parses. Two examples of *syntactic ambiguity*, as this is referred to, are *attachment ambiguity* and *coordination ambiguity* (Jurafsky and Martin, 2009, chap. 13). In coordination ambiguity, the problem is to find the boundaries of phrases which are joined by a coordinating conjunction. For example, the phrase *nationwide television and radio* can be parsed into nested noun phrases as either *[nationwide [television and radio]]* or *[[nationwide television] and radio]*.

The classic example of prepositional phrase attachment ambiguity is given in Figure 2.2. Two parses of the sentence *He saw the man with a telescope* are given. In (a), the prepositional phrase modifies the noun phrase *the man*, while in (b) the verb *saw* is modified. The two parses denote different possible meanings of the sentence. In (a) a man that has a telescope is seen by John, while in (b), John is looking through a telescope when he sees the man.

A *treebank* is a syntactically annotated corpus. Every sentence is syntactically annotated with a parse tree. The Penn Treebank (PTB) project (Marcus *et al.*, 1993) produces English treebanks. The PTB syntactic annotations are described extensively in Bies *et al.* (1995). The PTB syntactic tagset, used for syntax trees in this thesis, is given in Appendix A.

An alternative class of grammar formalisms is called *dependency grammars*. Dependency grammars describe the structure of sentences in terms of binary syntactic or semantic relations between words (Jurafsky and Martin, 2009, chap. 12). A dependency parse is represented as a directed acyclic graph with the words in the sentence as nodes. In a *typed* dependency parse, edges representing word relations are labeled. Typically, labels express functional relations between words, though category labels may also be included. A widely used dependency grammar formalism is the Stanford typed dependencies representation (De Marneffe *et al.*, 2006).

2.2 Grammar errors

The focus of research in grammatical error correction is on the errors that language learners make. Language learners with different levels of proficiency in English differ in the kind and number of errors they make. However, learners that already have a high level of language proficiency still tend to make errors that first language speakers rarely make. We should also note that there are significant differences between the distributions of grammar mistakes made by first and by second language speakers. Interestingly, many of the most common errors made by first language speakers occur in complex constructions that are usually avoided by language learners (Leacock *et al.*, 2010, chap. 3).

In this section we introduce language learner corpora and discuss some common grammatical errors that we deal with in this thesis.

2.2.1 Language learner corpora

The main source of data regarding language learner errors is *language learner corpora*. These corpora usually consist of essays written by language learners for some language course or examination. Most language learner corpora are not freely available, and only some are error-annotated. This has until recently been an obstacle to research in grammatical error correction. A comprehensive list of language learner corpora can be found in Leacock *et al.* (2010, chap. 4). We discuss the two corpora used in developing grammar correction models in this thesis.

Error type	Percentage
Replace word errors	36.92
Missing word errors	19.72
Unnecessary word errors	11.11
Tense errors	8.05
Word form errors	6.88
Agreement errors	5.31
Derivation errors	4.68
Inflection errors	1.59
Count errors	0.65
Other errors	5.09

Table 2.1: Errors in the FCE Corpus by error type.

2.2.1.1 CLC FCE

The Cambridge Learner Corpus (CLC) (Nicholls, 2003) is the world's largest corpus of English learner writing. The corpus consists of a large collection of English Language competency examination scripts: It has at least 16 million words, of which a large portion has been error-annotated. The CLC is not publicly available. However, an extract of the corpus taken from the First Certificate in English (FCE) exam has been made publicly available (Yannakoudakis *et al.*, 2011). The dataset consists of 1141 annotated essays, totalling about 500 000 words. An additional 97 essays are designated as test data. The annotations include error types (using the CLC error annotation scheme), suggested corrections, as well as meta-data on the age and first language of the writer of each essay. Since learners taking the FCE test have a relatively low proficiency in English, the essays contain a large number of errors. The FCE dataset was used as training data for the HOO 2012 shared task, though a different test set, not publicly available, was used.

The CLC error classification has two dimensions: The nature of the error (word insertion, deletion, form, etc.) and the word category of the error (verb, noun, preposition, etc.). A breakdown of the relative error frequencies of these two classifications in the FCE dataset are given in Tables 2.1 and 2.2. Note that spelling errors, which are also annotated in the corpus, are excluded from this analysis.

2.2.1.2 NUCLE

The National University of Singapore Corpus of Learner English (NUCLE) (Dahlmeier *et al.*, 2013) is another large, fully error-annotated corpus. It consists of 1414 essays written by students from the National University of Singapore, totalling more that 1.2 million words. It is available free of charge, but with a licence agreement. The second version of this dataset was released as training data for the CoNLL-2013 shared task in grammatical error correction. A blind test set consisting of 50 essays was used for the shared task and released after the evaluation.

The grammar errors in NUCLE are very sparse, as the students who wrote the essays already had a relatively high proficiency in English. In the corpus, 57.6% of sentences have no errors, while only 11.2% of sentences have more that two errors.

Error category	Percentage
Verb errors	23.03
Punctuation errors	15.63
Preposition errors	12.12
Determiner errors	11.36
Noun errors	10.24
Pronoun errors	5.45
Adjective errors	5.45
Adverb errors	3.96
Word order errors	3.25
Conjunction errors	1.56
Quantifier errors	1.13
Other errors	12.27

Table 2.2: Errors in the FCE Corpus by word category.

The first language of most of the learners whose essays are included in NUCLE is Chinese.

A breakdown of error types in the NUCLE corpus, that has its own error annotation scheme, is given in Table 2.3. The error classification is a mix of word category classes and more specific error types. A breakdown of the relative frequency of each of the error types considered in the CoNLL-2013 shared task is given in Table 2.4. From the table it is clear that the relative frequency of errors is much higher on the test data than on the training data.

2.2.1.3 Annotator agreement

A challenge in the annotation of learner corpora is that there may be significant differences among human annotators over the grammaticality of constructions, and what the best correction for an incorrect word of a phrase is. Annotators may also miss some of the incorrect constructions. Usage errors is a particular source of disagreement among annotators. An annotator agreement study on a subset of the NUCLE corpus shows significant disagreement among annotators (Dahlmeier *et al.*, 2013). Annotator agreement is measured by Cohen's kappa coefficient, defined as

$$\kappa = \frac{P(a) - P(e)}{1 - P(e)}, \tag{2.2.1}$$

where $P(a)$ is the probability of agreement between two annotators and $P(e)$ is the probability of chance agreement. Kappa scores between 0.2 and 0.4 are considered *fair*, and scores between 0.4 and 0.6 *moderate*.

On the subset of NUCLE annotated by multiple annotators, the kappa for agreement of annotated tokens (disregarding the error type and correction) is 0.388, while the kappa of the error class and correction, given the identification, is 0.484.

As a consequence, when evaluating error correction systems against annotated test data, one must bear in mind that annotator disagreement places an upper bound on expected system performance.

Error type	Percentage
Article or Determiner	14.75
Wrong collocation or idiom	11.82
Local redundancy	10.50
Noun number	8.37
Verb tense	7.14
Punctuation and spelling	7.01
Preposition	5.37
Word form	4.82
Subject-verb agreement	3.38
Other errors	3.29
Verb form	3.22
Link word or phrases	3.06
Unclear meaning	2.65
Pronoun reference	2.06
Runons, comma splice	1.95
Incorrect sentence form	1.48
Citation	1.47
Tone	1.32
Parallelism	1.15
Verb modal	0.96
Missing verb	0.91
Subordinate clause	0.81
Adverb or adjective position	0.75
Fragment	0.56
Noun possessive	0.54
Pronoun form	0.41
Dangling modifier	0.12
Acronyms	0.11

Table 2.3: Errors in the NUCLE Corpus by error type.

Error type	Training set	Test set
Determiner	5.73	23.62
Preposition	2.07	10.68
Noun number	3.25	13.56
Verb form or SVA	2.56	8.42

Table 2.4: Number of errors per 1000 words in NUCLE, for different error types.

2.2.2 Common language learner errors

Many grammar errors are made as a result of differences between the first language of a speaker, and the second language which they are using. These errors are referred to as *transfer problems* (Leacock *et al.*, 2010, chap. 3). Transfer problems are most apparent where a feature in the second language (in our case English) is not present in the learner's first language. Subtle differences between languages with similar but not identical grammatical constructions can also cause problems. We now discuss some of the common errors made by language learners.

Most *determiner errors* are article errors. The correction of article errors requires word insertion or deletion more than most other error types. The main reason for the high occurrence of determiner errors is that many languages (for example, Chinese and Russian) do not have articles. It has been shown that there is a considerable difference in the frequency of article errors made by English learners whose first language does not have articles, and those that do (Leacock *et al.*, 2010, chap. 3). There are also some differences in the use of articles between languages that do have articles, which may lead to transfer errors. For example, for some constructions an article is included in English, but not in the German equivalent. The choice of whether an article should be included or excluded before a noun phrase is dependent on the noun, its grammatical context and its semantics. The *countability* of a noun determines whether it may take the indefinite article (*a/an*). Some nouns are countable in some usage contexts but not in others. Pragmatics may also play a role in the choice of article.

Another common type of closed class word category error is *preposition errors.* A many-to-many correspondence between prepositions in different languages leads to transfer errors. Prepositions are often used as syntactic rather than semantic constructions. The choice of preposition may be governed by the verb of the clause in which it occurs, or by the noun phrase following the preposition. Many phrasal verbs consist of a verb and a preposition (e.g. *give in, hold on, catch up*). Prepositional phrase attachment ambiguity further increases the difficulty for language learners to choose the correct preposition in a given context.

The open class word category with the highest error frequency is that of verbs. *Verb errors* include incorrect inflections (e.g. *eated/ate*) and wrong tense errors (e.g. *eat/ate/ has eaten*). Another important type of verb error involves *subject-verb agreement* (SVA). In English, the verb that follows a third person singular subject has a distinct form, usually adding *-s* or *-es* as suffix (e.g. *I eat/he eats*). Many languages (e.g. French and German) have much more complicated agreement rules than English, while others (such as Afrikaans) have almost no special verb forms. Most agreement errors made by language learners occur when the head noun of the subject noun phrase does not directly precede the verb. This makes it particularly challenging to detect these errors.

Noun errors usually occur when an incorrect or invalid form of a noun is used. The most common noun mistake is *noun number errors*, i.e. incorrect singular or plural noun forms. Some languages, including Chinese, do not have distinct plural forms for most nouns, again leading to transfer errors.

Collocation errors involve the incorrect usage of conventional combinations of words. Mastering these preferences for some word combinations over others is a significant challenge for language learners. Tests show that learners obtain very low scores in exercises testing the correct usage of word combinations (Leacock *et al.*, 2010, chap. 3). It has been found that around 40% of verb-object constructions are collocations (e.g. *throw a party, hold an election*). Other constructions that are frequently used as collocations include adjective-noun, noun-noun and verb-adverb POS combinations. There is some overlap between collocation errors and some other errors classified by word category.

A class of errors that we will not consider in this thesis, but which also occurs quite frequently, is *punctuation errors*. The most common punctuation errors involve the incorrect usage of commas.

2.3 Training and test data

Most approaches in the literature that are relevant for this thesis consider grammatical error correction as a machine learning problem. Therefore, training and test data are required to construct and evaluate systems. The three main sources of training and test data are well-formed text corpora, learner corpora and artificial error corpora.

2.3.1 Well-formed text corpora

The first resource, used by almost all error correction systems in some way, is large text corpora consisting of correct sentences. Examples of corpora frequently used include the Gigaword corpora, the British National Corpus and the Wall Street Journal (WSJ) corpus.

To understand how well-formed text is used, a distinction should be made between the problems of *selection* and *correction* of a word at a sentence position. The selection problem is to predict a word (for example, an article or preposition) at some position in a sentence that has been left blank. The word should usually be chosen from a *confusion set* of words. The correction problem is to find the correct word given a possibly incorrect word at a sentence position. In both cases the given or predicted word may be the empty string.

Well-formed text is used to train and test models for the selection task. It is also used to train language models, which are often a component in a GEC system. Well-formed text cannot be used to train correction models, as it does not contain pairs of incorrect and correct usage instances. However, a selection model can still be used to perform grammar correction, though the original word will not be used as a feature.

2.3.2 Learner corpora

The most appropriate training and testing data for grammatical error correction systems are annotated learner corpora. Learner corpora were discussed in Section 2.2.1. The main limitation in using learner corpora to train GEC systems is data sparseness. Only a few large annotated corpora are available. Even in these corpora, the occurrence of errors is sparse: Many errors in a corpus appear only a few times, and definitely not in all contexts in which they possibly may occur. For many open class words, none of the errors that may be associated with it may appear in a corpus.

2.3.3 Artificial error corpora

In the absence of sufficiently large corpora, an alternative is to use an artificially created error corpus. Such a corpus is created by inserting errors of certain types into a corpus of well-formed text with some stochastic process. A parallel corpus of correct and incorrect sentences, that can be used to train error correction systems, is obtained in this way. Training a machine learning system from artificial training data is not an ideal solution, as the system is not learning a true distribution of the occurrence of errors in the text. Still, it has been shown that such approaches can be helpful in building automatic error correction systems.

Foster and Andersen (2009) propose a system for automatically generating erroneous sentences from well-formed text. The type of errors to be generated and the desired proportion of errors of each type can be set as parameters. These parameters can be set manually, or estimated from the errors in a language learner corpus. In the latter case, the system can generate a large artificial corpus that mimics the characteristics of a small learner corpus, while containing a wider range of error examples.

2.4 Evaluation

The standard automatic evaluation metrics used for grammatical error correction are precision, recall and F1 scores. The selection task is usually evaluated by *accuracy*, the proportion of correct predictions made at the considered positions.

To perform evaluation an annotated set of test sentences is used. The annotated corrections of the test set is referred to as the *gold standard.* The test sentence corrections proposed by a GEC system is evaluated against the gold standard corrections. Changes made to incorrect sentences are represented by edit sequences.

Let s be the number of edits made by the system, g be the number of gold standard edits, and c be the number of correct edits, i.e. system edits that are also gold standard edits. Suppose that there are n test sentences. The sufficient statistics evaluating system performance on the ith sentence is the 3-tuple (s_i, g_i, c_i), such that $s = \sum_{i=1}^{n} s_i$, $g = \sum_{i=1}^{n} g_i$ and $c = \sum_{i=1}^{n} c_i$. The precision p, recall r, and F1 score f are defined as follows:

$$p = \frac{c}{s} \tag{2.4.1}$$

$$r = \frac{c}{g} \tag{2.4.2}$$

$$f = \frac{2pr}{p+r} \tag{2.4.3}$$

Precision is the proportion of system edits which match gold standard edits, while recall is the proportion of gold standard edits which were made by the system. The F1 score is the harmonic mean of the precision and recall scores. Scores are usually expressed as percentages.

In the HOO 2011 and 2012 shared tasks (Dale and Kilgarriff, 2011; Dale *et al.*, 2012), precision, recall and F1 scores were computed for the detection, recognition and correction of errors. *Detection* measures how well the system determines that some edits must occur in the text, while *recognition* measures how well the system determines the exact positions of where edits must occur. In this thesis we are more interested in how well a system performs corrections than in how well it detect errors. Therefore we focus on evaluating error correction.

Dahlmeier and Ng (2012*b*) suggest an approach to compute the sufficient statistics for GEC evaluation from system output sentences. In general, there may be multiple edit sequences that result in the same system output sentence when applied to the original sentence. As the gold standard is usually represented as an edit sequence, the system edit sequence that matches the gold standard edit sequence as closely as possible will lead to the most accurate evaluation result. Dahlmeier and Ng (2012*b*) represent possible system edit sequences in a lattice, using Levenstein edit distances as weights. Weights of edges corresponding to gold standard edit operations are then modified to have a negative weight. It is proven that the shortest path through the lattice corresponds to the optimal edit sequence. That edit sequence is used to compute the evaluation scores. The M^2 scorer, an open-source implementation of this approach, is used as the official scorer for the CoNLL-2013 shared task.

2.5 Existing GEC models

Next we review the principal approaches to grammatical error correction in the literature.

2.5.1 Classification-based approaches

There is a growing body of literature on the use of statistical classifiers to correct specific types of grammar errors. Most of the research on these approaches focus on article and preposition errors. Most system entries in the 2011 and 2012 HOO shared tasks fall under this approach. There are two model categories: Models trained on well-formed text, and models trained on learner text. We review models in both these categories.

State-of-the-art article selection systems achieve accuracies of around 90% when evaluated on well-formed text. For preposition selection, accuracies of around 75% can be obtained. The performance of correction systems is much worse when evaluated on learner data. The best systems achieve a recall of about 40% for determiner correction and 20% for preposition correction, with a maximum precision of around 60%. An important insight in GEC research was that it is beneficial to include the original word as a feature in classifiers for the correction task.

2.5.1.1 Models trained from well-formed text

The seminal work on the classification approach to grammatical error correction was done by Knight and Chander (1994). They focussed on article correction of the output of Japanese-to-English machine translation. A decision tree was trained for each of the 1600 most frequently occurring head nouns in training data of well-formed text. The classifier was then applied to all head nouns in the target language output of the translation system. An accuracy of 81% was obtained on the noun phrases considered.

De Felice and Pulman (2008) use maximum entropy classifiers to correct determiners and prepositions. The confusion set for preposition correction is the 9 most frequent prepositions in the data. The feature set used include the POS tags of the word context and semantic information such as WordNet categories. Accuracies of 92% and 70% were obtained on determiner and preposition selection, respectively.

The Microsoft Research ESL Assistant (Gamon *et al.*, 2008) was available as a web-based service for English second language speakers from 2008 to 2011. The system uses decision tree classifiers for preposition and article errors. The *presence* classifier predicts whether an article or preposition should be present at a given position or not. If there should be an article or preposition, the *choice* classifier predicts the choice of article or preposition from a confusion set. The classifiers are trained with well-formed text from different domains. A large language model trained on the Gigaword Corpus is used to filter corrections suggested by the classifiers: If the proposed word is different from the original word, the change is only accepted if the LM score of the proposed sentence is higher than the LM score of the original sentence.

A systematic evaluation of linear classifiers trained for the selection task and applied to correction learner writing is carried out in Rozovskaya and Roth (2011). Using a consistent feature set, the averaged perceptron gave the best performance, followed by a naive Bayes classifier, a Language model and a count-based method. The averaged perceptron is a mistake-driven online learning algorithm that gives similar performance to logistic regression and support vector machines (SVMs), but is trained more efficiently. However, as is frequently the case in machine learning, the choice of features and the amount of training data has a greater influence on the results than the choice of classifier.

2.5.1.2 Models trained from learner text

Han *et al.* (2010) were the first to train a system using a large-scale corpus of learner text, using the Chungdahm corpus. They focus on preposition errors, and use a maximum-entropy model with features similar to that of systems trained from well-formed text, except that the original (possible incorrect) word choice is also taken into consideration. The model performs markedly better when trained on learner text than when trained on well-formed text, even when the size of the well-formed text is five times that of the learner text. When this method is evaluated on learner text, a precision of 82% and a recall of 13% is obtained.

Gamon (2010) extends the Microsoft Research ESL assistant to use a learner corpus as additional training data. Scores from classifiers and a language model are combined using a decision tree meta-classifier that is trained using the error-annotated Cambridge Learner Corpus.

Rozovskaya and Roth (2011) propose a technique to adapt a naive Bayes model to learner text by training the prior probability of the classifier from error-annotated learner text. This allows for easy adaptation of the model to the error distributions of learners with different native languages.

An approach to combine classifiers for selection and correction is presented by Dahlmeier and Ng (2011*b*). The authors use Alternation Structure Optimization, a machine learning algorithm that uses auxiliary problems to improve classification performance on a target problem by exploiting the common structure of these problems. In the case of grammar correction on learner text, the selection task is an informative auxiliary problem. Compared to baselines trained either only for the selection task, or only on leaner text, improved F1 scores are obtained for both article and preposition correction. Using the NUCLE corpus, F1 scores of 19.29% on article correction and 11.15% on preposition correction are achieved.

Dahlmeier and Ng (2012*a*) present a beam-search decoder for combining classifiers for specific error categories. The method enables the correction of sentences with multiple, interacting errors. The decoder performs an iterative search over sentence hypotheses. *Proposers* generate new hypotheses by making incremental changes to current hypotheses. *Experts* then score the grammatical correctness of new hypotheses. The beam width determines how many hypothesis are kept after each iteration. Error categories handled include spelling, articles, prepositions, punctuation and noun number errors.

The highest scoring submission in the HOO 2012 shared task is that of the National University of Singapore (Dahlmeier *et al.*, 2012). Their system uses a pipeline of linear classifiers; classifiers for determiner correction, replacement preposition correction, and missing and unwanted preposition correction are applied in turn. Each step involves feature extraction, classification and language model filtering. The classifiers are trained using confidence-weighted learning, a machine learning algorithm suitable for high dimensional, sparse feature spaces. The determiner correction classifier is trained to predict the correct article from the confusion set $\{\text{a/an, the}, \varepsilon\}$. The types of features used in the classifier include lexical, POS, head word, web n-gram count, dependency, preposition and verb object features. The replacement preposition correction classifier uses a set of 36 frequent prepositions as confusion set. Features similar (but not exactly equal) to that for determiner correction are used. For missing and unwanted preposition correction a separate binary classifier is trained for each of a set of 7 prepositions. In all classifiers, the observed word is also used as a feature. All corrections are filters with a language model: A correction is only accepted if it strictly increases the language model score of the sentence (normalized by the sentence length). The system obtained 63.9% precision and 31.9% recall on determiner correction, and 60.22% precision and 22.95% recall on preposition detection.

2.5.2 Rule-based and hybrid approaches

The earliest grammar checking tools, such as the *Unix Writer's Workbench* (MacDonald *et al.*, 1982), were based on string matching. Later, systems started using full linguistic analyses with hand-crafted grammars (Leacock *et al.*, 2010, chap. 2). A difference between traditional grammars and those needed for error detection is that the latter should be error-tolerant and capable of indicating that a parse contains a violation of standard grammatical constraints. Linguistically expressive grammar formalisms such as Head Driven Phrase Structure Grammar, Lexical Functional Grammar and Constraint Grammar are capable of doing this better than context-free grammars. Strategies to make grammars error-tolerant include over-generating parse trees and ranking them in order of number of constraints violated, introducing *mal-rules* to allow the generation of erroneous sentences, and fitting together partial parses.

The most widely-used grammar checker for native English speakers is arguably the one in Microsoft Word. The grammars in the Microsoft NLP system are based on Augmented Phrase Structure Grammars (APSGs) (Leacock *et al.*, 2010, chap. 2). Productions in APSGs can be annotated with linguistic restrictions on the left hand side and features and attributes on the right-hand side. In order to perform grammar correction, the APSG parse of a sentence is converted to a dependency graph that represents syntactic and semantic information. Further analysis converts the dependency graph to a high-level semantic graph that represents the meaning relations in the sentence. Resources used in this model include the MindNet ontology and large dictionaries with morphological information.

Leacock *et al.* (2010, chap. 8) argue that for some error types, manually constructed rules may be easier to develop than statistical ones. This is especially the case when only very local contextual information is needed to detect errors. One example of this is over-regularized verb inflection (e.g. *writed* instead of *wrote*). A list of irregular English verbs and their over-regularized forms can be constructed and applied to correct these verb inflection errors without any additional information. Some subject-verb agreement errors or noun number errors may also be handled with rule-based methods, though in many cases language learner errors of these types involve more complex word interactions.

Many practical GEC systems use heuristic rules for certain error types and statistical classifiers for others. For example, the Microsoft Research ESL Assistant has heuristic-based modules for errors related to verbs, nouns and adjectives. Examples of constructions handled heuristically include modal verbs, phrasal verbs, adjective word ordering, adjective/noun confusion and noun number errors (Leacock *et al.*, 2010, chap. 8). Heuristic rules are created manually by inspecting learner data. The focus is especially on constructing rules that achieve high precision.

2.5.3 Language modelling approaches

Another approach to GEC is to perform edits with the goal of maximizing the fluency of a phrase or sentence as judged by a language model.

An early statistical approach to grammatical error correction was proposed by Atwell (1987). A trigram model is constructed over POS sequences. In the test data trigrams with a low POS model score, or that occur frequently in error examples, are flagged as errors. If an alternative POS tag at a position leads to a higher model score, the word at that position is also flagged.

More recently, the availability of large text corpora has lead to better language modelling approaches for GEC. The Google n-gram corpus is a large-scale corpus of n-grams of length 1 to 5. Bergsma *et al.* (2009) use this corpus to perform preposition selection, achieving 71% accuracy on well-formed text. Another approach is to use counts from search engines: A phrase with many hits is more likely to be grammatical than a phrase with a low number of hits. However, such an approach can be unreliable as there is no guarantee that the number of hits will correspond to the true frequency of the phrase in all the text being searched through.

Lee and Seneff (2006) describe a system for correcting language learner errors that uses an n-gram language model and a PCFG to score possible corrections. Firstly a given incorrect sentence is reduced by removing all articles, prepositions and auxiliaries, changing nouns to their singular forms and verbs to their root forms. A *word lattice* of possible corrections is then generated so that articles, prepositions and auxiliaries may be inserted between any two words, and that nouns and verbs may be changed to any valid lexical form. The lattice is scored with an n-gram language model, and the k-best candidate sentences are extracted. These candidate sentences are then parsed with a PCFG. The parser scores are used to rerank the candidates to find the best candidate correction. Better results were obtained using the PCFG scores than when using only the n-gram model scores. A weakness of this approach is that although multiple correct sentences may have the same reduced representation, the system will always recover the same correct sentence from a single reduced representation.

Turner and Charniak (2007) propose a model for article selection based on the Charniak language model that uses the scores of a lexicalized parser to assign sentence probabilities. The WSJ PTB and 20 million additional words from the North American News Text Corpus are used to train the model. Article selection is done for each noun phrase from the confusion set of articles by choosing the article and noun phrase combination with the highest model probability. An accuracy of 86.63% is obtained in this case.

A task related to grammar correction is that of classifying sentences as grammatical or ungrammatical. A simple approach is to use the score that a statistical parser assigns to a sentence to judge its grammaticality. However, treebank-induced grammars are not well suited to do this. The main reason is that they assign parses to incorrect sentences without penalizing the parser score sufficiently. As a result they cannot discriminate well between ungrammatical sentences and grammatical sentences that occur with a low probability. Wagner *et al.* (2007) propose an approach that uses both a broad-coverage precision grammar (a hand-crafted lexical functional grammar), and a n-gram model to classify sentences. Ferraro *et al.* (2012) perform sentence classification with a SVM that uses parse tree fragments as features. The best model performance is obtained when the 50 000 highest frequency tree fragments with a maximum height of 3 are used as features. Context-free grammar productions read off directly from the parse trees are also used as features. An accuracy of 89.1% is obtained with this approach.

2.5.4 SMT-inspired approaches

The final class of approaches we consider are inspired by statistical machine translation. The noisy channel formulation is usually followed. Suppose that we want to find the best correct sentence $\hat{c}$ corresponding to a given incorrect sentence i. Then, applying Bayes' rule,

$$\hat{c} = \arg\max_{c} P(c|i) = \arg\max_{c} P(c) \cdot P(i|c). \tag{2.5.1}$$

The model $P(i|c)$ is called an error model, and $P(c)$ is a language model. The intuition behind the noisy channel model is that some original message (the correct sentence) has been corrupted during transmission, and the goal is to try and recover the original message from the received message (the incorrect sentence).

Brockett *et al.* (2006) propose the use of phrasal SMT techniques to correct mass noun errors. Their motivation is that grammar errors do not occur in isolation (as they are essentially seen by statistical classifiers) and often require phrasal rewrites.

A parallel corpus of correct sentences and artificially constructed incorrect sentences is used to train the error model. A test set from the Chinese Learner Error Corpus was used. A recall of 0.618 is obtained on the error type under consideration.

Ehsan and Faili (2013) propose a hybrid approach to grammar correction that combines a SMT approach with a rule-based grammar checker. An artificial error corpus was also used to train the SMT model. GEC models are constructed for English and for Persian. An F1 score of 14% is obtained when only the SMT system is used. This improves to 22.7% when a hybrid approach is followed.

Park and Levy (2011) take a noisy channel approach to grammar correction, formulated with weighted finite-state transducers. An n-gram language model and models for spelling errors, preposition and article choice errors, and word insertion errors are formulated as FSTs, which are composed to obtain a single model. The error models are trained with the EM algorithm on an unannotated learner corpus. This is an example of unsupervised training. The BLEU and METOER machine translation evaluation techniques were used to evaluate system corrections on the test set, using up to eight possible reference corrections.

Dahlmeier and Ng (2011a) use phrasal SMT to correct collocation errors. A paraphrase model that is constructed from models to translate between English and the first language of the language learners, in this case Chinese, and back. Phrasal translation models are extracted from a Chinese-English parallel corpus, in both directions. For an English phrase e and a foreign (Chinese) phrase f, the models $P(e|f)$ and $P(f|e)$ are trained. These probability models are used to construct a paraphrase model for English sentences. Let e_1 and e_2 be English phrases, then the paraphrase probability model is

$$P(e_1|e_2) = \sum_f P(e_1|f)P(f|e_2). \tag{2.5.2}$$

The SMT error correction system is based on this paraphrase model. The model is augmented with features for spelling, homophones, and synonyms.

Finally, Madnani *et al.* (2012) use *round-trip* machine translation to correct sentences. A given English sentence is translated into 8 different pivot languages and back to English, using Google Translate. The candidate corrections are aligned, and a lattice of corrections is constructed, such that paths through the lattice may contain corrections from different candidates. A greedy search through the lattice was found to give the best results.

2.6 Conclusion

In this chapter we surveyed English grammar and grammatical error correction. Section 2.1 gave some linguistic background. Section 2.2 gave an overview of grammar errors made by English language learners. Sources of training and test data were discussed in Section 2.3, while 2.4 discussed evaluation. In Section 2.5 we reviewed classification-based approaches (2.5.1), rule-based heuristics (2.5.2), language modelling (2.5.3) and SMT-inspired approaches (2.5.4).

The use of different methods for different error types, as well as significant differences between the test sets used to evaluate systems, makes it difficult to establish which methods currently are the highest-performing ones in this field. For most error types, it seems that statistical classifiers currently provide the best performance. However, all the features used by these classifiers can also be expressed by systems that are formulated as sentence rewriting models (such as the SMT-inspired approaches). Rule-based approaches still being used can be recast as probabilistic models: Their expressive formalisms can be combined with machine learning methods to automatically acquire the linguistic knowledge needed to perform grammar correction.

Chapter 3

Automata Theory

In this chapter we develop the formal models that are studied in this thesis. The foundation of these models was laid by Chomsky (1957; 1959) in his work on generative grammars as formal descriptions of natural language.

There are a few important differences between the models we study in this chapter and those of classical automata theory. Firstly, the models here are weighted; an automaton assigns a weight to input. Secondly, we focus on automata for trees, in contrast to string automata. Lastly, we are particularly interested in transducers, a class of formal models which take strings or trees as input, and produce strings or trees as output, for strings or trees in the domain.

After giving some preliminary definitions, we define weighted finite-state transducers and weighted finite-state automata. Then we move on to tree automata and regular tree grammars. Top-down tree transducers and tree-to-string transducers are defined. Finally, we discuss the kind of natural language transformations that can be expressed with different transducer classes. Throughout the chapter we give examples to motivate the use of these models in grammatical error correction.

3.1 Preliminaries

In this section we give formal definitions of concepts used in this chapter and throughout this thesis.

3.1.1 Sets, relations and alphabets

The set of non-negative integers (including 0) is denoted by $\mathbb{N}$. By $[n]$ we denote the subset $\{1, 2, \ldots, n\} \subset \mathbb{N}$. The set of reals is denoted by $\mathbb{R}$, and the non-negative reals by $\mathbb{R}_+$. We denote by $X = \{x_1, x_2, \ldots\}$ an infinite set of *variables*, and let $X_k = \{x_1, \ldots, x_k\}$.

Let A and B be sets. We refer to $R \subseteq A \times B$ as a *relation* on A and B. A relation on A is subset of $A \times A$. The set of all finite sequences over A is denoted by A^*. The *power set* of A, defined as the set of all subsets of A, is denoted by $\mathcal{P}(A)$.

Let H, I and J be sets. An $I \times J$ *matrix over* H is a mapping $\mathcal{M} : I \times J \to H$ (Fülöp and Vogler, 2009). The set of all $I \times J$ matrices over H is denoted by $H^{I \times J}$. We write $\mathcal{M}(i, j) \in H$ as $\mathcal{M}_{i,j}$. An I *vector over* H is defined similarly as a mapping $v : I \to H$, and an element $v(i) \in H$ is denoted by v_i. The set of all I-vectors over H is denoted by H^I. An $[n]$-vector v is denoted by $\mathbf{v}$.

An *alphabet* is a finite, nonempty set of symbols. Suppose Δ is an alphabet. Elements of Δ^* are called *strings*. A *string language* over Δ is any subset of Δ^*. The empty string is denoted by ε. We denote $(\Delta \cup \varepsilon)$ as Δ_ε. The *string concatenation* of strings a and b is denoted by $a \cdot b$.

A *ranked alphabet* (Σ, rk) is a tuple consisting of an alphabet Σ and a mapping $rk : \Sigma \to \mathcal{P}(\mathbb{N})$, such that $rk(\sigma)$ is finite and non-empty for all $\sigma \in \Sigma$. For each $\sigma \in \Sigma$, $rk(\sigma)$ is called the set of *ranks* of σ. For each $k \in \mathbb{N}$, let $\Sigma_k = \{\sigma \in \Sigma \mid k \in rk(\sigma)\}$, the set of all symbols of rank k. Note that it may happen that $\Sigma_k \cap \Sigma_l \neq \emptyset$ for some $k \neq l$. We often denote the ranked alphabet (Σ, rk) simply by Σ.

Example 3.1.1 Let $\Delta = \{a, b, c\}$ be an alphabet. The set $\{(a, a), (a, b), (b, c)\}$ is a relation on Δ. Some strings over Δ are a, ab and abb. The set of all strings where a is followed by any number of b's, also described by the regular expression ab^*, is a string language over Δ.

3.1.2 Trees

Most of the models studied in this thesis are based on trees. We give some definitions related to trees, following the definitions of Engelfriet (1975) and Fülöp and Vogler (2009).

Definition 3.1.1 *Let Σ be a ranked alphabet, with $(\, , \,) \notin \Sigma$. The set of all trees over Σ, denoted by T_Σ, is the smallest set T of strings over the alphabet $\Sigma \cup \{(\, , \,)\}$ such that:*

- *If $\sigma \in \Sigma_0$, then $\sigma \in T$.*
- *For every $k \in \mathbb{N}, k \geq 1$, if $\sigma \in \Sigma_k$ and $t_1, t_2, \ldots, t_k \in T$, then $\sigma(t_1 \ldots t_k) \in T$.*

For notational convenience, when $k = 0$, we denote $\sigma()$ by σ. A *tree language* over Σ is any subset of T_Σ. A set of trees that extends a tree language T_Σ may be denoted as follows:

Definition 3.1.2 *Let S be a set. The set of trees indexed by S, $T_\Sigma(S)$, is defined inductively:*

- *$S \cup \Sigma_0 \subseteq T_\Sigma(S)$.*
- *For $k \in \mathbb{N}, k \geq 1$, if $\sigma \in \Sigma_k$ and $t_1, t_2, \ldots, t_k \in T_\Sigma(S)$, then $\sigma(t_1 \ldots t_k) \in T_\Sigma(S)$.*

Let Q be a set of symbols of rank 1. Then we denote by $Q(X_k)$ the set of trees $\{q(x) \mid q \in Q, x \in X_k\}$, and by $Q(T_\Sigma)$ the set of trees $\{q(t) \mid q \in Q, t \in T_\Sigma\}$.

Next we introduce notation for modifying existing trees.

Definition 3.1.3 *Let $s_1, \ldots, s_n \in \Sigma_0 \cup S$ all be different symbols, and $\sigma_1, \ldots, \sigma_n \in T_\Sigma(S)$. The **tree concatenation** of $t \in T_\Sigma(S)$ with $\sigma_1, \ldots, \sigma_n$ at $s_1, \ldots, s_n$, denoted by $t\langle s_1 \leftarrow \sigma_1, \ldots, s_n \leftarrow \sigma_n\rangle$, is defined as follows:*

- *For* $\sigma \in \Sigma_0 \cup S$*,*

$$\sigma\langle s_1 \leftarrow \sigma_1, \ldots, s_n \leftarrow \sigma_n\rangle = \begin{cases} \sigma_i & \textit{if } \sigma = s_i, \\ \sigma & \textit{otherwise.} \end{cases}$$

- *For* $\sigma \in \Sigma_k$ *and* $t_1, \ldots, t_k \in T_\Sigma$*,*

$$\sigma(t_1 \ldots t_k)\langle \ldots \rangle = \sigma(t_1\langle \ldots \rangle \ldots t_k\langle \ldots \rangle),$$

where $\langle \ldots \rangle$ *abbreviates* $\langle s_1 \leftarrow \sigma_1, \ldots, s_n \leftarrow \sigma_n\rangle$*.*

Let $k \in \mathbb{N}, \sigma \in \Sigma_k$ and $t = \sigma(t_1, \ldots, t_k)$. The set of *positions* of a tree is defined by the function $pos : T_\Sigma \to \mathcal{P}(\mathbb{N}^*)$, where

$$pos(t) = \{\varepsilon\} \cup \{iv \mid 1 \leq i \leq k,\ v \in pos(t_i)\}.$$

For clarity, when we write a position, "." is used to separate integers. For example, 1.1 and 1.2.3.1 denote positions. The *label* of tree t at position $p \in \mathbb{N}^*$, $label_t : \mathbb{N}^* \to \Sigma$, is defined by

$$label_t(p) = \begin{cases} \sigma & \text{if } p = \varepsilon, \\ label_{t_i}(v) & \text{if } p = iv. \end{cases}$$

Alternatively, a tree can be defined as a finite, rooted, ordered and labeled directed graph that has no cycles. By means of the positions of a tree, there is a one-to-one mapping between the ranked symbols in the tree and nodes in the graph of the tree. The label of a node is the label of its position. For each position $v \in pos(t)$, there is an edge in the graph from the node at position v to the node at each position $vi \in pos(t)$. The node at v is called the *parent* node of the nodes at positions vi, and conversely the nodes at each position vi are called *child* nodes of the node at position v. A node corresponding to a rank k symbol has k children. The tree node that has no parents is called the *root* node. Tree nodes that have no children are called *leaf nodes*, and their positions *leaf positions*. The set of leaf positions of t is denoted by $leafpos(t)$. Trees are usually drawn with the root at the top and the leaves at the bottom. We shall use this directionality to refer to the top and the bottom of a tree. Edges are conventionally drawn without arrows, as their directionality is always downwards. The left-to-right ordering of the children of a node corresponds to the order of their positions.

The *yield* of a tree is the left-to-right concatenation of its leaf symbols. There is a *path* from node a to node b if the position of a is a prefix of the position of b. Then b is called a *descendent* of a and a is called an *ancestor* of b. A subtree of a tree t is determined by a node in t (the root node of the subtree) and all its descendants. The subtree of t rooted at position v is denoted by $t|_v$. The *size* of a tree t, denoted by $|t|$, is the number of elements in $pos(t)$, i.e., the number of nodes in the tree. The *height* of a tree t is a function $height : T_\Sigma \to \mathbb{N}$, defined recursively as

$$height(t) = \begin{cases} 1 & \text{if } pos(t) = \{\varepsilon\}, \\ 1 + \max\{height(t|_i) \mid 1 \leq i \leq rk(label_t(\varepsilon))\} & \text{otherwise.} \end{cases}$$

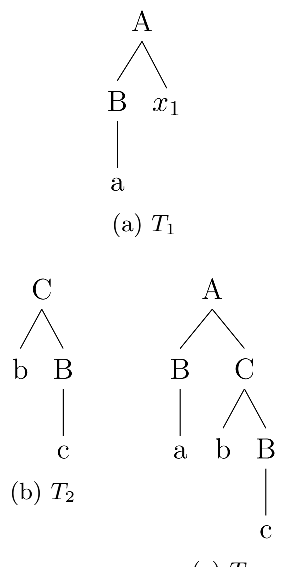


Figure 3.1: Example trees.

Let $s \in (T_\Sigma(X) \cup \Delta)^*$, where Σ is a ranked alphabet and Δ is an unranked alphabet. Then s is *linear* in X_k if each element of X_k occurs at most once in s, and *nondeleting* in X_k if each element of X_k occurs at least once in s. A tree $t \in T_\Sigma$ is *binarized* if none of the symbols in t have a rank greater than 2.

The set of *context trees* over Σ is the set of trees in $T_\Sigma(X)$ which are linear and nondeleting in X_1. In other words, a context tree is a tree with exactly one variable.

Example 3.1.2 Let Σ be a ranked alphabet such that $\Sigma_0 = \{a, b, c\}$, $\Sigma_1 = \{B\}$ and $\Sigma_2 = \{A, C\}$. In Figure 3.1, $T_1 \in T_\Sigma(X)$ is a context tree and $T_2, T_3 \in T_\Sigma$. T_3 is obtained by tree concatenation: $T_3 = T_1\langle x_1 \leftarrow T_2\rangle$. The set of positions of T_2 is $\{\varepsilon, 1, 2, 2.1\}$. The root node of T_2, at position ε, is labeled C. The leave positions of T_2 are 1 and 2.1, with corresponding labels b and c, respectively. The yield of T_2 is bc. In T_1, the node labeled A is an ancestor of the node labeled a, and conversely the node labeled a is a descendent of the node labeled A. The height of T_3 is 4.

3.1.3 Semirings

The automata and grammars that we study in this thesis are *weighted*. The weights and operations on weights are formalized by using *semirings*. This allows us to use an abstract notion of weights in our automata definitions.

For all $a, b, c \in W$, a binary operation $\oplus$ on set W is *associative* if $(a \oplus b) \oplus c = a \oplus (b \oplus c)$, and *commutative* if $a \oplus b = b \oplus a$. Operation $\otimes$ *distributes* over $\oplus$ if $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$.

Definition 3.1.4 (Mohri *et al.* (2002)) *A **semiring** is a tuple $(W, \oplus, \otimes)$, such that W is a set with an associative and commutative plus operation $\oplus$ and an associative times operation $\otimes$, with identities $\overline{0}$ and $\overline{1}$, respectively. Furthermore, $\otimes$ distributes over $\oplus$ and $\overline{0} \otimes a = a \otimes \overline{0} = \overline{0}$.*

Semiring	Set	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$
Boolean	$\{0, 1\}$	$\vee$	$\wedge$	0	1
Probability	$\mathbb{R}_+ \cup \{+\infty\}$	$+$	$\times$	0	1
Log	$\mathbb{R} \cup \{-\infty, +\infty\}$	$\oplus_{\log}$	$+$	$+\infty$	0
Tropical	$\mathbb{R} \cup \{-\infty, +\infty\}$	min	$+$	$+\infty$	0

Table 3.1: Important semirings.

Where there is no ambiguity we shall refer to both the semiring and its underlying set as W. The set in a semiring does not need to be closed under taking either additive or multiplicative inverses. This is the primary characteristic that distinguishes semirings from other algebraic structures such as rings or fields.

A semiring is *commutative* if $\otimes$ is commutative. A semiring is *complete* if the plus operator $\oplus$ is extended to infinite summations (see May (2010, chap. 2) or Mohri (2009) for complete definitions). In particular, a complete semiring W can be augmented by a unary closure operator *, defined by $w^* = \oplus_{i=0}^{\infty} w^i$ for any $w \in W$ ($w^0 = \bar{1}$ and $w^{n+1} = w^n \otimes w$ for all $n \in \mathbb{N}$).

Though we define automata later in this chapter, we briefly discuss the intuition behind weighted automata. In general, weights are multiplied to obtain the weight of a path or derivation, and added over different paths or derivations to obtain the weight of a string or a tree. Semirings that are important in our context are listed in Table 3.1 (cf. Mohri (2009)). These semirings are all commutative and complete. In this thesis we only deal with commutative and complete semirings.

Automata with weights over the Boolean semiring are semantically equivalent to unweighted automata. The weight of each path is either 0 or 1. Paths with weight 1 are *accepted* and paths with weights 0 are *rejected.* The unary closure operation is defined as $0^* = 1^* = 1$.

In the *probability* semiring, weights are usually interpreted as probabilities which are multiplied and added together. The unary closure operation is defined as $w^* = \frac{1}{1-w}$ for $0 \leq w < 1$ and $w^* = +\infty$ otherwise.

In the *log* and *tropical* semirings, weights usually represent negative log probabilities. In the log semiring, the plus operator is defined as $a \oplus_{\log} b = \log(\exp(a) + \exp(b))$.

An isomorphism between the probability and the log semirings is given by $h : \{\mathbb{R}_+ \cup +\infty\} \rightarrow \{\mathbb{R} \cup -\infty, +\infty\}$, where

$$h(p) = \begin{cases} +\infty & \text{if } p = 0, \\ -\log(p) & \text{if } p > 0,\ p \in \mathbb{R}_+, \\ -\infty & \text{if } p = +\infty. \end{cases}$$

In the tropical semiring, the weight that an automaton associates with a string or a tree is the weight of a path with the lowest weight. In the case of negative log probabilities, this weight corresponds to the weight of the path with the highest probability. This weight is also known as the *Viterbi* approximation of the weight of a string or a tree. In the tropical semiring, the unary closure operation is defined as $w^* = 0$ for $w \in \mathbb{R}_+$, and $w^* = -\infty$ otherwise.

3.2 Weighted finite-state transducers

The first automata formalism we define is weighted finite-state transducers (FSTs). FSTs are a class of finite automata in which each transition between states is augmented with an output label (in addition to the input label) and a weight. FST-based models have been applied widely in speech and language processing (Knight and Al-Onaizan, 1998; Mohri *et al.*, 2002; Pirinen and Lindén, 2010; Gispert *et al.*, 2010).

Definition 3.2.1 (Droste and Gastin (2009)) *A **weighted finite-state transducer** over a semiring W is a tuple $M = (Q, \Sigma, \Delta, \mu, \lambda, \rho)$, where*

- *Q is a finite set of states;*
- *Σ is an input alphabet;*
- *Δ is an output alphabet;*
- *$\mu : (\Sigma_\varepsilon) \times (\Delta_\varepsilon) \to W^{Q \times Q}$ is a transition mapping;*
- *$\lambda : Q \to W$ is an initial weight function; and*
- *$\rho : Q \to W$ is a final weight function.*

The weight of a transition from state p to state q $(p, q \in Q)$ that consumes the input-output pair $(\sigma, \delta) \in \Sigma_\varepsilon \times \Delta_\varepsilon$, is given by $\mu(\sigma, \delta)_{p,q}$. The pair (σ, δ) is referred to as the *transition label.* In a concrete representation of M, transitions with weight 0 are usually omitted.

Let $\sigma \in \Sigma$ and $\delta \in \Delta$. A transition labeled as (ε, δ) is an *input-ε* transition, while a (σ, ε) transition is called *output-ε*. Transitions labeled as $(\varepsilon, \varepsilon)$ are called ε-transitions.

Next we define the semantics of finite-state transducers. A *path* in M is an alternating sequence $p = (q_0, (a_1, b_1), q_1, \ldots, q_{n-1}, (a_n, b_n), q_n) \in Q \cdot ((\Sigma_\varepsilon \times \Delta_\varepsilon) \cdot Q)^*$. The *run weight* of path p is defined as

$$rw(p) = \lambda(q_0) \otimes_{1 \leq i \leq n} \mu(a_i, b_i)_{q_{i-1}, q_i} \otimes \rho(q_n), \tag{3.2.1}$$

where $rw(p) \in W$. The *label* of p is the input-output pair (s, t), where $s \in \Sigma^*$ and $t \in \Delta^*$, such that $s = a_1 \cdot a_2 \ldots a_n$ and $t = b_1 \cdot b_2 \ldots b_n$. Note that multiple paths through a transducer may have the same label.

A transducer M is *regulated* if the weight it associates with any pair $(s, t) \in \Sigma^* \times \Delta^*$, defined as

$$wt_M(s, t) = \oplus_{p:\text{label}(p)=(s,t)} rw(p), \tag{3.2.2}$$

is an element of W, such that $wt_M(s, t) \in W$ and its value is independent of the order of the $\oplus$ summation in (3.2.2). Specifically, for transducers without ε-transitions, there are a finite number of paths, and therefore such transducers are regulated. Transducers over a commutative and complete semiring are still regulated in the presence of ε-transitions, as infinite summations are defined. Therefore weighted transducers over the semirings defined in Section 3.1.3 are all regulated.

A *weighted finite-state automaton* (FSA) is a transducer with identical input and output symbols on all transitions. A transition label (a, a) can be denoted simply by a if it is clear that we are working with a FSA. The weight of FSA A on string s is denoted by $wt_A(s)$. The class of string languages accepted by FSA over the Boolean semiring is the class of regular languages.

A string s can be *embedded* in a FSA by constructing a FSA that associates weight $\bar{1}$ with s, and weight $\bar{0}$ to all other strings.

The *left projection* of FST M yields a FSA, denoted as $\downarrow M$, by omitting the output labels, such that

$$wt_{\downarrow M}(s) = \oplus_t wt_M(s, t). \tag{3.2.3}$$

Similarly, the right projection $M \downarrow$ can be obtained by omitting the input labels.

Transducers can be combined with the operation of *composition.* Suppose that M_1 and M_2 are FSTs over the same commutative and complete semiring W such that the output alphabet of M_1 and the input alphabet of M_2 coincides. Then the composition of M_1 and M_2 is a transducer $M = M_1 \odot M_2$, such that the weight that M associates with any string pair (s, t) is

$$wt_M(s, t) = \oplus_r wt_{M_1}(s, r) \otimes wt_{M_2}(r, t), \tag{3.2.4}$$

FSTs are *closed* under composition: For any two transducers M_1 and M_2 under the conditions above the composite transducer M can be constructed. Consequently, composition can be extended to a *cascade* of n transducers, $M_1, \ldots, M_n$, such that $M = M_1 \odot M_2 \odot \ldots \odot M_n$.

Example 3.2.1 A FSA L over the alphabet $\Delta = \{$*the, a, man, children, help, helps* $\}$ is given in Figure 3.2. L represents a bigram language model, which is a Markov model over the words in Δ. The alphabet of states is Δ_ε; each state represents the previous word in the path. Weights are defined over the tropical semiring. The initial weight function is 0.0 for the ε state and ∞ for all other states, and all states are assigned a final weight of 0.0. In language models, ε-transitions are used to indicate a backoff to a lower order model. The transition from state a to state ε is an ε-transition. An example path through the automaton is (ε, *the, the, man, man, helps, helps, the, the, children, children*), that has the label *the man helps the children.*

Example 3.2.2 A single-state FST R over the same alphabet Δ is given in Figure 3.3. This transducer is a simple error model that performs transformations between incorrect and correct sentences. As there is only one state, word transitions are independent of the path history. The transducer has input-ε and output-ε rules to insert or delete the article *the.*

Example 3.2.3 An embedded FSA E for the string *the man help the children* is given in Figure 3.4.

Examples of composition and projection are given in Section 4.4, where FST inference is discussed.

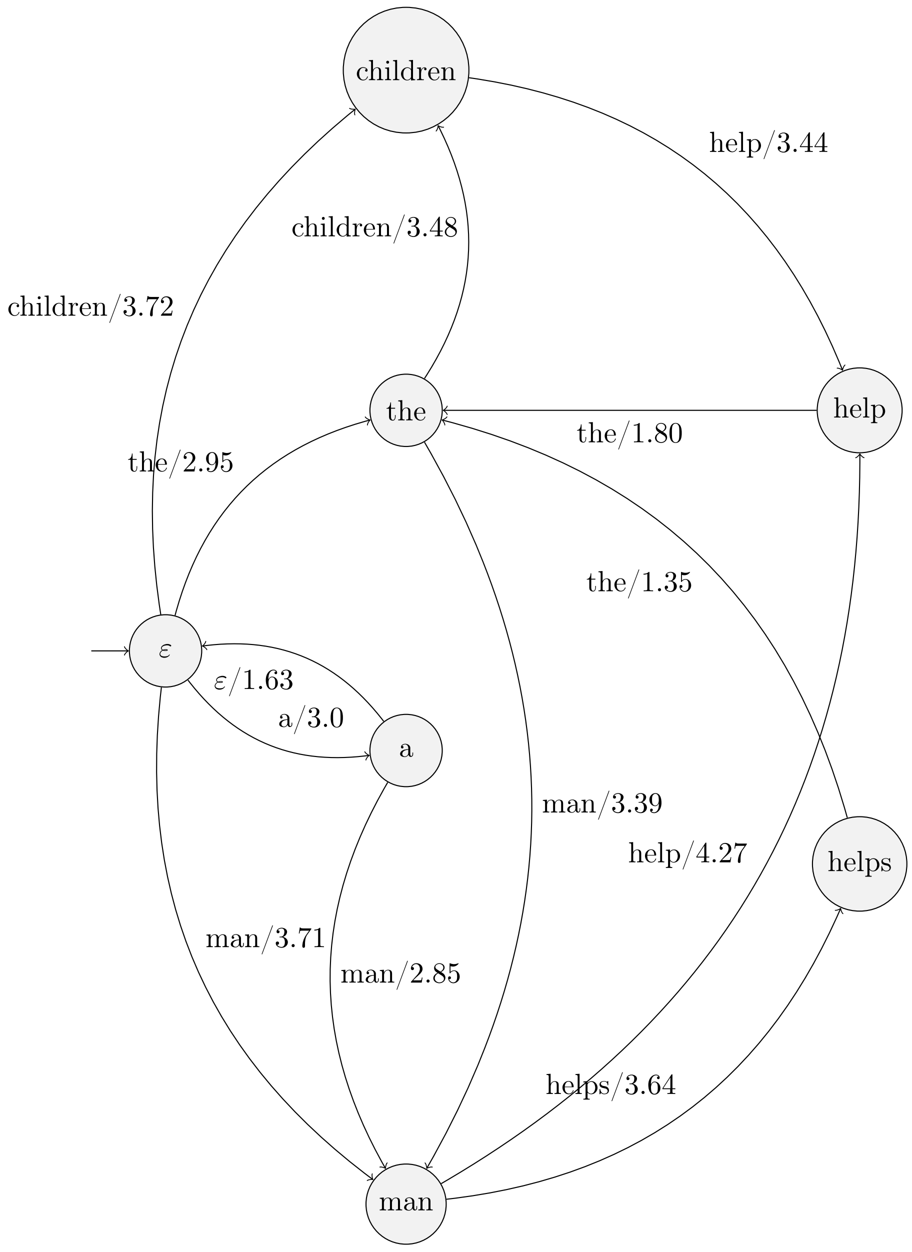


Figure 3.2: A bigram language model FSA.

3.3 Tree automata and tree grammars

In this section, we start by formally defining context-free grammars (CFGs). Then we define tree automata, finite-state machines for trees. Regular tree grammars, a closely related formalism, are also defined.

3.3.1 Context-free grammars

Definition 3.3.1 (Chomsky (1959)) *A* ***weighted context-free grammar*** *over a semiring W is a tuple $G = (N, \Delta, P, \pi, S)$, where*

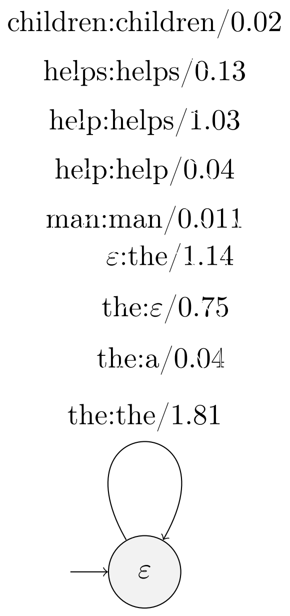


Figure 3.3: An error model FST.

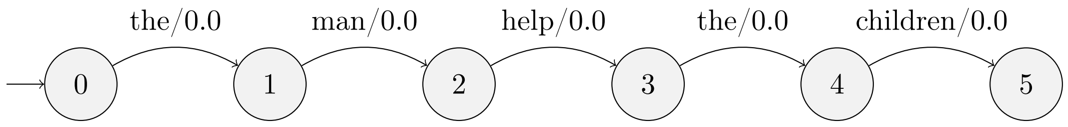


Figure 3.4: An embedded automaton.

- *N is an alphabet of nonterminals;*
- *Δ is a terminal alphabet;*
- *P is a finite set of productions, each of the form $A \to \gamma$, where $A \in N$ and $\gamma \in (\Delta \cup N)^*$;*
- *$\pi : P \to W$ is a weight function; and*
- *$S \in N$ is the initial nonterminal symbol.*

For every $\alpha, \beta \in (N \cup \Delta)^*$, a *derivation relation* $\alpha \Rightarrow_G \beta$ is defined if and only if there exist strings $\alpha_1, \alpha_2 \in (N \cup \Delta)^*$ and a production $p : A \to \gamma$ such that $\alpha = \alpha_1 \cdot A \cdot \alpha_2$ and $\beta = \alpha_1 \cdot \gamma \cdot \alpha_2$. The reflexive and transitive closure of $\Rightarrow_G$ is denoted by $\Rightarrow_G^*$. A derivation relation $\alpha \Rightarrow \beta$ obtained by applying production p is denoted by $\alpha \Rightarrow_G^p \beta$.

The sequence $d = (p_1, \ldots, p_m)$ is a derivation of the terminal string s if $S \Rightarrow_G^{p_1} t_1 \Rightarrow_G^{p_2} \ldots \Rightarrow_G^{p_m} s$, where $t_i \in (\Delta \cup N)^*$ for $1 \leq i < m$. A derivation is *left-most* if the left-most nonterminal in t_i is replaced at each derivation step. Unless specified otherwise, we assume that derivations are left-most. The string language generated by a CFG is the set of all strings that can be derived from the initial nonterminal. The weight of a derivation $d = (p_1, \ldots, p_m)$ is

$$wt(d) = \otimes_{i=1}^{m} \pi(p_i). \tag{3.3.1}$$

For a production $p \in P$, where $p : A \to \gamma$, the *left hand side* of p, A, is denoted by $LHS(p)$, and the *right hand side* of p, γ, by $RHS(p)$.

A CFG derivation can be represented as a tree, called a *derivation tree.* The initial nonterminal is the root of the tree. For each production in the derivation, the symbols on the right hand side are the children of the left hand side nonterminal. The terminal string of the derivation is the yield of the derivation tree.

A context-free grammar is in *Chomsky normal form* (CNF) if each production has the form $A \to B\ C$ or $A \to a$, where $A, B, C \in N$ and $a \in \Delta$. The derivation trees of CNF grammars are binary. Any CFG can be converted to a CNF grammar generating the same string language.

In a CFG that represents a constituency grammar, the terminals are words and the nonterminals are categories. The nonterminal S, indicating a simple clause or a sentence, is usually the initial nonterminal. CFG derivation trees are parse trees. Productions in which the right hand side contains terminals are called *lexical* productions, as they produce words.

Example 3.3.1 Let L be a CFG with nonterminal alphabet N = {S, VP, NP, DT, NN, NNS, VBP, VBZ} and terminal alphabet Δ = {*the, a, man, children, help, helps*}. Example productions of L are given below.

(a) S $\to$ NP VP

(b) NP $\to$ DT NN

(c) NP $\to$ DT NNS

(d) VP $\to$ VBP NP

(e) VP $\to$ VBZ NP

(f) DT $\to$ *the*

(g) NN $\to$ *man*

(h) NNS $\to$ *children*

(i) VBP $\to$ *help*

(j) VBZ $\to$ *helps*

3.3.2 Weighted tree automata

Definition 3.3.2 (Fülöp and Vogler (2009)) *A* ***weighted tree automaton*** *over a semiring W is a tuple $A = (Q, \Sigma, \mu, \nu)$, where*

- *Q is a finite set of states;*
- *Σ is a ranked alphabet of input symbols;*
- *$\mu = (\mu_k | k \in \mathbb{N})$ is a family of transition mappings $\mu_k : \Sigma_k \to W^{Q^k \times Q}$; and*
- *$\nu \in W^Q$ is the root weight vector.*

The weight of a transition from a tuple of states $p \in Q^k$ to a state $q \in Q$, that consumes the ranked symbol $\sigma \in \Sigma_k$, is given by $\mu_k(\sigma)_{p,q} \in W$. For the transition, the weight vector over all states $q \in Q$ is denoted by $\mu_k(\sigma)_p \in W^Q$.

Next we define the run semantics of the tree automaton A. A run r of A on tree $t \in T_\Sigma$ is a mapping $r : \text{pos}(t) \to Q$. Let $R_A(t)$ be the set of all runs of A on t. The run induced by $r \in R_A(t)$ at position p is the run r' on the subtree $t|_p$, defined for every $p' \in \text{pos}(r|_p)$ as $r'(p') = r(p.p')$. The weight of a run r on tree t is defined recursively as

$$rw(r) = \begin{cases} \mu_k(\sigma)_{(\varepsilon, r(\varepsilon))} & \text{if } t = \sigma, \\ rw(r|_1) \otimes \ldots \otimes rw(r|_k) \otimes \mu_k(\sigma)_{(r(1),\ldots,r(k)),r(\varepsilon)} & \text{if } t = \sigma(t_1, \ldots, t_k). \end{cases} \tag{3.3.2}$$

A *tree series* is a mapping $L : T_\Sigma \to W$ that assigns a weight in semiring W to trees over the ranked alphabet Σ. The weight that A associates with each tree t is given by the tree series $wt : T_\Sigma \to W$, where

$$wt(t) = \oplus_{r \in R_A(t)} rw(r) \otimes \nu_{r(\varepsilon)}. \tag{3.3.3}$$

A tree automaton over the Boolean semiring accepts trees that are assigned weight 1 and rejects trees that are assigned weight 0. The class of tree languages accepted by tree automata over the Boolean semiring is called the *regular* tree languages. The set of derivation trees of any context-free grammar is a regular tree language. Furthermore, for each regular tree language L, the set of yields of the trees in L is a context-free language.

3.3.3 Regular tree grammars

Weighted regular tree grammars (RTGs) are the generative counterpart of weighted tree automata.

Definition 3.3.3 (May (2010)) *A* ***weighted regular tree grammar*** *over a semiring W is a tuple $G = (N, \Sigma, P, \pi, S)$, where*

- *N is a finite set of nonterminals;*
- *Σ is a ranked alphabet of terminals;*
- *P is a set of productions, each of the form $A \to t$, where $A \in N$ and $t \in T_\Sigma(N)$;*

- $\pi : P \to W$ *is a weight function for the productions; and*

- $S \in N$ *is the initial nonterminal.*

The rewrite semantics for the grammar is defined as follows. The *derivation relation* $\Rightarrow_M$ is defined for every $s, s' \in T_\Sigma(N)$ by $s \Rightarrow_G s'$ if and only if there exists a context tree $c \in T_\Sigma(X \cup N)$ over $\Sigma \cup N$, a nonterminal $n \in N$ and a production $p \in P$, with $p : n \to t$ such that $s = c\langle x_1 \leftarrow n\rangle$ and $s' = c\langle x_1 \leftarrow t\rangle$.

A derivation relation $s \Rightarrow_G s'$ obtained by applying production $p \in P$ is denoted as $s \Rightarrow_G^p s'$. A derivation is called *left-most* if all the leaves in s to the left of the node where the substitution is made, are terminal symbols. Unless noted otherwise, we will assume that all derivations are left-most. The transitive and reflexive closure of $\Rightarrow_G$ is denoted by $\Rightarrow_G^*$.

The derivation sequence $d = (p_1, \dots, p_m)$, where $p_i \in R$ for all $1 \le i \le m$, is a derivation of the pair (a, b) if $a \Rightarrow_G^{p_1} t_1 \Rightarrow_G^{p_2} t_2 \dots t_{m-1} \Rightarrow_G^{p_m} b$, where $t_i \in T_\Sigma(N)$ for all $1 \le i < m$. For notational convenience, $d : a \Rightarrow_G^* b$ denotes any derivation sequence d of the pair (a, b), where $a, b \in T_\Sigma(N)$.

The weight of a derivation $d = (p_1, \dots, p_m)$ of a pair (a, b) is defined as

$$dw(d) = \pi(p_1) \otimes \dots \otimes \pi(p_m), \tag{3.3.4}$$

where $dw(d) \in W$.

It is possible that different (left-most) derivations will produce the same tree, just as different paths in a FSA may have the same string as label. The weight that G associates with each tree $t \in T_\Sigma$ is given by the tree series $wt_G : T_\Sigma \to W$, where

$$wt_G(t) = \oplus_{d:S \Rightarrow_G^* t} dw(d). \tag{3.3.5}$$

The tree series is defined if W is a commutative and complete semiring, corresponding to the similar condition for FSTs. RTG productions of the form $A \to B$, where $A, B \in N$, are called *chain productions.* They are comparable to ε-transitions in a FSA. The presence of chain productions in a RTG can lead to an infinite number of possible derivations. For any RTG G, a RTG without chain productions that recognizes the same tree series can be constructed.

An RTG G is in *normal form* if each of its productions has the form $A \to a(B_1 \dots B_k)$, for $k \in \mathbb{N}, a \in \Sigma_k$ and $A, B_1, \dots, B_k \in N$. For an arbitrary RTG, an equivalent normal form RTG can be constructed. For any (normal form) RTG, a weighted tree automaton that recognizes the same tree series can be constructed. A normal form RTG G is bottom-up *deterministic* if for each $k \in \mathbb{N}$, $a \in \Sigma_k$ and $B_1, \dots, B_k \in N$ there is at most one production of the form $A \to a(B_1 \dots B_k)$.

Example 3.3.2 Let L be a RTG which generates parse trees in $T_\Sigma(\Delta)$ for the vocabulary $\Delta = \{$*the, a, man, children, help, helps*$\}$ and the ranked category labels $\Sigma = \{$S, VP, NP, DT, NN, NNS, VBP, VBZ$\}$. Example productions of L are given in Figure 3.5. A nonterminal label denotes a category and the head POS tag of the phrase that is generated from that state. The non-terminals are used to model subject-verb agreement.

3.4 Weighted tree transducers

A weighted tree transducer is a finite-state machine that associates weights with input-output tree pair transformations. Tree transducers are defined with tree rewriting systems rather than with a transition function, as rewriting systems are better suited to represent recursive trees structures. Transducer states are considered to be a ranked alphabet where all symbols have rank 1. During tree rewriting, some nodes in the partial output tree are labeled as states.

Tree transducer variants are defined to process input trees bottom-up or top-down. In this thesis, we work with top-down tree transducers. We also define tree-to-string transducers, which take trees as input and produce strings as output.

3.4.1 Top-down tree transducers

Definition 3.4.1 (Maletti (2006)) *A **weighted extended top-down tree transducer** over a semiring W is a tuple $M = (Q, \Sigma, \Delta, R, \pi, S)$, where*

- *Q is a ranked alphabet of states, all of rank 1;*
- *Σ is a ranked alphabet of input symbols;*
- *Δ is a ranked alphabet of output symbols;*
- *R is a finite set of rules, each of the form $q(v) \to u$, where $q \in Q, v \in T_\Sigma(X_k)$ and $u \in T_\Delta(Q(X_k))$ for some $k \in \mathbb{N}$, such that v is linear and nondeleting in X_k;*
- *$\pi : R \to \mathcal{W}$ is a weight function on the rules; and*
- *$S \subseteq Q$ is a set of initial states.*

M is a weighted (non-extended) top-down tree transducer if each rule $r \in R$ has the form $q(\sigma(x_1 \dots x_k)) \to u$, where $k \in \mathbb{N}, \sigma \in \Sigma_k, q \in Q$ and $u \in T_\Delta(Q(X_k))$. A weighted top-down tree transducer is denoted by T, and a weighted extended top-down tree transducer by xT.

The rewrite semantics for an xT is defined as follows (Maletti, 2006). The *derivation relation* $\Rightarrow_M$ is defined for every $s, s' \in T_\Delta(Q(T_\Sigma))$ by $s \Rightarrow_M s'$ if and only if there exist trees $t_1, \dots, t_k \in T_\Sigma$ and $t \in Q(T_\Sigma)$, a context tree $c \in T_\Delta(X \cup Q(T_\Sigma))$ over $\Delta \cup Q \cup \Sigma$ and a rule $p \in R, p : l \to u$, such that

- $s = c\langle x_1 \leftarrow t\rangle$;
- $t = l\langle x_1 \leftarrow t_1, \dots, x_k \leftarrow t_k\rangle$; and
- $s' = c\langle x_1 \leftarrow (u\langle x_1 \leftarrow t_1, \dots, x_k \leftarrow t_k\rangle)\rangle$.

A derivation relation $s \Rightarrow_M s'$, obtained by applying a rule r, is denoted as $s \Rightarrow_M^r s'$. A derivation is called *left-most* if the state q in s where the rewrite is performed, is the left-most state is s. Unless stated otherwise, we will assume that all derivations are left-most. The transitive and reflexive closure of $\Rightarrow_M$ is denoted by $\Rightarrow_M^*$.

The derivation sequence $d = (r_1, \dots, r_m)$, where $r_i \in R$ for all $1 \leq i \leq m$, is a derivation of the pair (a, b) if $a \Rightarrow_M^{r_1} t_1 \Rightarrow_M^{r_2} t_2 \dots t_{m-1} \Rightarrow_M^{r_m} b$, where $t_i \in T_\Delta(Q(T_\Sigma))$ for all $1 \leq i < m$. For notational convenience, $d : a \Rightarrow_M^* b$ denotes any derivation sequence d of the pair (a, b), where $a \in T_\Sigma$ and $b \in T_\Delta$.

The weight of a derivation $d = (r_1, \dots, r_m)$ of a pair (a, b) is defined as

$$dw(d) = \pi(r_1) \otimes \dots \otimes \pi(r_m), \tag{3.4.1}$$

where $dw(d) \in W$.

A *weighted tree transformation* is a mapping $\tau : T_\Sigma \times T_\Delta \to W$, where Σ and Δ are ranked alphabets and W is a commutative and complete semiring. The weight that M associates with each pair $(s, t) \in T_\Sigma \times T_\Delta$ of input-output trees is given by the weighted tree transformation $wt_M(s, t) : T_\Sigma \times T_\Delta \to W$, defined as

$$wt_M(s, t) = \oplus_{d:q(s) \Rightarrow_M^* t, q \in S} dw(d). \tag{3.4.2}$$

M is *linear* if the right hand side of each rule is linear, and *nondeleting* if the right hand side of each rule in nondeleting with respect to X_k, where k is the number of variables on the rule left hand side. Linear and nondeleting tree transducers are denoted by L and N, respectively. For example, xLNT denotes an extended linear and nondeleting top-down tree transducer.

Suppose $q, p \in Q$, and $u \in T_\Delta(Q(X))$ and $v \in T_\Sigma(X)$ are context trees over $\Delta \cup Q$ and Σ, respectively. A transducer rule $r \in R$ is called *input-ε* if it has the form $q(x_1) \to u$, and *output-ε* if it has the form $q(v) \to p(x_1)$. A transducer is *input-ε-free* if it has no input-ε rules, and *output-ε-free* if it has no output-ε rules. A *chain* rule has the form $q(x_1) \to p(x_1)$.

Look-ahead restrictions can be added to the left hand side of transducer rules. We use one-symbol look-ahead that restricts the root node of the subtree matching a variable to a specific ranked symbol. For any transducer with one-symbol look-ahead, an equivalent transducer without look-ahead restrictions can be constructed by augmenting the state symbols.

For notational convenience, when tree transducer rules are represented visually, as in Figure 3.6, states and the nodes directly underneath them are merged; $q(\sigma)$ is denoted by $q.\sigma$, where q is a state and σ is a ranked symbol. One-symbol look-ahead of σ to variable $x_i \in X$ is denoted by $x_i{:}\sigma$.

Example 3.4.1 Let M be an xT that performs transformations between correct and incorrect sentences. Example xT rules for M are given in Figure 3.6. Rules (a) and (b) are equivalent rules used respectively in transducers without and with one-symbol look-ahead. For notational convenience we mostly use transducer rules with one-symbol look-ahead. Rule (d) is an input-ε rule. Rule (e) expresses the same transformation (inserting *the*) as (d), but is not an input-ε rule.

In the application of tree transducers in NLP, there is specific interest in the transformations in the class of input-ε-free xLNT with one-symbol look-ahead. Note that this is for the case where an output tree is given and the goal is to find input trees. When we perform grammar correction (or tasks such as machine translation) it is not necessary to copy subtrees. When deletion should be performed, it is defined for specific words or phrases, not for uninspected subtrees. As shown in Example 3.4.1, it makes more sense to define insertions without using input-ε rules. In addition to linguistic motivations, removing these restrictions on transducer rules may markedly increase the complexity of performing transducer inference.

3.4.2 Tree-to-string transducers

Several problems in NLP, including parsing and syntax-based machine translation, involve transformations between trees and strings. To express such transformations, extended top-down transducers are modified to produce strings rather than trees as output.

Definition 3.4.2 (May (2010)) *A* ***weighted extended top-down tree-to-string transducer*** *over a semiring W is a tuple $M = (Q, \Sigma, \Delta, R, \pi, S)$, where*

- *Q, Σ, π and S are defined as for weighted extended top-down tree transducers;*
- *Δ is an (unranked) alphabet of output symbols; and*
- *R is a finite set of rules, each of the form $q(u) \to v$, where $q \in Q, u \in T_\Sigma(X_k)$ and $v \in (\Delta \cup Q(X_k))^*$ for some $k \in \mathbb{N}$, such that v is linear and nondeleting in X_k.*

A (non-extended) tree-to-string transducer is defined by placing the same restriction on rule left hand sides than the corresponding definition for tree transducers. The classes of weighted and weighted extended top-down tree-to-string transducers are denoted by TS and xTS, respectively. Linear and nondeleting transducer classes, as well as input-ε and output-ε rules, are defined as for tree transducers. Tree-to-string transducer rules of the form $q(u) \to \varepsilon$, where $q \in Q, u \in T_\Sigma$, are also output-ε rules.

The rewrite semantics for an xTS is defined as follows. The *derivation relation* $\Rightarrow_M$ is defined on elements of $(\Delta \cup Q(T_\Sigma))^*$. These elements are sequences of terminal symbols and trees. The string concatenation and tree concatenation operations are extended to these sequences. For every $s, s' \in (\Delta \cup Q(T_\Sigma))^*$, the relation $s \Rightarrow_M s'$ is defined if and only if there exist trees $t_1, \ldots, t_k \in T_\Sigma$, $t \in Q(T_\Sigma)$, $\alpha, \beta \in (\Delta \cup Q(T_\Sigma))^*$ and a rule $r \in R$ with $r : l \to u$, such that

- $s = \alpha \cdot t \cdot \beta$;
- $t = l\langle x_1 \leftarrow t_1, \ldots, x_k \leftarrow t_k\rangle$; and
- $s' = \alpha \cdot u\langle x_1 \leftarrow t_1, \ldots, x_k \leftarrow t_k\rangle \cdot \beta$.

A derivation relation $s \Rightarrow^r_M s'$, the closure $\Rightarrow^*_M$, a derivation $d = (r_1, \ldots, r_m)$, the derivation weight $dw(d)$ and left-most derivations are all defined as for xT.

A *weighted tree-to-string transformation* is a mapping $\tau : T_\Sigma \times \Delta^* \to W$, where Σ and Δ are ranked alphabets and W is a commutative and complete semiring. The weight that M associates with each pair $(s, t) \in T_\Sigma \times \Delta^*$ is given by the weighted tree-to-string transformation $wt_M(s, t) : T_\Sigma \times \Delta \to W$, defined as

$$wt(s,t) = \oplus_{d:q(s) \Rightarrow^*_M t, q \in S} dw(d). \tag{3.4.3}$$

Example 3.4.2 Rules of a tree-to-string transducer that transforms correct trees to incorrect sentences are given in Figure 3.7. These rules correspond to the tree(-to-tree) transducer rules in Figure 3.6.

We briefly mention two related formalisms also used in NLP. Weighted synchronous context-free grammar (SCFG) (Chiang, 2007) is a formalism that expresses transformations similar to single-state NLTS models with one-symbol look-ahead. Weighted synchronous tree substitution grammars (STSGs) (Eisner, 2003) are closely related to single-state, one-symbol look-ahead xLNTs. Their non-synchronous counterparts, Tree Substitution Grammars (TSGs), are closely related to RTGs.

3.5 Natural language transformations

Transducers are used in NLP to perform linguistically motivated transformations. Sentences are represented as strings and constituency parse trees as trees. A FST performs transformations between two sentences. Tree transducers perform transformations between parse trees, and tree-to-string transducers between parse trees and sentences. For many applications, even when the result of a transformation is a parse tree, we are ultimately interested in the sentence yield of that tree. Knight (2007) identifies four properties that transducers should have to perform useful linguistic transformations:

- *Expressiveness*: Transducers should be expressive enough to capture complicated natural language transformations.
- *Inclusiveness*: More expressive formalisms should not lose the abilities of simpler transformations.
- *Modularity*: Complicated transformations should be decomposable into cascades of simple transformations.
- *Teachability*: It should be possible to infer linguistically plausible transformations for the transducers from observed data.

We now discuss the first two of these properties. Modularity refers to transducer composition, which is discussed under tree transducer inference in Section 4.6.2. Teachability refers to tree transducer training, presented in Section 4.6.3.

3.5.1 Expressiveness

There are several kinds of transformations that can be performed with tree transducer rules (DeNeefe *et al.*, 2007). We list some of these transformations below, and give examples in Figure 3.8. These transformations were originally identified for machine translation, but we discuss them in the context of grammatical error correction.

A rewrite rule that has a word either as a leaf of the left hand side tree or as a symbol on the right hand side is called a *lexical* rule. A rule is an *identity rewrite* if the yield of the left hand side tree is equal to the right hand side string (when ignoring the RHS states). In this section we are concerned with rules that perform non-identity lexical rewrites. The transformations are expressed as xTS rules.

- *Constituent phrase rewriting*
  A constituency phrase is rewritten directly as another phrase, without using any variables. The depth of the left hand side rule tree is greater than 2.
- *Non-constituent phrase rewriting*
  A (contiguous) phrase of words that is not a constituency phrase, is rewritten. The remaining sub-constituents of the constituent phrases that partly overlap with the phrase are represented by variables. This may be helpful for example in modelling collocation phrases, which are often not constituency phrases.
- *Non-contiguous phrase rewriting*
  Non-adjacent, linguistically related words are rewritten. For example, in subject-verb agreement it is possible that the head noun of the subject is not adjacent to the verb. Non-contiguous phrase rewriting can handle such constructions.
- *Context-sensitive word insertion or deletion*
  Performing word insertions and deletions in a syntactic context is useful, especially for the insertion and deletion of articles.
- *Constituent reordering*
  Phrase reordering is an important motivation for the use of tree transducers in statistical machine translation. An example of reordering in the context of grammar errors is the incorrect order of the noun and adjective phrases.

Most of these phrase rewrites require extended rule left hand sides; they cannot be preformed by non-extended tree transducers.

3.5.2 Inclusiveness

A tree transducer class is inclusive if it generalizes FSTs. To perform the class of rewrites that FSTs perform, input-ε and output-ε rules should be allowed in tree transducers or tree-to-string transducers. The rewrites that these rules perform correspond to FST input-ε and output-ε transitions.

However, in practice the use of these ε-rules are commonly restricted or disallowed as they may lead to unbounded insertions. Suppose that a string s is applied backwards to a tree-to-string transducer M, resulting in an application RTG A. The presence of output-ε rules may lead to *cycles* in M. If there are cycles in M, then the set of trees that A generates with non-zero weight (which is the decoding *search space*) may be infinite. However, it is possible to eliminate cycles by removing some of the output-ε rules; this will restrict the search space to be finite.

A FST model with input-ε or output-ε transitions with non-zero weights may also lead to an application FSA with cycles. However, FST implementations can handle decoding in an infinite search space more efficiently than tree transducer implementations.

3.6 Conclusion

This chapter introduced the automata theoretic models that we use in this thesis to model grammar correction. Section 3.1 provided some preliminary definitions.

Section 3.2 defined FSA and FST models. Weighted tree automata and regular tree grammars were defined in Section 3.3. Section 3.4 defined tree transducers and tree-to-string transducers. Finally, Section 3.5 discussed syntax-based natural language transformations and the properties that tree transducer models need to perform these transformations. In the next chapter we consider the application of these models to probabilistic modelling.

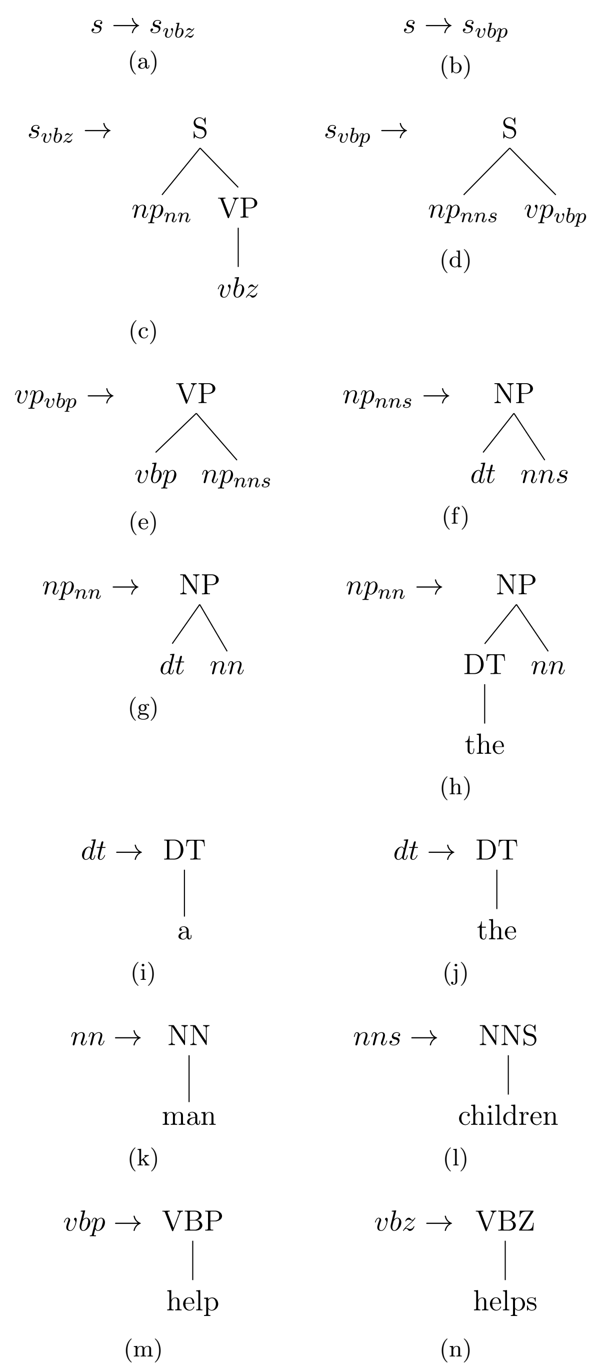


Figure 3.5: Regular tree grammar productions.

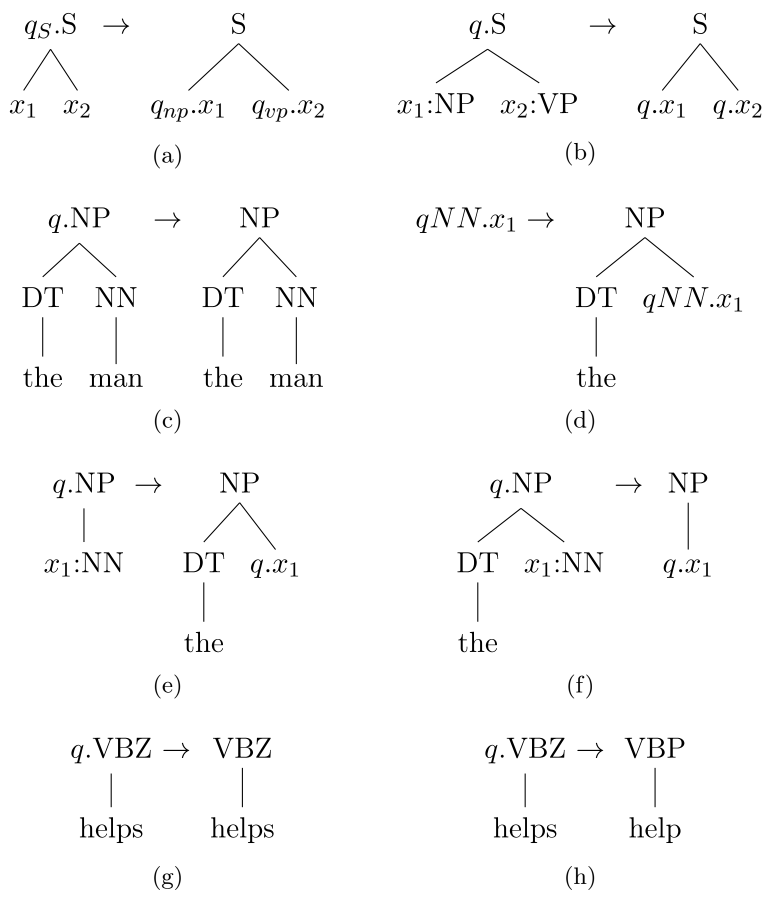


Figure 3.6: Extended tree transducer rules.

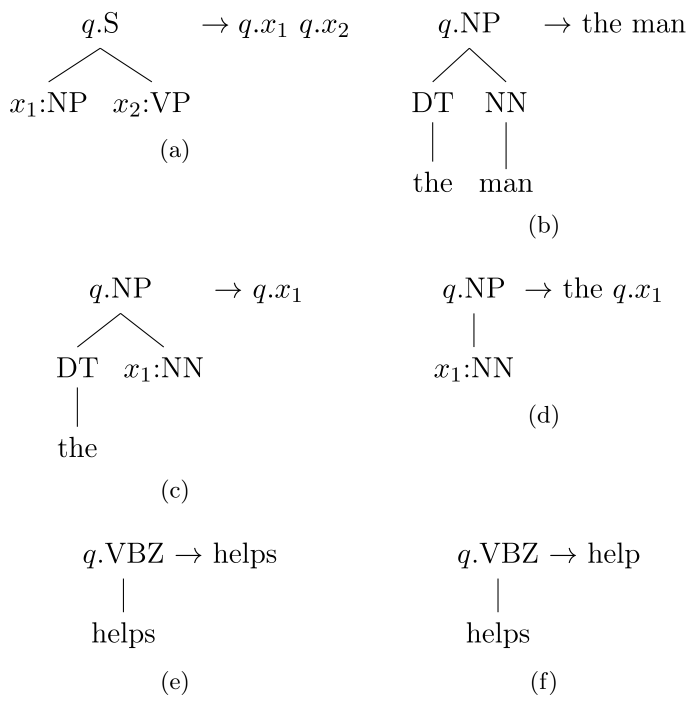
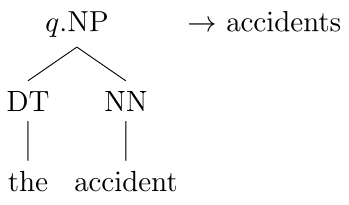
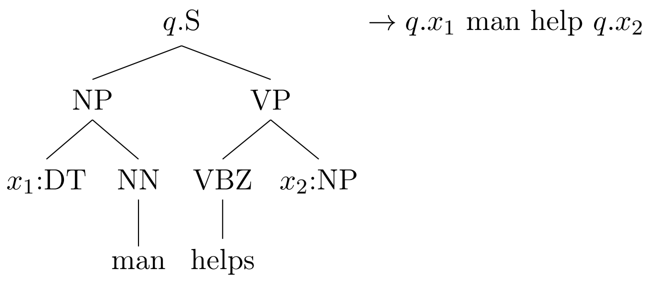


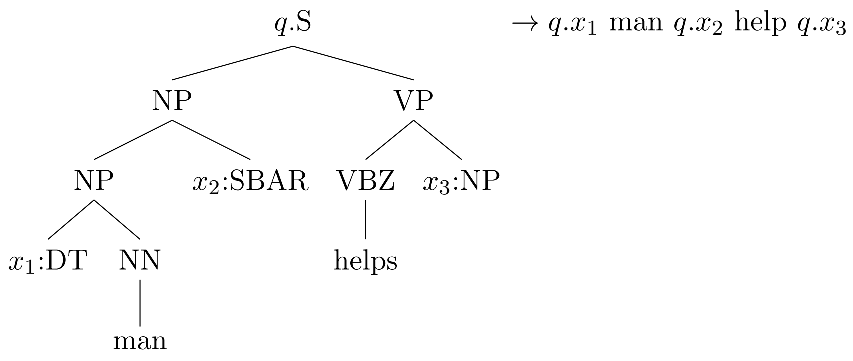
Figure 3.7: Extended tree-to-string transducer rules.



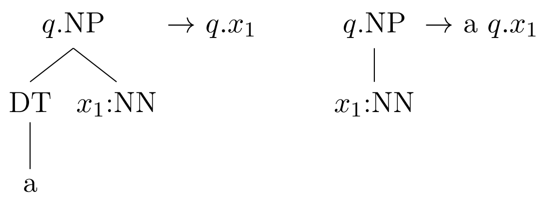
(a) Constituent phrase rewrite



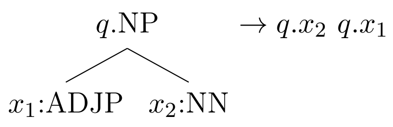
(b) Non-constituent phrase rewrite



(c) Non-contiguous phrase rewrite



(d) Context-sensitive word insertion and deletion



(e) Constituent reordering

Figure 3.8: Linguistically expressive xTS transformations.

Chapter 4

Probabilistic Models

In this chapter we present the probabilistic models used in this thesis. We start by introducing representation, inference and training in probabilistic models. Then we discuss n-gram language models and statistical syntactic parsing. In this thesis probabilistic modelling is performed mainly with weighted automata and transducers. We define probabilistic models for FSTs, RTGs, tree transducers and tree-to-string transducers. Unless stated otherwise, automata weights in this chapter are from the probability semiring. In practice, however, weights usually represent log probabilities, using the log or tropical semirings.

4.1 Probabilistic modelling

The goal of *supervised machine learning* is to learn a function $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ is the set of possible inputs and $\mathcal{Y}$ is the set of possible outputs, from a *training set.* A training set is a set of N input-output pairs $(x_1, y_1), \ldots, (x_N, y_N)$, where $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$.

The probabilistic approach to machine learning is to learn a probability distribution over a set of variables that represents that domain $\mathcal{X}$ and the range $\mathcal{Y}$ of f. The set of variables may also include *hidden variables* that represent neither the input nor the output of f.

In this section we discuss how probability distributions are represented, how they are used to predict the value of output variables given the value of input variables, and how their parameters are estimated from training data.

4.1.1 Representation

Suppose we have K ordered random variables $x_1, x_2, \ldots, x_K$. By the *chain rule* of probability the joint distribution over these variables can be factored as

$$P(x_1, \ldots, x_K) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \ldots P(x_K|x_1, \ldots x_{K-1}). \qquad (4.1.1)$$

Suppose that each of the variables can take M discrete values. Then the distribution $p(x_i|x_1, \ldots, x_{i-1})$ can be represented with a table of size M^i, of which $M^{i-1}(M-1)$ are free parameters. In total the probability model will have $O(M^K)$

parameters. For most real-world problems it is not feasible to estimate so many parameters from training data, or even to store them.

In order to represent such a joint distribution compactly, *conditional independence assumptions* should be made about the relation between variables in the distribution. Suppose x, y and z are random variables. Then x and y are conditionally independent given z, denoted by $x \perp y \mid z$, if and only if

$$P(x, y|z) = P(x|z)P(y|z). \tag{4.1.2}$$

A common independence assumption to make is the *Markov* assumption:

$$x_{i+1} \perp (x_1, \dots, x_{i-1}) \mid x_i \text{ for all } i \text{ with } 1 \leq i < n. \tag{4.1.3}$$

The probability distribution then factors as

$$P(x_1, \dots, x_n) = P(x_1) \prod_{i=2}^{n} P(x_i|x_{i-1}). \tag{4.1.4}$$

This model is called a *Markov chain.* A generalization of a Markov chain is an *m-th order Markov chain*, where each variable is conditioned on the m previous variables. In this model, the probability distribution factors as

$$P(x_1, \dots, x_n) = P(x_1) \dots P(x_m|x_1, \dots, x_{m-1}) \prod_{i=(m+1)}^{n} P(x_i|x_{i-1}, \dots, x_{i-m}). \tag{4.1.5}$$

In general, a probability distribution can be factored so that each term represents some *event* in the data, which is only conditioned on other events that influence it directly.

Probabilistic graphical models (PGMs) is a framework to represent probability distributions by making conditional independence assumptions (Murphy, 2012, chap. 10). The factorization of a probability distribution is represented as a graph, where nodes represent random variables and edges are used to encode conditional independence assumptions between variables. Within the framework of PGMs methods for the representation, inference and training of probabilistic models have been developed. We do not use PGMs explicitly in this thesis, but many of the models we use can also be expressed with them.

4.1.2 Inference

Probability distributions are used to perform probabilistic *inference.* Inference refers to the task of estimating unknown qualities from known qualities. Suppose that we have a joint distribution $P(\mathbf{x}, \mathbf{y}, \mathbf{z}|\theta)$, where $\mathbf{x}$ is a set of input variables, $\mathbf{y}$ a set of output variables, $\mathbf{z}$ a set of hidden variables, and θ the model parameters. For notational convenience θ is usually omitted where it is implied. The inference problem of finding the best output variables for a given set of input variables is referred to as *decoding.*

To perform inference we may either compute the joint distribution of the input and output variables

$$P(\mathbf{x}, \mathbf{y}) = \sum_{\mathbf{z}} P(\mathbf{x}, \mathbf{y}, \mathbf{z}) \tag{4.1.6}$$

or the conditional distribution of the output variables given the input variables

$$P(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{z}} P(\mathbf{y}, \mathbf{z}|\mathbf{x}), \tag{4.1.7}$$

where

$$P(\mathbf{y}, \mathbf{z}|\mathbf{x}) = \frac{P(\mathbf{x}, \mathbf{y}, \mathbf{z})}{P(\mathbf{x})}. \tag{4.1.8}$$

In both cases the hidden variables $\mathbf{z}$ are *marginalized* out of the distribution.

4.1.3 Training

Training or *learning* refers to the estimation of the parameters θ of a probability distribution, given some training data $\mathcal{D}$ of variable values. θ can be estimated as

$$\begin{aligned} \hat{\theta} &= \arg\max_{\theta} P(\theta|\mathcal{D}) && (4.1.9)\\ &= \arg\max_{\theta} P(\mathcal{D}|\theta)P(\theta). && (4.1.10) \end{aligned}$$

The term $P(\theta|\mathcal{D})$ is called the *posterior*, $P(\mathcal{D}|\theta)$ the *likelihood* and $P(\theta)$ the *prior* of the distribution. In this thesis we usually estimate model parameters with the *maximum likelihood estimate* (MLE)

$$\hat{\theta} = \arg\max_{\theta} P(\mathcal{D}|\theta), \tag{4.1.11}$$

which corresponds to assuming a uniform prior.

in which case a uniform prior is assumed.

Suppose that $\mathcal{D}$ consists of N training examples, each of the form $(\mathbf{x_i}, \mathbf{y_i})$. The goal of MLE training is find parameters θ that maximize the *observed data log likelihood*

$$\begin{aligned} l(\theta) &= \log P(\mathcal{D}|\theta) && (4.1.12)\\ &= \sum_{i=1}^{N} \log P(\mathbf{x_i}, \mathbf{y_i}|\theta) && (4.1.13)\\ &= \sum_{i=1}^{N} \log \left[\sum_{\mathbf{z_i}} P(\mathbf{x_i}, \mathbf{y_i}, \mathbf{z_i}|\theta) \right]. && (4.1.14) \end{aligned}$$

The *complete data log likelihood* is defined as

$$l_c(\theta) = \sum_{i=1}^{N} \log P(\mathbf{x_i}, \mathbf{y_i}, \mathbf{z_i}|\theta). \tag{4.1.15}$$

If there are no hidden variables, the training data is *complete.* For the family of exponential probability distributions, the likelihood function is concave if the training data is complete. The likelihood function also factorizes over the model parameters. In the presence of these two properties, the MLE of the model parameters can be computed simply by normalized frequency counts of events in the training data. For many of the models in this thesis, parameters can be estimated in this way.

A weakness of MLE is the 0 *count* problem. If an event does not occur in the training data, it will be assigned a probability of 0. Usually, we do not want to assign 0 probabilities to any event in the space of possible events. One solution to this problem is to use a (non-uniform) prior distribution. Another, related solution is *smoothing*, i.e. adjusting the frequency of events so that no event has a zero probability. A simple smoothing method is called *plus one* smoothing. Here, a count of 1 is added to the frequency of all events in the event space. Plus one smoothing corresponds to assuming a symmetric Dirichlet prior. Another smoothing technique that we use in this thesis, Good-Turing re-estimation, is described in Section 6.2.1.

In the case of hidden (or missing) variables, the likelihood no longer factorizes and is no longer concave (Murphy, 2012, chap. 11). In this case the goal of the training algorithm is to find a "good" local optimum for the likelihood function. One strategy is to use gradient descent or some other optimization method to maximize the likelihood function directly. Another strategy is to use the expectation maximization (EM) algorithm (Dempster *et al.*, 1977), an algorithm devised specifically to optimize likelihood functions. EM is a simple iterative algorithm that performs closed-form updates at each step, and has some theoretical guarantees about increasing the likelihood at each iteration. It converges towards a local optimum, which is sensitive to the initial parameters of the algorithm. One strategy to ensure that a good local optimum is found is to run the algorithm several times, using (different) randomly chosen initial parameter values. An alternative is to use an approximate maximum likelihood estimate, estimated from some approximation of the event space without latent variables, to initialize the parameters.

EM exploits the fact that if the data and hidden variables were fully observed the ML estimate would be easy to compute. Suppose that θ^0 is the initial parameter vector and θ^j is the parameter vector after iteration number j of the EM algorithm. The *expected complete data log likelihood* (Murphy, 2012, chap. 11) is defined as

$$Q(\theta, \theta^{j-1}) = \mathbb{E}[l_c(\theta)|\theta^{j-1}]. \tag{4.1.16}$$

The function computes the expected value of $l_c(\theta)$, for which some of the variables are hidden, given the parameter estimates θ^{j-1} of the previous iteration. The goal of the *E-step* is to compute the *expected sufficient statistics*, the terms inside $Q(\theta, \theta^{j-1})$ that the MLE depends on. Usually, these statistics are fractional counts of the events associated with the hidden variables, estimated by performing inference using the θ^{j-1} parameter estimate.

The goal of the *M-step* is to optimize the function Q with respect to θ:

$$\theta^j = \arg\max_{\theta} Q(\theta, \theta^{j-1}). \tag{4.1.17}$$

This is usually computed by normalizing the fractional counts computed in the E-step.

The EM algorithm is performed by iterating between the E-step and the M-step for a set number of iterations, or until the change in expected complete data log likelihood falls below a set threshold.

The theoretical basis for EM lies therein that the expected complete data log likelihood is a lower bound for the log likelihood, and that EM will monotonically increase the observed data log likelihood at each iteration until a local maximum is reached. A proof of this can be found in Murphy (2012, chap. 11).

4.2 Language models

Language models (LMs) assign probabilities to sentences. They are used widely in speech and language processing. A sentence is represented as a string $s \in \Delta^*$, where Δ is the *vocabulary* of the language (a finite set of words in the language). Let $s = w_1, w_2, \dots w_n$, where $w_i \in \Delta$ for $1 \leq i \leq n$. Then the language model is a probability distribution over Δ^* such that

$$P(s) = P(s_{\text{start}}, w_1, w_2, \dots, w_n, s_{\text{stop}}), \tag{4.2.1}$$

where s_{start} and s_{stop} are reserved words in the vocabulary denoting the beginning and end of the sentence, respectively.

An n-gram language model is an $(n-1)$th order Markov chain over words (an n-gram is a sequence of n consecutive words). For $n = 1, 2, 3$ such it is referred to as a unigram, bigram or trigram model, respectively. The most widely-used language models are trigram models. Usually a vocabulary is specified for the model, and all other words are mapped to a reserved word, typically denoted as `<unk>`.

Even though an n-gram model makes strong local independence assumptions, it still suffers from data sparseness: A trigram model has $O(v^3)$ parameters for a vocabulary of size v. For a typical 64000 word vocabulary there are far too many parameters to be estimated even from very large text corpora. Most of the possible trigrams will never appear in the text. The solution is to combine n-gram models of different orders. There are different ways to do this, usually in combination with smoothing. The state-of-the-art method is considered to be modified Kneser-Ney smoothing (Chen and Goodman, 1999).

4.3 Syntactic parsing

Parsing is the problem of mapping a sentence to its associated syntactic tree structure (Collins, 1999). In this thesis we are interested in performing constituency parsing. The main challenge of parsers is to resolve syntactic ambiguity. Syntactic ambiguity can lead to an exponential number of parses per sentence. Since we frequently need to work with sentences of more than 30 words in NLP (the average sentence length in a corpus that we used in this thesis is more than 20 words), this is a important issue.

In this section we discuss statistical parsing, parsing algorithms, and different parameterizations of probabilistic context-free grammar parsing models.

4.3.1 Statistical parsing

Early approaches to parsing were mainly rule-based, using large hand-crafted grammars that resolve ambiguity with selectional restrictions on words (Collins, 1999).

However, these approaches did not scale well from restricted domain text to wide-coverage domains such as newspaper articles. It was also observed that many structural preferences that help resolve ambiguity are better encoded as soft preferences than hard constraints.

Consequently, researchers began to investigate machine learning approaches to parsing. The machine learning problem is to induce a parsing function from a training set of sentence-tree pairs. A test set of sentence-tree pairs is used to evaluate the model's accuracy. Model performance is usually evaluated with precision, recall and F1 scores on phrase structure labels. Parsing is usually formulated as a probabilistic model. In this case the training problem is to estimate model parameter values. Statistical parsing models are trained with treebanks. The WSJ PTB is widely used to train English parsers.

Let Δ be an alphabet, Σ a ranked alphabet and W the probability semiring. Suppose $s \in \Delta^*$ and $t \in T_\Sigma(\Delta)$, such that s is the yield of t. A *statistical parsing model* is a mapping $\mathit{score} : T_\Sigma(\Delta) \times \Delta^* \to W$, where the domain is restricted to pairs (t, s), where s is the yield of t. The most likely tree is then defined as

$$T_{\text{best}}(s) = \arg\max_{t \in T_\Sigma(\Delta)} \mathit{score}(t, s). \tag{4.3.1}$$

The probability $\mathit{score}(t, s)$ can be either a joint probability $P(t, s)$ or a conditional probability $P(t|s)$. Usually a joint probability is used, as the model can then just assign a probability to the derivation of parse tree t. A *parser* is an algorithm to search for T_{best} for any input sentence s.

Note that, from the definition of conditional probability,

$$P(t|s) = \frac{P(t, s)}{P(s)}. \tag{4.3.2}$$

$P(s)$ is constant for a given sentence s. Therefore, in the parsing search problem,

$$T_{\text{best}}(s) = \arg\max_t P(t|s) = \arg\max_t P(t, s). \tag{4.3.3}$$

For the model to have a tractable number of parameters, t is decomposed into a number of *events* $e_1, e_2, \ldots, e_n$, that each have a weight $\mathit{score}(e_i)$. The model score is then a product of terms

$$\mathit{score}(t, s) = \prod_{i=1,\ldots,n} \mathit{score}(e_i). \tag{4.3.4}$$

An event is usually associated with a tree fragment. The choice of parameterization of the model is then the way that the tree is decomposed into fragments with which parameters are associated. As we discuss later in this section, events are commonly associated with probabilistic CFG derivations.

The first important criterion in the choice of parameterization is the discriminative power of the model. Events should include contextual information to help distinguish between good and bad parse trees. The second criterion is the compactness of the model. The resulting model should have as few parameters as possible, as the amount of training data necessary to estimate the parameters accurately is proportional to the number of model parameters. The trade-off between these two criteria is critical for good parsing models.

4.3.2 PCFGs

Early statistical approaches to parsing investigated the use of probabilistic context-free grammars (PCFGs). A PCFG is a CFG over the probability semiring. The probability that a PCFG assigns to a parse tree t is the weight of the derivation d of which t is the derivation tree.

CFG productions can be read directly from a parse tree, as the parse tree can be seen as a CFG derivation tree; from every node and its child or children a CFG production can be read off. Consequently, a treebank can implicitly constitute a CFG of a language. As a simple parsing model parameterization, let an event represent the application of a CFG production p with weight $P(p|LHS(p))$. The model score for a derivation $d = (p_1, \ldots, p_n)$ is then

$$P(d) = \prod_{i=1}^{n} P(p_i|LHS(p_i)). \tag{4.3.5}$$

Let $f(p)$ be the count of production p in the treebank derivations. The rule probabilities are computed by maximum likelihood estimation as

$$score(p) = P(p|\mathrm{LHS}(p)) = \frac{f(p)}{\Sigma_{p' \in P:\mathrm{LHS}(p')=\mathrm{LHS}(p)} f(p')}. \tag{4.3.6}$$

However, this parameterization does not result in a good parsing model. There are several reasons for this: Firstly, PTB trees tend to be flat, i.e. nodes often have many children. Therefore, a CFG read off directly from a treebank will have many low frequency productions with large right hand sides, for which model parameters cannot be estimated accurately. Secondly, the model is not sensitive to lexical information. All lexical productions have the form $A \to w$, where A is a POS tag and w is a word. The model assumes that, in a parse tree, the tree structure above A is conditionally independent of w given A. Thirdly, CFGs do not adequately model structural preferences in larger tree fragments.

4.3.3 Improved PCFGs

One parameterization developed to address the shortcomings of PCFGs, is *lexicalized* PCFGs. In a lexicalized PCFG derivation tree, each node is annotated with the head word and POS tag of the phrase under it. Instead of generating all the children of a node with one CFG production, the child nodes are generated one by one. The head child (the node with the same head word as the parent) is generated first. Then child nodes to the left of the head child are generated one by one, and then the nodes to the right of the head child. The probability of a node is conditioned on its parent and sibling nodes generated before it. We do not give further details, as we do not use lexicalized parsers in our models. The best performing lexicalized parsers include those of Collins (1999) and Charniak and Johnson (2005).

Another strategy to address the deficiencies of standard PCFGs is to split nonterminals in order to break down false independence assumptions. Nonterminals are annotated with additional information to improve the model parameterization. However, these grammars are fundamentally different from lexicalized grammars in that their annotations do not include open class words. A systematic approach to finding linguistically motivated splits is presented by Klein and Manning (2003). This approach is implemented in the widely-used Stanford parser.

A fundamental idea of this approach is the vertical and horizontal *Markovization* of productions. To perform vertical Markovization, a node is annotated with its v preceding ancestors. To perform horizontal Markovization, the right hand side of a production is decomposed in a similar manner to lexicalized grammars. The generation of a node is conditioned on the history of the h previously generated nodes (head node and left or right dependents). A default treebank PCFG has $v = 1$ and $h = \infty$, giving a parsing F1 score of 72.6%. Using $v = 3$ and $h \leq 2$, this can be improved to 79.7%.

We mention some of the linguistically motivated splits performed to further improve parsing accuracy. Part-of-speech nodes are subcategorized to capture linguistic features useful to parsing that are not adequately expressed by the default POS tagset. It is beneficial to mark nonterminals that have only one child, as such productions are otherwise often applied in incorrect contexts. Annotating a node with its head POS tag is also beneficial. Including these and some other improvements, Klein and Manning (2003) obtain a F1 score of 87.0%.

This manual grammar construction approach is improved further by automatically splitting and re-merging nonterminals to maximize the likelihood of a treebank. This approach is followed by Petrov *et al.* (2006), and is implemented in the Berkeley parser. This parser obtains a F1 score of 90%, which is on par with the performance of the lexicalized Charniak and Johnson (2005) parser, while having much fewer parameters. The method is able to automatically recover many of the linguistically motivated splits described above.

A challenge of this parsing model is that it is intractable to perform exact inference with the resulting grammar, as to do so the algorithm has to sum over all the possible subcategorizations (latent annotations) for each node. One approximation is to find the derivation with the highest probability. Petrov and Klein (2007) present more sophisticated inference algorithms.

4.3.4 Parsing algorithms

Two general parsing strategies are top-down and bottom-up parsing (Jurafsky and Martin, 2009, chap. 13). A top-down parser searches for valid parses by building the tree from the root downwards to the leaves. Whenever a leaf node is reached, parses that are not able to match the words in the sentence are discarded. A bottom-up parser starts with the input words as tree leaves and builds trees upwards by applying grammar rules. A parse is successful if the nonterminal that covers the whole input sentence is the initial nonterminal of the grammar.

Neither top-down nor bottom-up parsing algorithms are able to search efficiently in the presence of ambiguity. The solution to efficient parsing lies in using dynamic programming. Tables are used to store subtrees for each constituent as they are discovered. Consequently subtrees do not have to be re-parsed every time they are attached to different nodes. The most widely used dynamic programming parsing algorithms are the Cocke-Kasami-Younger (CKY) algorithm and the Earley algorithm. Both these algorithms can parse sentences in $O(Gn^3)$ time, where n is the sentence length and G is a constant representing the size of the CFG (the number of rules, terminals and nonterminals).

The CKY algorithm performs a bottom-up search using dynamic programming. CKY parsing requires the CFG to be in CNF in order to parse in cubic time. The algorithm can be modified to also handle rules of the form $A \to B$, where A and B are nonterminals, without additional complexity, but crucially no rule RHS may have more than 2 nonterminals. CKY parsing for a sentence of length n is usually represented by the upper-triangular part of a $(n+1) \times (n+1)$ table. Each (i, j) cell contains a set of nonterminals that spans all the terminals between positions i and j in the input. As each nonterminal in the grammar has exactly two children, its span can be split into two, divided at position k. The matching first child subconstituent (i, k) will be in a cell to the left of (i, j) and the second child subconstituent (k, j) will be in a cell below (i, j). Parsing is performed by filling in the table in a bottom-up fashion. The cell $(0, n)$ represents the entire input. The insight here that makes dynamic programming possible is that for each nonterminal in a cell we only need to keep the highest-scoring entry, as any other rule at that point is guaranteed to be suboptimal. The highest scoring parse can be found from the initial nonterminal entry in the $(0, n)$ cell by following pointers between the cells. The CKY algorithm is the most common parsing algorithm used by statistical parsers.

The Earley algorithm performs a top-down search using dynamic programming. A single left-to-right pass fills a chart with $(n + 1)$ entries. For each terminal position, the chart contains a list of partial parse trees that have been generated so far. Another parsing strategy, that generalizes both CKY and Earley parsing, is *chart parsing*. Further details can be found in Jurafsky and Martin (2009, chap. 13).

4.4 Finite-state transducers

As probabilistic models, FSTs are closely related to (discrete) HMMs, and inherit some HMM algorithms.

4.4.1 Representation

A FSA is used here to represent a probability distribution over strings. A FST represents either a joint distribution over input and output strings, or a conditional distribution over the output strings given the input strings. In both cases, the standard way to parameterize the distribution is by letting transition weights represent *locally normalized* conditional probabilities.

In the joint case, a transition weight of a FST represents the conditional probability

$$\mu(a_i, b_i)_{q_{i-1}, q_i} = P(a_i, b_i, q_i | q_{i-1}), \tag{4.4.1}$$

while in the case of a conditional distribution

$$\mu(a_i, b_i)_{q_{i-1}, q_i} = P(b_i, q_i | a_i, q_{i-1}). \tag{4.4.2}$$

Similarly, for a FSA,

$$\mu(a_i)_{q_{i-1}, q_i} = P(a_i, q_i | q_{i-1}). \tag{4.4.3}$$

In all of these cases, the initial weight function λ is defined as $\lambda(q) = P(q)$, and the final weight function ρ as $\rho(q) = 1$, for all states $q \in Q$.

4.4.2 Inference

The weights of all paths through a FST can be computed efficiently using an instance of the *forward-backward* dynamic programming algorithm (Rabiner and Juang, 1986). At each state q, the *forward* weight is the sum over all paths that lead to q (excluding the final weight), and the *backward* weight is the sum over all partial paths from q (excluding the initial weight). The forward and backward weights can be computed efficiently using dynamic programming. The product of the forward and backward weights at state q is equal to the sum of weights over all the paths that pass through q. A slightly more general problem is finding the sum over all paths between any two states. This is computed using an instance of the Floyd-Warshall algorithm (Mohri, 2009).

Application refers to the process of transforming some input by a transducer (May, 2010, chap. 4). Inference in transducers is usually done through application. The application can be either *forward*, when we are given an input string (or more generally, an input FSA) and we want to know how the transducer transforms it, or *backward*, when an output string is given and we want to find input strings that cause the transducer to produce this output. The result of application is usually represented as a FSA.

Concretely, the strategy to perform forward application of a string s to a FST T is as follows: Firstly, s is embedded in a FSA I. Then the composition $C = I \odot T$ is computed. Finally the right projection of C, that represent the result of the application, is found. A similar procedure is followed for backward application. This method of application is referred to as *embed-compose-project*.

Application can be extended to a cascade of transducers. There are several strategies to do this (May, 2010, chap 4.). We briefly describe the *bucket brigade* method. Suppose that I is the embedded input FSA, to be applied forwardly to the cascade $T_1, T_2, \ldots, T_n$. I is firstly composed with T_1: $C_1 = I \odot T_1$. The compositions of the rest of the transducers are then computed iteratively as

$$C_i = C_{i-1} \odot T_i, \text{ for } i = 2, \ldots, n.$$

Finally, the right projection of C_n gives the result of the application.

Suppose that FSA A is the result of some application. The ultimate goal of inference is usually to compute the highest-probability output string or the k-best output strings. Suppose that the weights of A are negative log probabilities, and we want to find the lowest-scoring path (i.e., the path with the highest probability). Then we can apply the forward-backward algorithm over the tropical semiring. The FSA weight computed by the forward-backward algorithm will be the weight of the lowest-scoring path, known as the *Viterbi* path. If backpointers are used by the forward-backward algorithm, this path can be found easily. In general there may be multiple paths through A with the same label. One way to overcome this problem is to firstly *determinize* the transducer (Mohri, 2009). We do not perform determinization in this thesis, so we do not discuss it further here. In this thesis we use the label of the highest-probability path as an approximation of the highest-probability

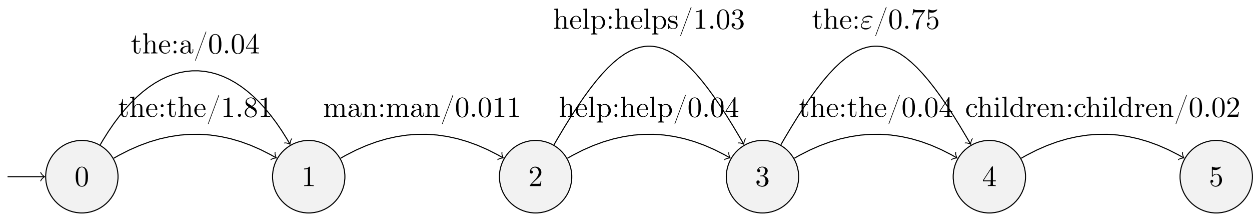


Figure 4.1: Composition of the embedded FSA and the error model FST.

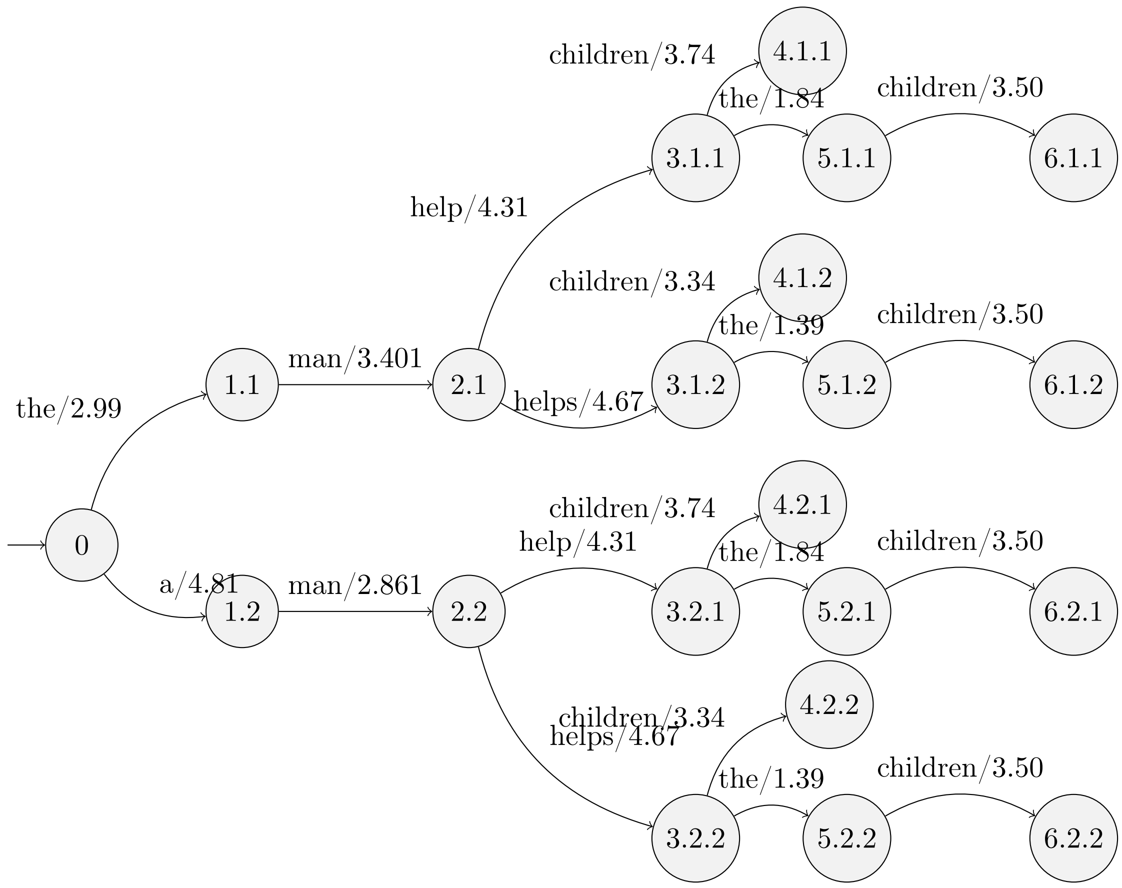


Figure 4.2: Application FSA after projection and language model composition.

string. An alternative is to compute the weights of the k highest probability paths, add the weights of paths with the same label, and find the highest-probability string from that list.

Example 4.4.1 Let E be the embedded automaton from Figure 3.4, R the error FST from Figure 3.3 and L the language model transducers from Figure 3.2. Let i be an incorrect sentence and c a correct sentence. R represents the probability distribution $P(i|c)$ and L the distribution $P(c)$. The cascade of these two transducers represents a noisy channel error correction model. Suppose that inference is performed with bucket brigade application. The composition $T_1 = E \odot R$ is given in Figure 4.1. Then $T_2 = T_1 \odot L$ is computed, and T_2 is right projected to FSA

A. A is given in Figure 4.2. The Viterbi path through A is then computed. The weight of this path is 14.401 and the label is *the man helps children*. Therefore the subject-verb agreement error is corrected, and there is an unnecessary article deletion. Other possible outputs include *the man helps the children*, with weight 15.951, and the original sentence with weight 16.041.

4.4.3 Training

Suppose that we are given a FST M and a training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ of string pairs. Let us first consider the case where for each possible input-output pair there is only one path through the transducer. Then the training data is complete, and the maximum likelihood estimates can be computed directly. Suppose that $f(a, b, p, q)$ is the number of times that the sequence $p, (a, b), q$ occurs in the paths of the training examples. For a joint probability distribution the transition weight $\mu(a, b)_{p,q}$ is estimated as

$$P(a, b, q|p) = \frac{f(a, b, p, q)}{\Sigma_{p'} f(a, b, p', q)}. \tag{4.4.4}$$

The transducer can also represent the conditional probability of the input given the output (we parameterize our error model FSTs in this way). In this case, $\mu(a, b)_{p,q}$ is estimated as

$$P(a, q|b, p) = \frac{f(a, b, p, q)}{\Sigma_{b',p'} f(a, b', p', q)}. \tag{4.4.5}$$

In the general case, where multiple paths have the same labels, the EM algorithm can be used to estimate the weights of M. For each pair (x_i, y_i), x_i and y_i are embedded in FSA I_i and O_i, respectively. The composite transducer $D_i = I_i \odot M \odot O_i$ is a *derivation* FST for the string pair. Each path in D_i represents a derivation of (a_i, b_i) in M. The forward-backward algorithm is then run on D_i to compute fractional counts of each transition. This is the E-step of the algorithm. Multiple transitions in D_i may correspond to the same transition in M. The fractional counts of transition weights are summed over all training examples. The counts are then normalized as transitions of M, which is the M-step. This process is iterated, the weights of M being updated after each iteration.

FST inference and training algorithms are implemented in OpenFST (Allauzen *et al.*, 2007) and Carmel[1]. OpenFST provides a more extensive set of inference algorithms, but EM training is implemented only in Carmel.

Eisner (2002) presents a training algorithm for a log-linear FST parameterization with *global normalization*. Chiang *et al.* (2010) propose Bayesian inference algorithms for FST models. For some tasks these algorithms give better model parameterizations. However, MLE and the EM algorithm remain good and efficient ways to train FSTs.

4.5 Regular tree grammars

A probabilistic RTG represents a probability distribution over trees. The weight of a production $r : A \to u$ is the conditional probability $P(u|A)$.

[1] `http://www.isi.edu/licensed-sw/carmel`

As there may be multiple paths in a FST with the same label, there may be multiple *derivations* in a RTG for the same tree. The sum of the weights of tree derivations can be computed efficiently using the inside-outside algorithm, a generalization of the forward-backward FST algorithm (Graehl *et al.*, 2008).

Given a RTG $G = (\Sigma, N, P, \pi, S)$ over a semiring W, let $n, n', m \in N$ and $u \in T_\Sigma(N)$, $p \in P$. The *inside weight* β_G for a nonterminal $n \in N$ is the sum of the weights of all trees that can be derived from it, defined as

$$\beta_G(n) = \oplus_{p:n\to u}\pi(p) \otimes_{l\in leafpos(u)} \beta_G(label(l)). \tag{4.5.1}$$

The inside weights are also defined for $u \in T_\Sigma(N)$, the right hand side of a production $p : n \to u, p \in P$, as

$$\beta_G(u) = \begin{cases} \bar{1} & \text{if } leafpos(u) = \emptyset, \\ \otimes_{l\in leafpos(u)}\beta_G(label(l)) & \text{otherwise.} \end{cases} \tag{4.5.2}$$

β_G can be evaluated over any complete semiring. In practice, RTGs are usually formulated without chain rules to avoid possible difficulties with computing infinite summations.

The *outside weight* α_G of a nonterminal $n \in N$ is the sum of all derivations using n, excluding the weight of the subtree under n, defined as

$$\alpha_G(n) = \begin{cases} 1 & \text{if } n = S, \\ \oplus_{p:m\to u, l\in leafpos(u)}\pi(p) \otimes \alpha_G(m) \otimes_{l'\in leafpos(u)-\{l\}} \beta_G(label(l')) & \text{otherwise.} \end{cases} \tag{4.5.3}$$

The sum of the weights of all trees generated by G is given by $\beta_G(S)$. Given the inside and outside weights, the sum of the weights of all trees using a particular production $p : n \to u$ is

$$\gamma_G(p) = \alpha_G(n) \otimes \pi(p) \otimes \beta_G(u). \tag{4.5.4}$$

We do not discuss the training of RTGs, as we do not train RTGs directly in this thesis. However, in the next section we show how RTGs and the inside-outside algorithm are used to performing EM training of tree transducers. When we perform inference with tree transducers, the inside-outside algorithm is used to find the highest-scoring derivations in a RTG.

4.6 Tree transducers

We now discuss how weighted tree transducers and weighted tree-to-string transducers are used as probabilistic models.

4.6.1 Representation

As in the case of probabilistic FST models, tree and tree-to-string transducers can represent joint or conditional probability distributions. Suppose t is an input tree, s is an output tree or string, and r is a tree transducer rule of the form $q(\sigma(t_1, \ldots, t_k)) \to u$. Then the *root* of r is defined as $root(r) = q(\sigma)$ if one-symbol look-ahead is used, and $root(r) = q$ otherwise. Suppose that a tree transducer represents a joint distribution $P(t, s)$. Then the weight of r is

$$\pi(r) = P(r|root(r)). \tag{4.6.1}$$

In the case of a conditional distribution $P(s|t)$, the weight of r is

$$\pi(r) = P(r|LHS(r)), \tag{4.6.2}$$

where $LHS(r)$ denotes the entire left hand side of r.

4.6.2 Inference

Inference with tree transducers is performed with application, as in the case of FSTs. However, tree transducer application is usually performed with custom algorithms for specific transducer classes. The generic approach used for FSTs cannot be used for most tree transducer classes, due to a lack of closure under composition. Application can only be performed with a limited number of transducer classes. The class of LNTs is closed under composition, but the class of xLNTs is not (Maletti *et al.*, 2009). Almost none of the classes that perform copying or deletion are closed under composition.

A RTG $G = (\Sigma, N, P, \pi, S)$ over a semiring W G is *embedded* in a LNT E if E assigns the weight $wt_G(t)$ to the tree pair (t, t) for every tree $t \in T_\Sigma$, and 0 to all other tree pairs. May (2010, chap. 4) gives a custom forward application algorithm for applying a RTG to a xLNT. Though the class of xLNTs is not closed under composition, embedded RTGs form a more restricted class, and the application RTG can be computed. For backward application of a RTG to a xLNT, the embed-compose-project approach can be used. If M is a xLNT and E is a embedded LNT, then the composition $C = M \odot E$ can be computed. C is then right projected to a RTG A. A modified bucket brigade approach for application to tree transducer cascades has also been proposed (May, 2010, chap. 4).

Tree-to-string transducers are typically applied with backward application of a string s to a xLNTS M. Transforming a string to a tree can be seen as a parsing problem, and therefore parsing algorithms are used to perform this application. An algorithm based on Earley parsing is proposed by May (2010, chap. 4). A CKY-based algorithm for this application has also been used (Galley *et al.*, 2006).

When we work with tree transducers in NLP applications, exact inference is usually intractable, since the search space is too large to perform an exact search given time and memory constraints. Therefore the search space is usually pruned heuristically. We discuss this more concretely in Section 6.3.2.

4.6.3 Training

We now discuss the training problem for probabilistic extended tree and tree-to-string transducers, given a set of training pairs. We assume here that the transducer rules are given, and that the goal is to estimate the rule weights. We discuss methods to extract rules from the training data in Section 6.1.

Suppose we have a xLNT $M = (\Sigma, \Delta, Q, R, \pi, Q_d)$ with $Q_d = \{S\}$ over the probability semiring, and a set $\{(t_1, s_1), \ldots, (t_N, s_N)\}$ of tree training pairs. Suppose

t is an input tree and s is an output tree. Let us first consider the case where only one derivation from each training pair is considered as training data. Let $f(r)$ denote the number of occurrences of rule r in the training derivations.

For a joint distribution $P(t, s)$ the MLE of the rule weight associated with r is

$$\pi(r) = P(r|root(r)) = \frac{f(r)}{\Sigma_{r':\mathrm{root}(r')=\mathrm{root}(r)} f(r')}. \tag{4.6.3}$$

Let us now consider the general case, where the EM algorithm is used to perform training. Every derivation of M can be represented with a *derivation tree* over R (the tree nodes are labeled with rules). For a tree pair (t_i, s_i), the set of derivation trees for M, and the associated derivation weights, can be generated by a *derivation* RTG G_i. Each production in G_i has the form $q \to r(q_1, \ldots, q_k)$, where symbol r denotes a rule $r \in R$ with k variables, and each state label $q, q_1, \ldots, q_k$ has the form $p \times pos(t_i) \times pos(s_i)$, where $p \in Q$. The start state S of G_i corresponds to the start state of M.

An algorithm to construct a derivation grammar G_i is given by Graehl *et al.* (2008). In this algorithm, in the worst case each of the rules in R has to be considered for each of the tuples $(q, p_t, p_s) \in Q, pos(t_i), pos(s_i)$. The time and space complexity are then both $O(|Q| \cdot |t_i| \cdot |s_i| \cdot |R|)$ or $O(Kn^2)$, where n is the total size of the input and output trees, and K is the grammar constant, representing the size of the rules and the states in M.

If the transducer has chain rules a corresponding derivation RTG may have cycles, leading to an infinite number of derivation trees. To avoid this additional complexity, chain rules are removed before the derivation grammars are constructed.

Similarly, derivation trees can also be constructed when M is a xLNTS and $\{(t_1, s_1), \ldots, (t_N, s_N)\}$ is a set of tree-string training pairs. However, computing xLNTS derivations is more complex than computing xLNT derivations. Input tree nodes are matched with arbitrary output string spans, instead of output subtrees. Suppose that no transducer rules have more than two variables. For an output string of length m there are $O(m^2)$ spans, and each binary production over a span has $O(m)$ ways to divide the span in two. These spans and span divisions should be considered for each of the n input tree nodes, the transducer rules and the different states. Let K again be the grammar constant. Then the time and space complexity of constructing a xLNTS derivation grammar is $O(Gnm^3)$.

An instance of the EM algorithm has been defined to train tree transducers (Graehl *et al.*, 2008). The model parameters θ are represented by the rule weight function $\pi : R \to W$. For each pair in the training data $\mathcal{D}$ a derivation grammar is constructed only once. For each iteration j of the EM algorithm the weights of the derivation grammars are updated to the current parameter values θ^j. A derivation grammar production with right hand side tree root labeled r is assigned weight $\pi(r)$.

The expected complete data log likelihood is

$$\begin{aligned} Q(\theta, \theta^{j-1}) =& \mathbb{E}[\log P(\mathcal{D}|\theta)|\theta^{j-1}] && (4.6.4)\\ =& \Sigma_{i=1}^{N} \Sigma_{r \in R} \overline{f}_{G_i}(r) \log \pi(r) && (4.6.5)\\ =& \Sigma_{r \in R} \log \pi(r) \Sigma_{i=1}^{N} \overline{f}_{G_i}(r), && (4.6.6) \end{aligned}$$

where $\overline{f}_g(r)$ is the expected number of times that rule r is used in derivation trees generated by derivation RTG g, when g is parameterized by θ^{j-1}. To estimate $\overline{f}_g(r)$, the number of times that r is used in each derivation tree is weighted by the derivation tree weight, and the sum of these weights is normalized over the weights of all the derivation trees. Therefore, we have

$$\overline{f}_g(r) = \frac{\sum_{d \in T_R} n_d(r) wt_g(d)}{\sum_{d \in T_R} wt_g(d)}, \tag{4.6.7}$$

where $n_d(r)$ is the number of times that r occurs in derivation tree d. We can compute this efficiently with the inside-outside weights of g:

$$\overline{f}_g(r) = \frac{\sum_{p:a \to u, \mathrm{root}(u)=r} \gamma_g(p)}{\beta_g(S)}. \tag{4.6.8}$$

In the E-step of the EM algorithm, the expected fractional counts are computed for each derivation grammar. The expected count of each rule r is then obtained by summing over all the training examples:

$$\overline{f}(r) = \sum_{i=1}^{N} \overline{f}_{G_i}(r). \tag{4.6.9}$$

In the M-step the MLE of the rule weights are updated, for joint probability distributions, using equation (4.6.3).

Tiburon (May and Knight, 2006) implements tree transducer algorithms, including EM training. We use Tiburon in performing experiments with our tree transducer models.

Recently, Bayesian methods for tree transducer training have also been developed (Jones *et al.*, 2012). Another alternative is a training method based on large margin training for structural SVMs (Cohn and Lapata, 2007).

4.7 Conclusion

In this chapter methods and applications of probabilistic models were presented. Section 4.1 introduced probabilistic modelling. Language models were presented in Section 4.2, and parsing in Section 4.3. We discussed probabilistic modelling with FSTs in Section 4.4. Section 4.5 discussed probabilistic RTGs, and specifically the computation of inside-outside weights. Finally, probabilistic modelling with tree transducers, including the EM training algorithm, was discussed in Section 4.6.

Chapter 5

Experimental Setup

In this chapter we present the experimental setup used in developing and testing our probabilistic tree transducer models for grammatical error correction. We discuss the various steps in preprocessing and parsing the learner corpora training data. We make use of standard, publicly available NLP tools to perform many of the processing steps. The processing pipeline and the steps that we perform ourselves are implemented in Python. We briefly discuss additional training resources, as well as the n-gram language model used. Finally we present a baseline FST model which include some of the model components used in the transducer models in Chapter 6.

5.1 Training and testing data

The two corpora used to train and test our models are NUCLE and FCE, described in Section 2.2.1. Details of the formats of these two corpora are given in Appendix B. The preprocessing steps we describe in the following sections are implemented separately for the data formats of the two corpora, though the same steps are followed.

Both corpora have separate training and test sets. We divide the training sets of each corpus into 80% *training data*, 10% *validation data* and 10% *development data*. Splitting is performed by random selection at essay level. The training sets are used to train the transducer error correction models, while hyperparameters such as the weight of the language model are tuned on the validation sets to optimize the system performance on the evaluation metric used directly. The development set is used to compare the performance of different modelling choices, while the test sets are used to perform the final evaluation of our models.

For the NUCLE test data there are two sets of annotations. The first is the original version annotated by the official annotator. The second is a revised version, released after the CoNLL-2013 shared task. After the initial results for the shared task were released, participating teams were given the opportunity to suggest alternative correction annotations, based on the output of their systems. These alternative answers were then judged by the official annotator, and the revised version, that allow multiple possible corrections, was released. The idea is that the evaluation scores on the revised annotations are more accurate, as some of the corrections suggested by a system may be correct, although not originally suggested by the annotator.

5.2 Preprocessing

The learner corpora data consist of essays, which are subdivided into paragraphs. For each paragraph the original text is given, as well as error annotations. An error annotation usually consists of the position of the word or phrase in the original text that should be replaced, the type of error, and the suggested correction. We extract the following from a corpus, for each sentence in the original text:

- The tokenized original (possibly incorrect) sentence. We refer to this as the *incorrect sentence.*
- The tokenized *correct sentence.*
- A *word alignment* between the tokens of the correct and incorrect sentences.

5.2.1 Tokenization

An important preprocessing step is *tokenization.* Tokenization is the process of segmenting running text into words and sentences (Jurafsky and Martin, 2009, chap. 3). The tokenization is performed with NLTK (Bird *et al.*, 2009), following the conventions used in the CoNLL-2013 shared task.

The first tokenization step is to split the paragraphs into sentences. Sentence splitting is based on sentence-terminating punctuation (".", "?" and "!"). However, in some cases periods are ambiguous, as they are also used in abbreviations, for example "Ms." or "ex.". Tokenizers use heuristics or machine learning algorithms to classify sentence boundaries, using surrounding words and punctuation as features. In our implementation sentence splitting is performed with NLTK *punkt.* This tokenizer uses heuristics that, though not error-free, give good tokenizations.

Note that it is possible that due to punctuation errors there may not be an exact correspondence between the correct and incorrect sentences. In such cases we follow the sentence alignment of the incorrect sentence. Therefore it is possible that in some cases the corresponding correct "sentence" will consist of more than one sentence, or may not be a complete sentence.

Next, word tokenization is performed on each sentence. The primary goal of word tokenization (in English) is to separate punctuation from words. There are slight differences between tokenization conventions of different tokenizers. For example, in handling apostrophes, *shouldn't* can be tokenized as either *should n't* or *shouldn 't*. During tokenization, quotation marks may also be normalized (there are different textual representations for open and closing quotation marks). As the phrase structure in sentence parses are usually indicated with nested parenthesis, during tokenization parenthesis in the text are replaced by other symbols. For example, in PTB trees *(* and *)* are replaced with *-LRB-* and *-RRB-*, respectively, the acronyms denoting Left or Right Curly Bracket. Our implementation uses NLTK *word tokenize.* This tokenizer uses relatively simple heuristics. It does make some mistakes, but not enough to affect the performance of the system significantly. As an example, in some contexts quotation marks are not split from the words they precede or follow.

A large number of URLs occur in the NUCLE training data, as citations are included in some of the essays. We replace these with `<url>` symbols to reduce noise in the vocabulary.

Several symbols that occur in the text data are used by Tiburon as reserved symbols. These include #, @, % and >. Therefore, these symbols should be replaced by placeholder symbols such as -HSH-, -AT-, -PRC- and -GT-.

The last sentence preprocessing step is the normalization of capitalization. We convert all words to lowercase to reduce data sparsity in the constructed models. We store versions of the sentences with the original and the lowercased capitalization, so that the original capitalization can be restored on the system output after decoding. An alternative way to normalize capitalization is to use *truecasing*. In this method, the case of the first word of each sentence (which is always capitalized in English) is restored to its most frequent capitalization. We did perform some experiments using truecasing, but found that it is inadequate to eliminate the occurrence of both capitalized and uncapitalized versions of some words in the text.

Example 5.2.1 The tokenization and lowercasing of a sentence are given below. (1) is the original sentence, (2) the word-tokenized sentence (where all tokens are separated by spaces) and (3) the lowercased form of the tokenized sentence.

(1) *Most Chinese patents are “Appearance Patents”, not “Innovation Patents”.*

(2) *Most Chinese patents are " Appearance Patents " , not " Innovation Patents " .*

(3) *most chinese patents are " appearance patents " , not " innovation patents " .*

5.2.2 Applying corrections

As described in Section 2.2.1, the learner corpora contain annotations for many kinds of errors. In our experiments we construct models to correct only subsets of these errors.

For the FCE corpus we use a set of 9 error types classified according to word classes: Pronoun, conjunction, determiner, adjective, noun, quantifier, preposition, verb and adverb errors. We exclude errors such as spelling, punctuation, word order, idiom and inappropriate register errors.

For the NUCLE corpus we consider the set of five error types used in the CoNLL-2013 shared task: Article or determiner, preposition, noun number, verb form, and subject-verb agreement errors.

For both corpora we construct models to correct these error types. To construct the correct version of the training sentences, we only apply corrections for the error types that we want to correct in a specific model. Spelling and punctuation errors are corrected on the correct and incorrect sides of the training data to reduce noise that these errors may introduce into the model. Annotated errors of other error types are left uncorrected.

An alternative approach is to apply the corrections of the excluded error types to the correct and incorrect versions of the sentences. However, we decided against this in order to keep the training data realistically close to the test data, which also contain these other errors.

A disadvantage of performing the correction task for only a subset of error types is that multiple error annotations in a sentence may interact with each other, and the correction task may only involve performing some of these corrections. As a result some of the gold standard edits will not actually make a sentence more grammatical, as other edits should have been performed as well to make them sensible.

Example 5.2.2 Below we give an incorrect sentence, its error annotations, and the corresponding correct sentence. All the error annotations except the collocation error are applied. This example also shows the disadvantage of correcting only some of the annotated errors, as the phrase *amounts in the billions* is still incorrect.

> Incorrect sentence: *In countries like China and India, their population amounts to billions.*
>
> - Determiner error: *their population* → *the population*
> - Collocation error: *amounts* → *numbers*
> - Preposition error: *to* → *in*
> - Determiner error: *billions* → *the billions*
>
> Correct sentence: *In countries like China and India, the population amounts in the billions.*

5.2.3 Word alignment

The methods that we use to extract transitions or rules for our transducer models are based on *word alignments*. The concept of word alignments was originally developed to align words in sentence pairs used as training data for statistical machine translation models (Brown *et al.*, 1993). We consider alignments between words in the correct and incorrect version of sentences. The word alignment a of a sentence pair (s, t) is a set of pairs such that $(i, j) \in a$ if and only if the ith word in s is aligned with the jth word in t. In contrast to SMT alignment, here most of the words will be aligned to identical words. The edits to transform the one sentence to the other are given by the training data, so we use that to extract the alignments.

In our method, the first step is to align all words that do not occur in any edits one-to-one between the correct and incorrect sentences. Then each edit annotation's incorrect and correct phrases are considered. Words that occur in both the correct and incorrect edit phrases are aligned one-to-one. This is done with a simple left-to-right search through the phrases, with the restriction that alignments may not overlap. Adding these alignments may split an edit phrase into pairs of subphrases without alignments, which may be empty on either side. If such a subphrase is empty on one side, then the words on the other side are left unaligned. But if the subphrases are non-empty on both sides, then the words on the incorrect side are all aligned to each of the words on the correct side of the subphrase. There are relatively few cases where phrases with multiple words on both the correct and incorrect sides are aligned in this way. A further refinement to this alignment procedure would be to align words with the same lexeme or POS tag.

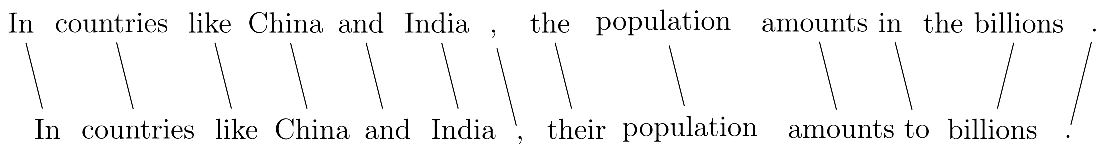

In countries like China and India , the population amounts in the billions .
In countries like China and India , their population amounts to billions .

Figure 5.1: Word alignment between a correct sentence (top) and an incorrect sentence (bottom).

Example 5.2.3 The word alignment for Example 5.2.2 is shown visually in Figure 5.1. No edits are made to the first part of the sentence *In countries like China and India*. For the edit *their population* → *the population*, the word *population* is aligned firstly, as it appears on both sides. Next the words *the* and *their* are aligned. The prepositions in the edit *to* → *in* are aligned. For the final edit, *billions* on both sides are aligned, and *the* is left unaligned.

5.3 Parsing

In order to train a syntax-based model for grammar correction, the sentences on the correct side of the training data are parsed. Initially, we experimented with both the Stanford parser (Klein and Manning, 2003) and the Berkeley parser (Petrov *et al.*, 2006), but we settled on the Berkeley parser, since it is the state-of-the-art unlexicalized parser. Given that the correct side of our training data will still contain errors, it is unlikely that lexicalized parsing will be more accurate.

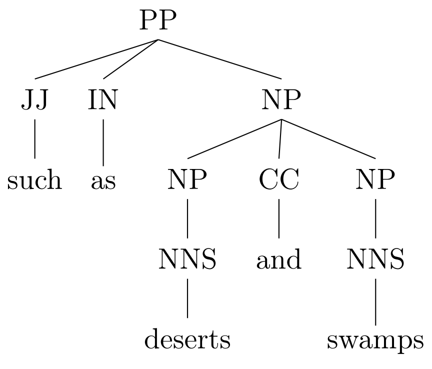
Two versions of the sentence parses are obtained: The first set consist of the standard parse trees, using the default options of the Berkeley parser. The second is the set of left-binarized parse trees obtained under Viterbi decoding, where the best parse is approximated as the parse tree of the highest-scoring derivation. Figure 5.2 shows an example of a standard parse tree (a) and its left-binarized version (b). The file format of parse trees is described in Appendix B.

5.4 Additional resources

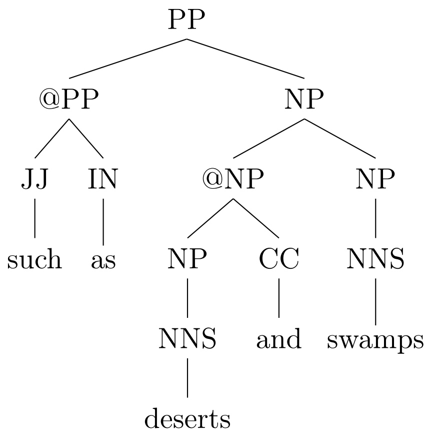
In addition to the learner corpora training data, we use some additional linguistic resources to improve the coverage of our transducer transitions or rules.

The first additional source is a vocabulary. The model vocabulary is usually the union of the language model vocabulary and the vocabulary of the words on the correct side of the training data. We add transitions or rules to our transducer models for all the words in the language model vocabulary which do not occur in the training data. Such rules perform identity rewrites of the words under consideration. For syntax-based rules, the POS tag of a word is also required. We use the NLTK POS tagger to obtain the one or two most likely POS tags for each of the vocabulary words.

Secondly, we use the NLTK interface to WordNet (Miller, 1995) to find pairs of singular and plural nouns and groups of verbs that have the same base lexeme. All verbs and non-proper nouns that occur in the language model vocabulary are grouped like this. These groups are used to construct additional substitution rules



(a)



(b)

Figure 5.2: A parse tree and its left-binarized equivalent.

between singular and plural nouns (in both directions) and between verbs with the same base forms (there are 6 verb forms in the PTB POS tagset that are considered). The motivation behind extracting these rules is that they represent some of the most common grammar errors involving open class words. Errors for open class words are more sparse in the training data than closed class errors such as article or determiner errors, as there are many more open class words than closed class words. Subject-verb agreement errors are also concerned with the verb form in the sentence, so added verb form rules will also be applicable to these errors.

5.5 Language model

We train the n-gram language model used in our models on a large corpus of text extracted from the English Wikipedia. The April 2013 Wikipedia XML dump[1] is used. This XML document is parsed with the *gwtwiki*[2] Wikipedia parser. All sentences consisting of 6 or more words are extracted. These sentences are tokenized

[1] `http://dumps.wikimedia.org/enwiki/20130304/enwiki-20130304-pages-articles-multistream.xml.bz2`

[2] `http://code.google.com/p/gwtwiki/`

with NLTK and lowercased. The corpus has about 1 500 million words. A trigram model is trained using Kneser-Ney smoothing. A 64 000 word vocabulary of the words that occur with the highest frequencies in the corpus are used. The symbol `<unk>` is used to indicate out-of-vocabulary words. The language model is trained with the widely used SRILM toolkit (Stolcke, 2002).

5.6 Decoding and evaluation

In the decoding problem, we are given some text and the goal is to correct the errors in that text. The format of blind test data from a learner corpus is similar to that of the training data, except that the annotations are kept separate. Tokenization is performed as on the training data. Words that do not appear in the vocabulary of the model being used are replaced with `<unk>` symbols. When we decode blind test data, spelling errors may occur in the given incorrect text. Since spelling correction is not in the scope of the models we study, incorrectly spelt words are also replaced by `<unk>`. In practice we found that spelling errors do not have a significant impact on the performance of our system. In the NUCLE test data, spelling errors are very sparse; the proportion of out-of-vocabulary words is 0.03%.

After decoding, some of the preprocessing steps are reversed. The original casing of words is restored. Words replaced with `<unk>` symbols are restored. Additional transformations, such as replaced brackets with special symbols, are reversed. If the output is presented to an end user, the tokenization should also be undone. However, to perform automatic evaluation the tokenization is retained.

The M^2 scorer is used to perform automatic evaluation on both corpora. The gold standard is represented by a file containing the original sentences as well as the edits for each sentence. This file format is described in Appendix B.

5.7 FST error correction model

We now describe a simple FST model for grammatical error correction. Although some of the SMT-inspired approaches in the literature are more general than this model, we are not aware of previous results with a FST model such as the one we describe here. An important criterion in constructing this baseline was that it should contain as little linguistic analysis as possible, so that we can carefully examine the effect of the syntactic information that is expressed by our transducer models. An important advantage of this model is that exact decoding can be performed efficiently with it, which is not the case for our tree transducer models.

5.7.1 Representation and training

We follow the noisy channel model by formulating a language model FSA and an error model FST. In our implementation we use the finite-state transducer package OpenFST (Allauzen *et al.*, 2007). The transitions in a FST are specified by a list of transitions with weights. All transitions between alphabet symbols that are not explicitly specified are assumed to have weight $\bar{0}$. Weights can be specified over different semirings. In our implementation we use the tropical semiring, but in this section the probability semiring is used to clarify the presentation.

We use the Wikipedia trigram model trained with SRILM as language model. In order to construct the language model FSA, we use a script from the *Transducersaurus* toolkit (Novak *et al.*, 2011) to convert the SRILM language model file to OpenFST format.

The error model is a single-state FST that transforms incorrect sentence strings to correct sentence strings. Transitions (with non-zero weights) are extracted directly from the training data word alignments. If word a on the incorrect side and word b on the correct side of the training data are aligned in some training example, a transition labeled with (a, b) is extracted. If a word in the training data is aligned to multiple other words, transitions for all of these alignments are extracted. For unaligned words, transitions of the form (ϵ, b) and (b, ϵ) are extracted. The number of times that each transition (a, b) is extracted, its count, is denoted by $f(a, b)$.

In addition to the transitions extracted from the training data, transitions from extra lexical knowledge sources are also included. For each word w in the language model vocabulary a transition (w, w) is extracted. The initial count associated with the extracted transition is 0. Next, transitions are added for noun number and verb form substitutions. Again, these extracted transitions get a count of 0, as they do not occur in the training data.

Out-of-vocabulary words should also be handled. For this purpose a transition labeled (`<unk>`, `<unk>`), with weight 1, is added. As the `<unk>` symbol is included in the language model vocabulary, this will be handled properly during composition. We do not allow transitions for which `<unk>` occurs only on one side, as this will make restoring the replaced words problematic.

For some extracted transitions, the word on the correct side of the transition label may not occur in the language model vocabulary. In our model we do not extract such transitions.

In order to alleviate the zero count problem with extra added transitions that do not occur in the training data, we perform plus one smoothing. Therefore, for each word pair (a, b) for which $f(a, b)$ is defined, $f(a, b)$ is updated to $f(a, b) + 1$.

As the transducer has only one state, transition weights are estimated as

$$P(a|b) = \frac{f(a, b)}{\sum_{a'} f(a', b)}. \tag{5.7.1}$$

Note that some training example word alignments do not correspond to exactly one path through the transducer. Therefore the normalized frequency counts are not strictly speaking maximum likelihood estimates on the training data, but MLE approximations. Similar count-based estimates are used in phrase-based SMT (Koehn *et al.*, 2003).

5.7.2 Decoding

We use the C++ API of OpenFST to perform decoding. The given incorrect sentences are embedded in transducers and stored in a *finite-state archive*, a compact representation that OpenFST provides. Words that do not appear in the vocabulary are replaced with `<unk>` symbols. After decoding these words are restored.

We use bucket brigade application to perform the decoding. We found that it is much more efficient to perform forward application than backward application when the implementation of composition in OpenFST is used.

In order to decode a sentence, the embedded sentence FSA is composed with the error FST. The error FST transforms an incorrect sentence i into a correct sentence c, even though it represents the probability distribution $P(i|c)$. This transducer is then composed with the language model FSA. This is right projected to obtain the application FSA A, that represents all the possible corrections. The label of the highest probability path through A is taken to be the system output, i.e. the proposed correct sentence. There may be multiple paths through A with the same label, so the highest probability string is approximated by the highest probability path. However, in case of this model there will usually be only one path that corresponds to a sensible sequence of edit operations.

5.8 Conclusion

This chapter described several processing steps and model components that we use in developing grammatical error correction systems. Section 5.1 presented the setup of training and test data. Section 5.2 discussed the preprocessing steps followed to construct the training data that we use. Parsing was described in Section 5.3, while Section 5.4 discussed additional resources used. We discussed language models and decoding in Sections 5.5 and 5.6, respectively. Finally, a baseline FST error correction model was presented in Section 5.7.

Chapter 6

Tree Transducer Models

In this chapter we develop novel models for grammatical error correction based on probabilistic tree-to-string-transducers. We start by describing how syntax-based transducer rules are extracted from training data, and how additional rules are added. We consider modelling choices regarding the structure of the rules and restrictions that may benefit the speed and accuracy of the decoding. Then we discuss the estimation of transducer weights. The steps followed to perform decoding are also described.

6.1 Transducer rules

The training data that we use to construct an error correction transducer are tuples of parsed correct sentences, incorrect sentences and word alignments, obtained as described in Section 5.2. The first step in the construction of our tree transducer models is to construct a set of rules. In this section, we describe our approach, and motivate some of the modelling choices made.

6.1.1 Rule extraction

The main set of rules of our tree-to-string transducer is extracted with the GHKM rule extraction algorithm (Galley *et al.*, 2004, 2006). This algorithm was developed to automatically extract linguistically plausible rules for statistical machine translation, but it can be applied more generally to natural language transformations between parse trees and strings.

A crucial motivation of this algorithm is the relation between xLNTS derivations and word alignments. Suppose we are given a training example (π, i, a), where π is the correct parse tree, i the incorrect sentence and a the word alignment. Suppose that $d = (r_1, \ldots, r_n)$ is a derivation of the tree-string pair (π, i). Each word in i and each node in π is generated by exactly one of the rules in d. The alignment a is *consistent* with d if and only if for each word pair (v, w) in the alignment, v and w are generated by the same derivation rule r. The GHKM algorithm finds derivations for (π, i) that are consistent with a. The rules used in these derivations form the set of extracted rules.

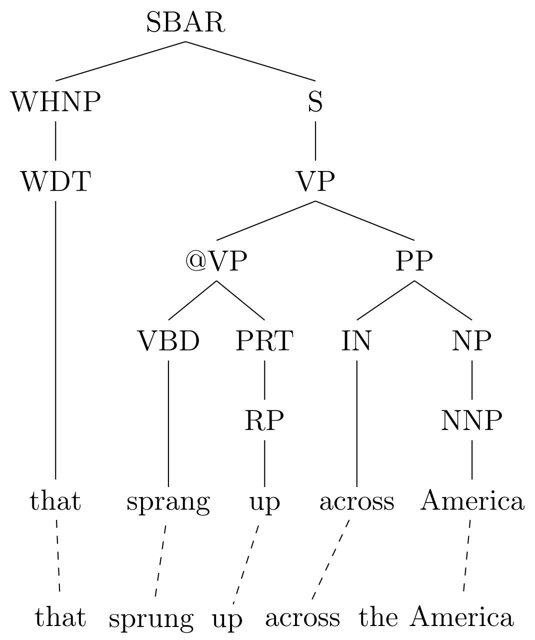


Figure 6.1: Alignment between a correct parse tree and an incorrect clause.

A training example is represented as a directed graph consisting of π, a node for each word in i, and directed edges from leaf nodes of π to word nodes of i, for each word alignment in a. An example of such a graph is shown in Figure 6.1 (all directed edges are assumed to be going downward). Unaligned words in the incorrect sentence are not connected to the rest of the graph. A derivation is induced by partitioning the graph into fragments, such that each fragment corresponds to a transducer rule. A rule is *minimal* if the graph fragment it corresponds to cannot be partitioned into smaller fragments that correspond to valid rules. The goal of GHKM is to extract rules for an xLNTS derivation of (π, i) that are *minimally consistent* (minimal and consistent) with the alignment a.

For each of the nodes in π, we compute a *span* and a *complement span* with respect to the nodes in i. Suppose that the words in i are indexed (the kth word in each sentence has index k). The span of a node n is the integer subset of indices between the first and last words in i that are reachable from n. The spans of the leaves in the tree (the words of the correct sentence) are defined by a, and the spans of the other nodes are computed bottom-up for each node from the spans of its child nodes. The complement span of n is the union of the spans of all nodes that are neither ancestors nor descendants of n. The complement spans are computed top-down for each node n as the union of the complement span of n's parent and the spans of n's siblings. Note that the indices in a complement span may be discontinuous.

Non-leaf nodes of π whose spans and complement spans do not overlap are called *frontier* nodes. From each frontier node n, a rule r is extracted: The left hand side of r is the concatenation of state q and a subtree rooted at n. The subtree is extracted by traversing π top-down from n, replacing all frontier nodes reached with variables, as more rules will be extracted from there. One-symbol look-ahead is added to the variable leaves. The right hand side is formed by the sequence of nodes in i that are spanned by n. For each variable x_i on the left hand side, the right hand side sequence of elements that are spanned by the frontier node corresponding to x_i is replaced by state $q.x_i$. An unaligned word w in i is placed on the right hand side of the rule of the lowest frontier node in π that spans w. The extracted rules correspond to exactly one derivation of (π, i).

Composite rules, that result from the composition of two or more minimal rules (Galley *et al.*, 2006), can also be extracted. Examples of minimal and composite rules are given in Example 6.1.1 below. Suppose that r_1 and r_2 are minimal rules extracted from training example (π, i, a), corresponding to frontier nodes m and n, respectively. Then r_1 and r_2 are composable if n is a descendent of m in π, and if there are no other frontier nodes on the path between m and n. The left hand side of r_2, excluding the state q, is concatenated to the left hand side tree of r_1 at the variable x_i in r_1 associated with the root node of the r_2 tree. On the right hand side of r_1, x_i is replaced by the right hand side of r_2, and the variables indices are renumbered, if necessary. Although rules can be composed recursively, for our models that include composite rules, we only extract compositions of two minimal rules, and only in cases where the composite rule performs a lexical change. These restrictions limit the number and size of rules, which impact the transducer decoding time.

Rules are extracted from all examples in the training data. The number of times each rule r is extracted is denoted by $f(r)$. This will be used to determine the rule weights. We implemented this algorithm in Python. The linear time algorithm described by Galley *et al.* (2004) is implemented, and the run time of our implementation is comparable to what they reported, at around 10 000 sentence pairs per minute.

We briefly discuss the kinds of rules that can be extracted with the aim of performing grammatical error correction. All non-lexical rules will be identity rules, with non-extended left hand sides. All extracted lexical identity rules will have the form $q(POS(w)) \rightarrow w$, where w is a word and POS is a part-of-speech tag. Most of the extracted transducer rules will have this form, though they do not perform edits. Single-word, context-independent lexical rewrites have the form $q(POS(w)) \rightarrow v$. More complex types of rewrite rules include constituent phrase rewrites, non-constituent phrase rewrites and context-sensitive word insertions and deletions, as described in Section 3.5.1. Although the rule extraction algorithm can extract non-contiguous phrase rewrites and constituent re-orderings, these occur very rarely in training data for the kind of grammar errors we are interested in correcting, and therefore we do not consider these rules in our models.

Minimal rules only take into consideration the minimal amount of context needed to perform rewrites. In some cases, a larger context is needed to model all words and syntactic elements that may have an influence on the grammar error represented by a rewrite rule. Therefore it is beneficial to include composite rules, although these rules present challenges to efficient training and decoding.

Example 6.1.1 Figure 6.2 shows the rules extracted from the training example in Figure 6.1. Rules (1) to (13) are minimal rules, while rules (14) to (17) are composite rules. Rule 14 is the composition of rules 6 and 7, 15 of rules 10 and 11, 16 of rules 10 and 12, and rule 17 of rules 5 and 10. In the example tree, all the constituent (non-

(1) q.SBAR (x_1:WHNP x_2:S) $\rightarrow$ $q.x_1$ $q.x_2$

(2) q.WHNP (x_1:WDT) $\rightarrow$ $q.x_1$

(3) q.WDT (*that*) $\rightarrow$ *that*

(4) q.S (x_1:VP) $\rightarrow$ $q.x_1$

(5) q.VP (x_1:@VP x_2:PP) $\rightarrow$ $q.x_1$ $q.x_2$

(6) q.@VP (x_1:VBD x_2:PRT) $\rightarrow$ $q.x_1$ $q.x_2$

(7) q.VBD (*sprang*) $\rightarrow$ *sprung*

(8) q.PRT (x_1:RP) $\rightarrow$ $q.x_1$

(9) q.RP (*up*) $\rightarrow$ *up*

(10) q.PP (x_1:IN x_2:NP) $\rightarrow$ $q.x_1$ *the* $q.x_2$

(11) q.IN (*across*) $\rightarrow$ *across*

(12) q.NP (x_1:NNP) $\rightarrow$ $q.x_1$

(13) q.NNP (*America*) $\rightarrow$ *America*

(14) q.@VP (VBD (*sprang*) x_1:PRT) $\rightarrow$ *sprung* $q.x_1$

(15) q.PP (IN (*across* x_1:NP)) $\rightarrow$ *across the* $q.x_1$

(16) q.PP (x_1:IN NP (x_2:NNP)) $\rightarrow$ $q.x_1$ *the* $q.x_2$

(17) q.VP (x_1:@VP PP (x_2:IN x_3:NP)) $\rightarrow$ $q.x_1$ $q.x_2$ *the* $q.x_3$

Figure 6.2: Rules extracted from Figure 6.1.

leaf) nodes are frontier nodes. We see that the composite rules, by taking a larger context into consideration, include additional syntactic context for the rewrites they perform.

6.1.2 Additional rules

The vocabulary of our tree transducer models is the union of the vocabulary of our language model and the vocabulary of the words of the correct sentences in the training data. In contrast to the FST model, we do allow extracted rules whose correct side contains a word that is not in the language model vocabulary. Lexical rewrite rules of the form q.POS(w) $\rightarrow w$ are added for all words w in the language model vocabulary. For pairs of nouns and groups of verbs, rules of the form q.POS(v) $\rightarrow w$ are added for a substitution between words v and w, where POS is the POS tag of v. All added rules that have not already been extracted from the training data are assigned a rule count of 0. The rule counts of existing rules are left unchanged. For each POS tag, a rule $q.\text{POS}(\langle unk \rangle) \rightarrow \langle unk \rangle$ is also added to accept out-of-vocabulary words.

6.1.3 Binarization and rule filtering

The sizes of rules and, more specifically, the number of variables in rules, have a considerable impact on the decoding time of xTS models. In this section we discuss a number of strategies to make efficient decoding possible. In practice the grammar constant for parsing with xTS is larger than that of standard parsing, which makes xTS parsing several times slower than PCFG parsing.

A rule is *binarized* if it has at most 2 variables. Binarization is important in parsing, as algorithms such as CKY require binarized grammars to perform cubic time parsing. A derivation tree of a tree-string pair in an xTS is binarized if the rules used in the derivation are binarized. Performing backward application of a string s to an xTS is equivalent to parsing s to find an xTS derivation tree. If the CKY algorithm is used to perform decoding, the xTS rules should be binarized. Rules should also be binarized to perform EM training.

The first solution is to perform *synchronous binarization* (Zhang *et al.*, 2006). Firstly, rules are extracted without any restrictions on the number of variables. Then, as far as possible, rules that have more than two variables are converted to binary rules. However, not all synchronous grammars are binarizable. Some types of rules that permute the variable ordering between the rule LHS and RHS cannot be rewritten into equivalent binarized rules. In our tree transducers we do not use rules that reorder variables, so in principle all the rules can be binarized. However, binarization may markedly increase the number of rules and the number of symbols in the transducer, which has a detrimental effect on decoding time.

Another solution is to binarize the trees in the training data *before* rules are extracted. This does not guarantee that the extracted rules are binarized, but it does noticeably decrease the number of rules that are not binarized. Transducer rules extracted from binarized trees will not be equivalent to rules extracted from the original trees. However, an advantage of rules extracted from binarized trees is that all unlexicalized rules and most lexicalized rules will be binarized. Binarizing tree fragments with symbols that have a large rank have an effect similar to the Markovization of productions in parsing models. We use left-binarized trees. Marcu *et al.* (2007) used the EM algorithm to automatically learn preferences for left- and right-branching binarization, in the context of machine translation. They found that, for most constituents, left-binarization is preferred.

Another alternative, that does not involve binarization, was proposed by Hopkins and Langmead (2010). The class of binarized rules is expanded to a class called scope-3, which still allows cubic time parsing. The *scope* of a rule is the number of pairs of consecutive variables in $x_i \cdot \gamma \cdot x_j$, where γ is the rule right hand side and x_i and x_j are variables. The time complexity of decoding is $O(n^p)$, where n is the sentence length and p is the maximum scope of transducer rules. For each pair of consecutive variables, the parser needs to consider up to n possible splits in the input sentence that correspond to the different ways of matching the variables to sentence fragments. To obtain a scope-3 transducer, rules that have a scope greater than 3 are simply pruned. No other changes are made to the tree structure. Decoding with these transducers are performed with chart parsing or Earley parsing (Chung *et al.*, 2011).

In our experiments we perform xTS decoding with Tiburon. The version of the Earley parsing algorithm implemented to perform the decoding, implicitly binarize rules during decoding. The parsing algorithm should run in cubic time even if rules with more than two variables are allowed. However, in practice we found that including rules without restrictions on the number of variables noticeably increases decoding time. While it is unclear exactly what causes this, there are a number of possible explanations. These include the overhead in performing implicit binarization, the general detrimental effect of having large rules, and inefficient implementation in Tiburon.

In our models we experiment with extracting rules from both the standard and binarized trees. We use scope-3 pruning on both kinds of trees. With the binarized trees, we alternatively pruned all rules with more than two variables. Furthermore, in all models rules that have more than 8 words on the right hand side are pruned.

6.2 Training

Next we discuss the training of our tree transducer models.

6.2.1 Smoothing

Sparse data is an inherent property of any real text corpus, encountered when collecting frequency statistics from a finite sized text (Katz, 1987). In the context of transducer rules extracted for grammatical error correction, a large proportion of rules occur only once or a few times in the training examples. The additionally added rules originally have a count of 0. This sparseness decreases the accuracy of probability estimates obtained by maximum likelihood estimation. In order to address this problem, we apply smoothing to the rule counts.

Following the practice of other syntax-based tree transducer models (e.g. Chiang *et al.* (2009)), we apply Good-Turing smoothing (Katz, 1987). Though this re-estimation technique was developed for n-gram models, it is considered good practice to apply it to transducer rule counts. The basic idea is to remove some probability mass from unreliable probability estimates of observed events (extracted rules) with low frequencies. This weight is then redistributed to events which never occur in the training data.

Let N be the total number of events we are considering, and n_r the number of events that occur exactly r times in the training data, so that $N = \Sigma_r r \cdot n_r$. Then the count of an event that occurs r times is re-estimated as

$$r^* = (r+1)\frac{n_{r+1}}{n_r}. \tag{6.2.1}$$

The *discounting coefficient* is $d_r = r^*/r$. The idea here is that counts of events that occur r times are re-estimated as the total count of events that occur $r+1$ times. This implies that events which occur 0 times are assigned the total frequency of events that occur once. This weight is then distributed amongst all count 0 events.

The total mass of events that occur once is re-estimated as the total mass of events that occur twice, etc.

It is possible to re-estimate the probabilities directly from these new counts. A problem that arises is that there will be some value $r \geq 0$ for which $n_r > 0$ and $n_{r+1} = 0$, leading to $r^* = 0$. However, usually that value of r is relatively large, and we are primarily interested in re-estimating small values of r. Katz (1987) proposed modifying the distribution of weights so that counts $r > k$ are considered to be reliable. The original discount coefficient is modified so that the sum of the re-estimated counts are still equal to N over all the events. Then we have

$$d_r = \begin{cases} (\frac{r^*}{r} - \frac{(k+1)n_{k+1}}{n_1})/(1 - \frac{(k+1)n_{k+1}}{n_1}), & 1 \leq r \leq k \\ 1, & r > k. \end{cases} \tag{6.2.2}$$

The re-estimated count for r is then $r \cdot d_r$. In practice, $k = 5$ is a good choice, and we use that in our models. In our implementation we add some additional checks to ensure that $0 < d_r \leq 1$, for $r \leq k$.

6.2.2 Weight estimation

The tree transducer models that we use represent joint probability distributions over correct parse trees and incorrect sentences. We consider two methods to train the transducer models. The first uses relative frequency estimates, while the second uses the EM algorithm.

When an incorrect sentence is decoded there is spurious ambiguity in the model at two levels: Firstly, it is possible that there can be different derivations for the same tree-string pair. During the application of the model this ambiguity occurs infrequently. One way to overcome this ambiguity is by performing weighted determinization on the application RTG (Büchse *et al.*, 2010). Not all RTGs can be determinized, but as a special case RTG without cycles can be. Weighted determinization is implemented in Tiburon. Secondly, the model can generate different trees with the same yield.

Suppose that c is a correct sentence in the set C of all possible correct sentences, and i is the given (possibly) incorrect sentence. Let $\tau(c)$ represent the set of all parse trees of c. Then we want to find the sentence

$$\begin{aligned} \hat{c} &= \arg\max_{c \in C} P(c|i) && (6.2.3) \\ &= \arg\max_{c \in C} P(c,i) && (6.2.4) \\ &= \arg\max_{c \in C} \Sigma_{\pi \in \tau(c)} P(\pi, i) P(c|\pi) && (6.2.5) \\ &= \arg\max_{c \in C} \Sigma_{\pi \in \tau(c)} P(\pi, i). && (6.2.6) \end{aligned}$$

In the case of minimal rules, the rules extracted from each training example corresponds to exactly one derivation. Let $f(r)$ be the smoothed count of the number of times that rule r occurs in all training derivations. Then the probability estimate of a rule is

$$p(r|root(r)) = \frac{f(r)}{\Sigma_{r':root(r')=root(r)} f(r')}. \tag{6.2.7}$$

As we work with a one-symbol look-ahead transducer, the root of a rule consist of the state and the head constituent node of the left hand side of the rule. The re-estimated rule counts used in smoothing are calculated separately for each of the roots.

An alternative parameterization is to condition rule probabilities on entire left hand sides, excluding the variable nodes. In this case the extended tree structure is encoded in the states of the transducer. The potential advantage of this approach is that a larger look-ahead is used to decide how rules are applied.

When composite rules are also included, we estimate the weights similarly. In this case the rules extracted from a training example do not correspond to exactly one derivation any more. However, the frequencies of the extracted composite rules still provide reasonable probability estimates.

We also experimented with using Tiburon to perform xTS training with the EM algorithm. Unfortunately we were not able to obtain results with EM training. Constructing derivation grammars from the training data took over 300 CPU hours, and after that Tiburon was not able to estimate sensible weights for the transducer rules, possibly due to numerical underflow.

Example 6.2.1 Rules extracted by our approach are shown in Table 6.1. Rule weights (obtained by relative frequency estimation) are also included. The weights are given as log (base 10) probabilities. A classification of the different kinds of rewrites that the rules perform are also given. The classification is similar to that used in Section 3.5.

6.3 Decoding

In this section we discuss how decoding is performed with our tree transducer model. Firstly, sentences are split into clauses. For each sentence the k best corrections proposed by the tree transducer are found and then reranked using an n-gram language model.

6.3.1 Sentence splitting

Long sentences present a particular challenge to our tree transducer models. In the NUCLE training data, the average sentence length is 20.38 words. 46% of the sentences are longer than 20 words, and 13% have length greater than 30.

The decoding time of a tree-to-string transducer for a sentence of length n is cubic. In practice this means that decoding time for sentences of length more than 20 becomes really long, even if heuristic pruning is applied to the search space during decoding.

One may argue that for grammatical error correction much of this additional complexity seems unnecessary, as we are only interested in rewriting words or phrases consisting of a few words, and not to restructure entire sentences. One of the main hypotheses that we investigate in this thesis is that non-local syntactic information play a useful role in correcting grammatical errors. However, as we saw in Section 2.1, much of the syntactic structure is limited to *clauses*. For long sentences

Rule	Weight
Non-lexicalized	
q.S (x_0:NP x_1:VP) $\rightarrow q.x_0\ q.x_1$	-0.596
q.S (x_0:VP x_1:VP) $\rightarrow q.x_0\ q.x_1$	-5.781
q.VP (x_0:VP x_1:SBAR) $\rightarrow q.x_0\ q.x_1$	-3.723
Word identity	
q.NN (*work*) $\rightarrow$ *work*	-2.614
q.VBP (*work*) $\rightarrow$ *work*	-2.475
q.DT (*the*) $\rightarrow$ *the*	-0.183
Single word substitution	
q.NN (*work*) $\rightarrow$ *works*	-4.343
q.VBP (*work*) $\rightarrow$ *working*	-4.541
q.VBZ (*works*) $\rightarrow$ *work*	-4.802
q.DT (*the*) $\rightarrow$ *a*	-3.100
q.IN (*of*) $\rightarrow$ *from*	-3.901
Constituency phrase substitution	
q.NP (DT (*the*) NN (*right*)) $\rightarrow$ *rights*	-5.109
q.VP (VBG (*being*) VP (VBN (*researched*))) $\rightarrow$ *under researching*	-6.272
Context-sensitive phrase substitution	
q.VP (TO (*to*) VP (VB (*work*) x_0:PP)) $\rightarrow$ *working* $q.x_0$	-6.272
q.PP (IN (*in*) S (VP (VBG (*generating*) x_0:NP))) $\rightarrow$ *to generate* $q.x_0$	-5.480
Context-sensitive word insertion and deletion	
q.NP (DT (*the* x_0:NN) $\rightarrow q.x_0$	-2.634
q.VP (VBZ (*has*) x_0:VP) $\rightarrow q.x_0$	-5.202
q.VP (x_0:VB x_1:NP) $\rightarrow q.x_0$ *into* $q.x_1$	-5.203

Table 6.1: Example transducer rules by type, with log probability weights.

which consist of multiple clauses, the information needed to make decisions regarding grammaticality will in most cases be limited to clausal level.

Therefore, to reduce the complexity of decoding long sentences, we perform linguistically motivated sentence splits on the sentences before decoding them. The idea is to split sentences into clauses, decode each of the clauses separately, and then recombine the output clauses to reconstruct the system output sentences. Sentence splitting is based on constituency parses (obtained with the Berkeley parser) of the incorrect sentences under consideration. The goal is to extract clauses whose form is similar to full sentences, though due to the hierarchical structure of clauses this is not always possible.

We distinguish between *S-clauses* and *SBAR-clauses*. Using PTB syntactic tags, we call clauses annotated with S, SINV or SQ tags *S-clauses*, and clauses annotated with SBAR or SBARQ constituents *SBAR-clauses*. A S-clause usually has the form of a complete sentence. A SBAR-clause usually consists of an introductory subordinating conjunction or *wh*-word, followed by a S-clause.

Our basic approach is to perform splits on S-clauses. A split is performed between

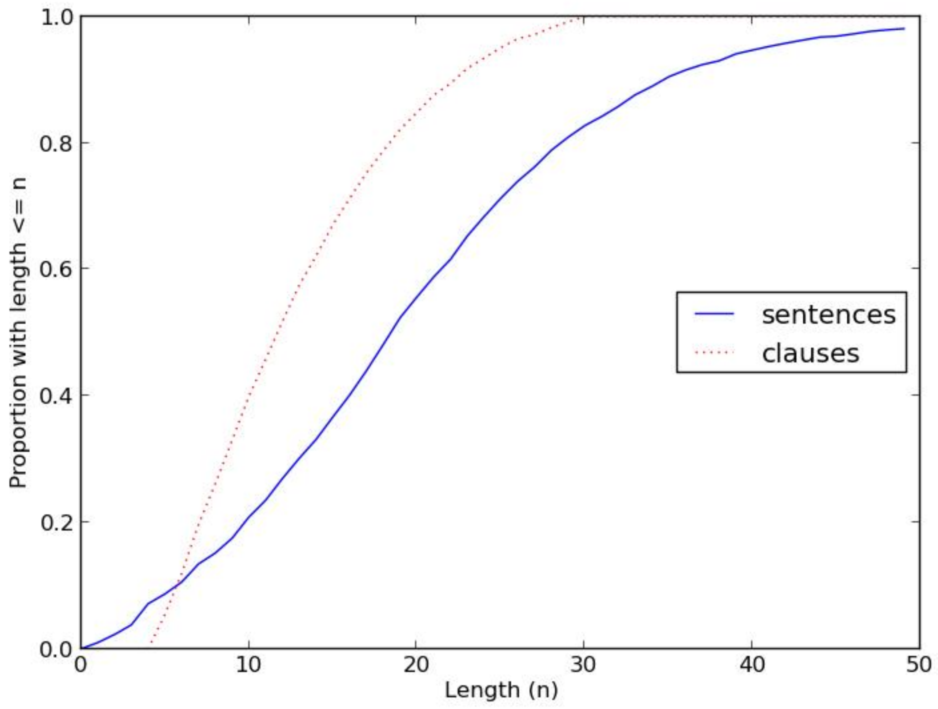


Figure 6.3: The effect of clause splitting on sentence lengths.

the phrase before the start position of a S-clause, which is seen as one clause, and the phrase after that position, which is seen as another clause. If the parse tree node of the S-clause is the child of a SBAR-clause node, the split is performed between the phrase before the starting position of the SBAR-clause, and the phrase after the start of the S-clause. The introductory word(s) in the SBAR-clause are excluded from the extracted clauses.

Splits are also performed between phrases separated by a coordinating conjunction (indicated by a CC tag), if the CC node is a child of a S-clause node. The phrase before the conjunction is split from the phrase after the conjunction, while the conjunction itself is excluded.

Extracted clauses that are shorter than 30 words and longer than 4 words are decoded. After decoding and reranking have been performed, the output clauses and words excluded from clauses are recombined to reconstruct the original sentences.

The NUCLE test set consists of 1381 sentences, which was split into 2247 clauses using our clause splitting heuristic. Figure 6.3 shows the distribution of sentence lengths before and after the sentences are split.

Example 6.3.1 A split of the sentence *The solution can be obtained by using technology to achieve a better usage of space* is shown in Figure 6.4. The parse tree of the complete sentence is decomposed using our clause extraction algorithm. Context trees are shown for the clauses, with the variable x_1 indicating where a split has been performed.

6.3.2 k-Best decoding

A method that is often used in NLP to combine models is *k-best reranking*. The given sentence is applied to a first model (in our case, a tree transducer). The k highest-scoring outputs from the application are then *reranked* using a second model. The scores of the two models are combined, and the hypothesis with the highest weight after rescoring is chosen as the output. In our models, we found that a good trade-off between speed and accuracy is to find a list of trees of the 1000-best derivations for a given (incorrect) sentence. The weights of different derivations for which the parse trees have the same yields, are then summed to find weights for each of the hypothesis sentences. This is an approximation of equation (6.2.6), which takes the sum over all parse trees with the same yield. We use Tiburon's implementation of backward application to xTSs to obtain the trees of the k-best derivations.

As the search space of the model is large, we need to apply some heuristic pruning. Following practices used in parsing models such as (Huang and Chiang, 2005) and translation models such as (Chiang, 2007), beam search is performed. The cell limit γ, the maximum number of hypotheses that can be kept at a state during decoding, is set to 30. The beam width β is set to 10^{-4}. This means that if a hypothesis score is worse than β times the score of the best partial hypothesis found up to a specific point in the model, the hypothesis is discarded. The parameters γ and β were tuned to make decoding feasible on a desktop computer with 8GB RAM, and so that it does not take more than 1 minute on average to perform tree transducer decoding.

6.3.3 Language model reranking

Although the tree transducer model defines a joint probability distribution, incorporating an n-gram language model into our system considerably increases its performance. The main reason for this is that the generative transducer model alone does not have enough discriminative power to distinguish between well-formed and ungrammatical sentences. Another reason is that we do not follow practices such as using a lexicalized grammar or splitting constituency symbols to reduce independence assumptions. We use SRILM to compute the language model score for each of the hypothesis sentences generated by the transducer model.

In order to combine the transducer model and language model scores, the log probabilities of these scores are normalized by the length of the incorrect sentence. For an incorrect sentence i and a correct sentence c, the normalized transducer model score is represented by $TT(c, i)$ and the normalized language model score by $LM(c)$. For a given incorrect sentence i and a set of hypothesis sentences $H(i)$ obtained from the transducer model, we want to find

$$\hat{c} = \arg\max_{c \in H(i)} [TT(c, i) + \alpha \cdot LM(c)]. \tag{6.3.1}$$

The parameter α is set discriminatively to maximize the F1 score of the model on a validation set. Let I be the set of incorrect sentences in the validation set. Then we want to find

$$\hat{\alpha} = \arg\max_{\alpha} \mathrm{F1}[\Sigma_{i \in I} edits(\hat{c}, i, g(i))], \tag{6.3.2}$$

where $\hat{c}$ is given by (6.3.1) and *edits* is the sufficient statistics for the F1 score of $\hat{c}$ for the incorrect sentence i and gold standard edits $g(i)$.

6.4 Conclusion

This chapter set out the tree transducer models we propose to perform grammatical error correction. Section 6.1 described the formulation of transducer rules. Transducer training and decoding were described in Sections 6.2 and 6.3, respectively. In the next chapter we present results of experiments performed with different configurations of our models.

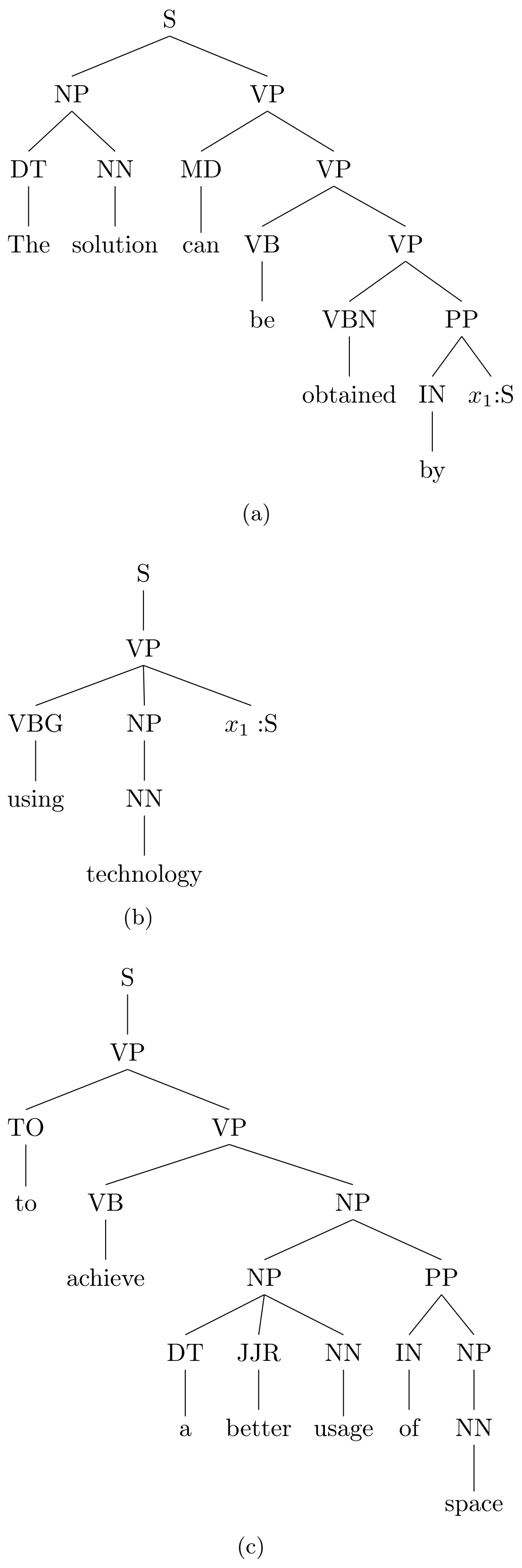


Figure 6.4: Parse trees of a sentence split into clauses.

Chapter 7

Results

In this chapter we present results for our grammar correction methods. We start by evaluating our baseline FST model. Then results for different modelling choices and parameterizations for our tree-to-string transducer models are given. We give results for the performance of different system components, as well as the performance of our approach on different error types. The NUCLE dataset is used for most of the analyses of modelling choices. However, we also give results for the FCE dataset, covering a broader set of errors. Most of the experiments reported in this chapter were performed on the Stellenbosch High Performance Computer.

7.1 FST model

Firstly, we give results for the baseline FST model, trained on the NUCLE data set to correct the five error types we consider for that corpus. The FST model does not have hyperparameters, so the validation sets are not used. Decoding is performed on the development and test sets. The precision, recall and F1 scores are given in Table 7.1, as percentages.

Data set	Precision	Recall	F1
Development	6.77	27.94	10.89
Test	24.30	27.37	25.74

Table 7.1: Results for the FST model on NUCLE.

Interestingly, the model recall on the two sets are almost equal, but the precision is much higher on the test set than on the development set. The main explanation for this is that the test set contains a larger proportion of errors than the NUCLE training data (from which the development set was drawn). As a result the system generates more false positives (changes made to correct words or phrases) on the development set than on the test set.

In Table 7.2 we give a breakdown of the scores for individual error types. In Section 7.4 these results are compared to that of our tree transducer models. As the system output is not labeled with error types of edits, we do an automatic

Data set	Development			Test		
Error type	P	R	F1	P	R	F1
Determiner	7.28	33.33	11.95	34.98	41.97	38.16
Preposition	4.75	25.90	8.04	11.86	14.15	12.90
Noun number	18.45	28.51	22.40	39.47	22.73	28.85
Verb form or SVA	11.65	17.13	13.87	14.21	10.57	12.12

Table 7.2: Results for the FST model on NUCLE, for each error type.

classification of error types, similar to the method of Ng *et al.* (2013), to compute the precision of different error types.

Especially interesting is the high performance of determiner error correction on the test set. An error analysis shows that a very common determiner error in the test set is unnecessary determiner errors, where the occurrence of an article is incorrect and the article should be deleted. The reason for the high frequency of these errors lies in the profile of the learners who wrote the NUCLE essays. Most of these learners are native speakers of Chinese, a language without articles. However, the learners are already relatively proficient in English, so they know that articles should be used, but now tend to use them in contexts where they should not be used as well.

The error transducer has transitions to insert or delete determiners independent of any context. The language model judges the grammaticality of hypothesis corrections with and without articles. Though the trigram model is a simple model, the results indicate that in many cases it is able to judge correctly that a determiner should be deleted. Furthermore, the language model gives a preference to shorter sentences, as they tend to have larger probabilities, which also increases the chance that the model proposes determiner deletions.

For the other error types, the general trend is that the recall is lower on the test set than on the development set, but the precision is higher. As both verb form and subject-verb agreement errors involve a change in verb form, we do not distinguish between them in the error classification. However, using the gold standard edit labels, we can obtain separate recall results. On the development set, recall is 24.13 for verb form errors and 10.64 for subject-verb agreement. On the test set, a recall of 14.75 for verb form errors and 6.45 for subject-verb agreement is obtained. Therefore, the model performance on subject-verb agreement errors is quite low. This can be explained by the fact that in most subject-verb agreement learner errors the verb of a clause and the head noun of the subject noun phrase preceding the clause are not adjacent to each other. If the distance between the subject head noun and the verb is greater than or equal to n, the dependency cannot be modelled by an n-gram model.

7.2 Language model reranking

In this and the following sections we present results for our tree transducer models.

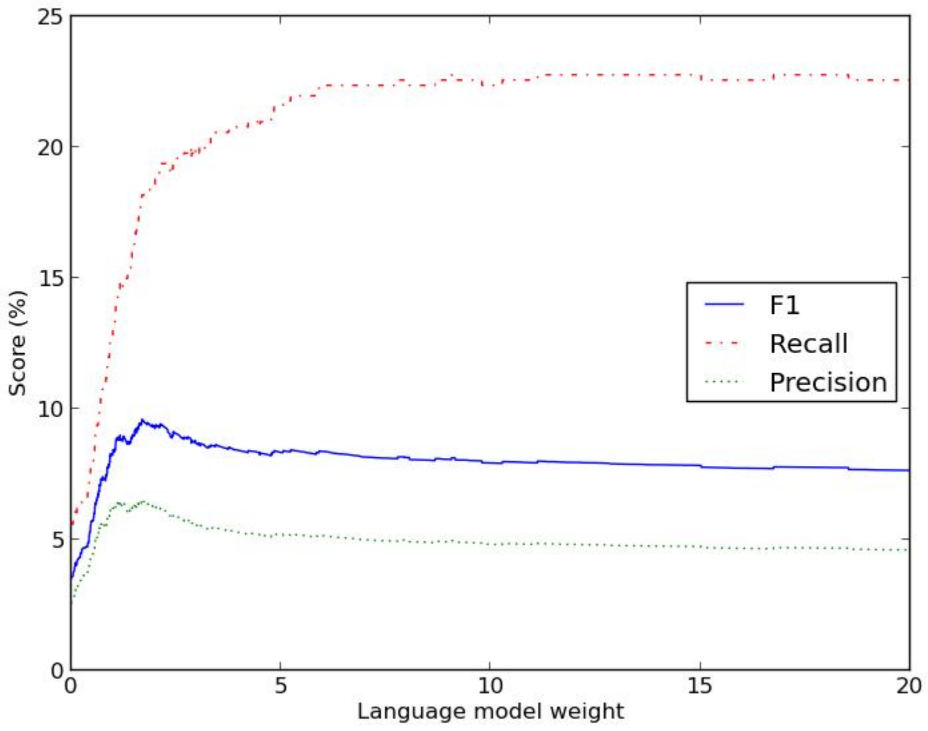


Figure 7.1: Precision, recall and F1 scores for different language model weights α on the validation set.

7.2.1 Language model weight

As explained in Section 6.3.3 the hypothesis clauses generated by the xTS model are reranked by weighing the xTS and n-gram language model scores of each hypothesis. The validation set is used to set the value of α, the weight of the language model score.

For the minimal rules, the best F1 score is obtained with $\alpha = 1.69$, and when composite rules are also included, with $\alpha = 1.59$. Figure 7.1 shows how the precision, recall and F1 scores are affected by the choice of α on the validation set when the rule set includes both minimal and composite rules.

The precision and recall both increase initially as the value of α is increased. The recall is always greater than the precision, and increases faster than the precision. When the value of α becomes greater than 5, the scores decrease, but only slightly.

The stronger the weight on the language model, the higher the recall of the model. As the language model score is only dependent on the proposed correction, it will give preference to hypotheses with a high recall (that are more grammatical), without regard to the precision (as it does not consider the original versions of the hypotheses).

Rule set	Minimal	Composite
Data set	Accuracy	Accuracy
Development Oracle no change	99.2	99.4
Development Oracle change	46.1	60.7
Test Oracle no change	98.4	100.0
Test Oracle change	38.8	50.1

Table 7.3: Oracle hypothesis coverage results on NUCLE.

7.2.2 Clause level results

As we use a combination of two models, it is useful to measure the performance of the different components. A way to do this is to perform *oracle reranking* on the hypothesis sets generated by the transducer model. For each sentence, the oracle picks the hypothesis with the highest sentence-level F1 score. This is an approximation of selecting hypotheses to maximize the overall F1 score. We are especially interested in the *hypothesis coverage*, the proportion of times that the correct clause occur among the hypothesis clauses.

Oracle hypothesis coverage results for the clausal level at which decoding is performed are given in Table 7.3. In a typical decoding experiment, the 1000-best list generated by the tree transducer contains on average 75.7 distinct hypotheses. If a sentence that has one or two errors is split into multiple clauses, it is likely that some of the clauses will have no more errors. Consequently, the proportion of clauses that have no errors is higher than the proportion of sentences that have no errors. On the development set, 82.5% of clauses should not be changed, while on the test set that drops to 54.2%. When no changes should be made to a clause, the correct (unchanged) clause is included in almost all the generated hypothesis sets. For clauses that should be changed, the oracle accuracy is less. The proportion of hypothesis sets that include the correct clause increases when composite rules are included, on both the development and the test sets.

7.3 Rule sets

In this section we present results regarding modelling choices in the construction of rule sets.

7.3.1 Minimal and composite rules

We performed experiments with rule sets consisting either only of minimal rules, or of minimal and composite rules. Results for these models are given in Table 7.4.

On the development set, including the composite rules increases the recall. The F1 score increases slightly, though not very noticeably. On the oracle scores we can clearly see the advantage of including composite rules. However, this does not translate into a similar improvement in the model performance after reranking. A reason for this may be that the n-gram model does not assign sufficiently high scores to the hypothesis generated by using composite rules, as they involve longer-distance dependencies.

Rule set	Minimal			Composite		
Data set	P	R	F1	P	R	F1
Development	8.03	15.02	10.46	8.01	17.12	10.91
Development Oracle	66.7	59.8	63.1	75.54	70.95	73.18
Test (original)	21.81	13.33	16.55	27.00	13.33	17.85
Test (revised)	35.96	14.61	20.77	37.12	18.91	25.05

Table 7.4: Results for minimal and composite rules.

For the test set, including composite rules increases the model performance. For our submission to the CoNLL-2013 shared task (Buys and van der Merwe, 2013), the version including composite rules was used. The recall on the test set and the development set are very similar. As with the FST model, the precision of the model is noticeably better on the test set than on the development set.

Our model F1 score of 17.85% ranked 6th out of the 17 entries to the CoNLL-2013 shared task. The highest-ranked team, from the University of Illinois, obtained recall, precision and F1 scores of 23.49%, 46.45% and 31.20%, respectively. They used statistical classifiers, while the second placed team, from the National Tsing Hua University, followed a language modelling approach to obtain an F1 score of 25.01%. Our model outperformed both a phrase-based SMT approach, with an F1 score of 16.06%, and a noisy channel model word transformation approach, with an F1 score of 7.56%. Interestingly though, our FST model would have ranked second, with an F1 score of 25.75%, mainly due to its strong performance in determiner error correction on the test set.

The revised test set includes alternative annotations we suggested based on plausible corrections in our system output that were not included in the original annotations. This explains why the model scores increase more on the rule set including the composite rules than on the minimal rule set. In the shared task, our model also ranked 6th when scored with the alternative annotations. The score of the top-ranked University of Illinois team increased to 42.14% in this case. As the revised annotations favour the specific models for which the alternative annotations were suggested, we do not use it in further model comparisons.

7.3.2 Rule binarization and pruning

Next we give results regarding the binarization of rules and related modelling choices in the construction of the rule set. The NUCLE development set is used, and only minimal rules are included. See the results in Table 7.5. The results show that applying scope-3 pruning to standard parse trees outperforms using binarized trees with all non-binarized rules pruned. Adding extra look-ahead to the context of the next rule to be applied does not improve performance for the standard (unbinarized) trees. However, it does improve the performance slightly when binarized trees are used. The reason for this is that there is less sparsity in the rules extracted from binarized trees than from standard trees. Therefore, the additional parameters that arise from the extra look-ahead can be estimated more accurately than when standard trees are used.

Rule set	Precision	Recall	F1
Binarized	8.03	15.02	10.46
Standard, Scope-3	7.88	19.43	11.21
Standard, look-ahead	5.80	19.33	8.92
Binarized, look-ahead	9.79	15.02	11.86

Table 7.5: Results for different rule set binarization and pruning methods.

Rule set	Minimal			Composite		
Error type	P	R	F1	P	R	F1
Determiner	8.00	15.90	10.64	8.92	15.64	11.36
Preposition	6.32	11.51	8.16	7.19	16.55	10.02
Noun number	16.40	17.36	16.87	12.59	22.31	16.10
Verb form or SVA	7.21	12.71	9.20	7.08	13.81	9.36

Table 7.6: Results on the NUCLE development set, for each error type.

Error type	Minimal			Composite		
	P	R	F1	P	R	F1
Determiner	29.56	17.39	21.90	30.53	12.59	17.83
Preposition	12.60	5.14	7.31	26.36	9.32	13.78
Noun number	38.92	14.64	21.28	43.63	18.18	25.67
Verb form or SVA	10.55	10.16	10.35	17.13	12.60	14.52

Table 7.7: Results on the NUCLE original test set, for each error type.

7.4 Performance on different error types

Next we analyse the performance of our models on different error types on the NUCLE data set. The results for the development set are given in Table 7.6, and for the test set in Table 7.7. In most cases the performance improves when composite rules are included. The exceptions are for noun number errors on the development set (though the difference is very small), and for determiner errors on the test set. As on the FST model, the precision is higher on the development set than on the test set, while the recall is lower in most cases (with the exception of determiner errors for the minimal rule set).

For determiner errors, the FST model performs better than the xTS models, especially on the test set. The xTS model also performs worse when composite rules are included. A possible explanation for this is that when more context is added to rules that perform determiner edits, the weight of these rules cannot be estimated accurately, as each of the contexts do not occur often enough in the training data. Another weakness of the xTS model is that there are no parameters that control the preference for performing insertions or deletions of word classes. We observe that the model tends to perform too many unnecessary deletions, especially of non-article determiners and content words, while performing too few insertions such as article insertions.

Data set	Development		Test	
Rule set	Minimal	Composite	Minimal	Composite
Verb form	18.39	18.39	13.11	14.75
Subject-verb agreement	7.45	16.55	7.26	9.32

Table 7.8: Recall for verb form and subject-verb agreement errors.

For preposition errors, the xTS model performs very similarly to the FST model on the development set for minimal rules, but when composite rules are included in the xTS model, it outperforms the FST model. On the test set, the xTS with minimal rules performs worse than the FST model, but when composite rules are included it performs better. This shows that the additional context in composite rules is very beneficial to correcting preposition errors.

For noun number errors, the FST model performs better than the xTS models, even though the xTS model performs better when composite rules are included on the test set. This shows that more context is beneficial, but that the additional parameters are not estimated accurately enough.

For verb form and subject-verb agreement errors, the xTS model outperforms the FST model on the test set when composite rules are included. On the development set, the xTS model performs worse than the FST model. The recall for verb form and subject-verb agreement errors are given in Table 7.8. For verb form errors, the FST model performs better on the development set, but similar on the test set. For subject-verb agreement, the recall is better for xTS with composite rules than for the FST model. One would expect a syntax-based model to perform better on SVA errors. However, in the constituency parse tree representation used in our model, the subject noun phrase and the predicate verb phrase are in different subtrees. As our model does not include head annotation (as the RTG in Figure 3.5), subject-verb agreement is modelled insufficiently with our model.

Next we briefly compare the error type results to that of participating systems in the CoNLL shared task. Our model ranked second in preposition correction. Interestingly, the top three models for preposition error correction used machine translation or language modelling approaches. The best model obtained an F1 score of 17.53%. For noun number correction our model ranked third; the top two teams obtained F1 scores of 43.26% and 44.25%, respectively. On determiner correction our xTS model ranked 8th. However, our FST model would have ranked first on determiner correction, as the F1 score of the top entry for this error type was 33.4%. On verb form and SVA errors, our system was also placed 8th. The top-scoring team achieved an F1 score of 24.51%. However, our model still outperformed the phrase-based SMT approach, with an F1 of 13.46%.

7.5 FCE results

We also evaluated the performance of our models on the FCE dataset.

Data set	Precision	Recall	F1
Development	13.09	18.52	15.34
Test	14.80	18.38	16.40

Table 7.9: Results for the FST model on FCE.

Error type	Development			Test		
	P	R	F1	P	R	F1
Verb	20.73	15.23	17.56	22.22	15.51	18.27
Noun	10.08	22.14	13.85	11.46	18.73	14.22
Adjective	20.26	21.23	20.73	19.46	14.08	16.33
Adverb	11.58	22.36	15.25	7.91	19.08	11.19
Preposition	13.68	20.74	16.48	12.76	18.80	15.20
Conjunction	8.28	14.46	10.53	7.37	10.94	8.81
Quantifier	33.89	19.44	25.92	41.12	17.07	24.14
Pronoun	6.58	10.30	8.03	6.41	8.57	7.33
Determiner	14.97	23.58	18.32	25.23	27.08	26.12

Table 7.10: Results for the FST model on FCE, for each error type.

Rule set	Minimal			Composite		
Data set	P	R	F1	P	R	F1
Validation	11.14	17.51	13.62	11.70	18.54	14.35
Development	10.50	15.63	12.56	10.97	16.49	13.17
Development Oracle	44.77	42.94	43.84	41.59	43.86	42.70
Test	10.71	15.28	12.59	9.87	14.06	11.6

Table 7.11: Results on FCE for minimal and composite rules.

7.5.1 FST model

The results of the FST model on FCE are given in Table 7.9. A breakdown of results for the different error types considered is given in Table 7.10. The model performance on the development and test sets are very similar for most error types. The overall increase in performance on the test set over the development set is caused mainly by an increase in performance on determiner errors.

7.5.2 Tree transducer models

We also evaluated the xTS models on the FCE corpus. The results are given in Table 7.11. For the minimal rule set, the α that gives the best performance on the validation set is 6.053. This high value shows that the contribution of the transducer model score is relatively small in performing optimal reranking. However, the reranked transducer model still performs slightly better than if only the language model score were used to rerank the hypothesis. When composite rules are also included, the model performs slightly better on the development set, but worse on the test set.

Rule set	Minimal	Composite
Data set	Accuracy	Accuracy
Development Oracle no change	93.3	88.6
Development Oracle change	29.1	19.2
Test Oracle no change	88.5	85.5
Test Oracle change	18.5	16.7

Table 7.12: Oracle clause level results on FCE.

Error type	Development			Test		
	P	R	F1	P	R	F1
Verb	8.92	11.87	10.19	8.02	12.03	9.62
Noun	12.55	20.54	15.58	10.56	17.19	13.08
Adjective	15.84	15.53	15.69	13.33	7.93	9.95
Adverb	6.99	10.48	8.39	8.89	9.63	9.25
Preposition	16.8	20.83	18.6	12.84	17.51	14.81
Conjunction	3.57	3.12	3.33	0.0	0.0	0.0
Quantifier	16.67	6.45	9.30	38.46	17.85	24.39
Pronoun	5.77	8.24	6.79	3.34	5.83	4.29
Determiner	12.03	23.56	15.94	16.38	19.14	17.65

Table 7.13: Results on FCE, for each error type.

Results on the performance at clausal level are given in Table 7.12. On the development set and the test set 57.6% and 55.4% of clauses do not contain any corrections, respectively. The oracle scores are low in comparison to the NUCLE oracle scores. This is mainly due to the fact that we based most of our modelling choices on NUCLE and the more limited set of errors considered for that dataset. Another explanation is that for some of the error types considered here the model does not generalize well enough, due to data sparsity in the training data.

A breakdown of the performance of the model for different error types when only minimal rules are included is given in Table 7.13. For most of the error types considered, the performance is worse than on the FST model. For errors involving prepositions and nouns, the xTS model performs better on the development set. It was for these error types that we also obtained the best relative performance on NUCLE. We expect that with modelling choices tailored more towards the broader set of error types it is possible to improve the model performance on the FCE dataset.

7.6 Conclusion

In this chapter we presented results of empirical evaluations of our proposed models. The baseline FST model was evaluated in Section 7.1. Results regarding language model reranking were given in Section 7.2. Section 7.3 gave results for minimal and composite rules, binarization and related modelling choices. Performance of the xTS models on different error types were presented in Section 7.4. The models were also evaluated on the FCE dataset, in Section 7.5.

Chapter 8

Conclusion

In this thesis we investigated a novel method for performing grammatical error correction with probabilistic tree transducers. We showed how weighted tree-to-string transducers are used as probabilistic models that express syntax-based transformations in natural language, and how grammatical error correction can be formulated as such a transformation. In this chapter we summarize the contributions of our study and discuss several directions for future work.

8.1 Contributions

We highlight the main contributions of our study:

- We proposed a novel application of weighted tree transducers and methods developed originally for syntax-based SMT, to grammatical error correction. In contrast to related SMT-inspired or noisy-channel model approaches, we model the syntax of the correct version of sentences directly.

- We showed that for some error types (especially for article errors) a simple FST model can perform better than some of the more sophisticated approaches in the literature.

- We argued that grammatical error correction can be formulated by linguistically-motivated phrase rewrites. Furthermore, we showed how the GHKM algorithm (which we re-implemented) can be applied to extract such rewrite rules. This can be seen as a kind of automatic feature extraction, taking different levels of context into consideration as necessary.

- We developed an end-to-end NLP system for GEC, based on tree transducers. Our framework can handle different data formats and is modular enough to incorporate different transducer models and decoders. We also gave results regarding the performance of Tiburon.

- We investigated strategies to make the search space of tree transducer models more feasible: Our algorithm to split sentences into clauses may also have applications in other NLP tasks where long sentences need to be processed. Heuristic beam search, pruning and placing restrictions on transducer rules were investigated empirically. A strategy to rerank the tree transducer output with a language model to optimize the F1 score directly on a validation set was also proposed.

- We considered different parameterizations of our tree transducer models, especially regarding the binarization of rules and trees, and taking different levels of context into consideration. Furthermore it was shown that, in most cases, including composite rules increases the model performance.

- We submitted a successful entry for the CoNLL-2013 shared task in grammatical error correction. Overall, our system (Buys and van der Merwe, 2013) ranked 6th out of the 17 participating systems, and was placed in the top three for two of the five error types considered.

8.2 Future work

While we were able to obtain promising results with our models, there are several avenues to improve and extend this research. We suggest the following for future work:

- Our model should be evaluated more systematically against statistical classifiers which use similar features. In particular, this should provide more insight on why our FST model performs so well without using any contextual features other than the information encoded in the n-gram model.

- More sophisticated decoding methods that have been developed for hierarchical phrase-based and syntax-based SMT should be applied to our models. Specifically, the n-gram language model should be integrated into decoding without first obtaining a k-best list of transducer model output. One strategy for language model integration is cube pruning (Chiang, 2007). Implementations of this are publicly available for SCFGs (for example in cdec (Dyer *et al.*, 2010)) but not, as far as we know, for xTS models. In another strategy, HiFST (Gispert *et al.*, 2010), the hypothesis sentences generated by a SCFG are encoded as a FST, which is then composed with a language model. This strategy makes fewer search errors than cube pruning, and can also be extended to xTS decoding, when the transducer does not contain cycles.

- Methods to perform discriminative training using multiple features should be applied to our models. Two methods that have been applied to syntax-based SMT are MERT (Och, 2003) and MIRA (Chiang *et al.*, 2009). These algorithms optimize feature weights to minimize some loss function related to the evaluation metric (for machine translation, the BLEU score). These algorithms can be used in our models to set weights to maximize the F1 score of a validation set. MERT is used for a small number of features (usually not more than 20), while MIRA is scalable to a larger number of features. An example where additional features may be helpful, is in controlling the model preference for performing word insertions and deletions. A feature for the weight that our

FST error model produces may also be useful. The balance between rules that perform changes and those that don't can also be controlled.

- Another method for discriminative training is a large margin framework based on structural SVMs, that has been applied to sentence compression with STSGs (Cohn and Lapata, 2007). In this framework, a loss function is also specified, and sparse features are used.

- Including a RTG language model will increase our system's ability to model syntax accurately. The set of parse trees of the correct side of training data sentences is large enough to enable our transducer model to perform parsing with a reasonable degree of accurately. However, if the tree-based language model and error model components are decomposed as a noisy channel model, the parameterization of both can be improved. Using manually-annotated parse trees, for example from the PTB, as training data for the tree-based language model will also produce a better RTG.

- Grammatical error correction systems need more knowledge sources than the almost purely syntactic model used by our system. Semantics plays an important role in many of the error types we are interested in correcting, so ways to include semantics explicitly in our models should be considered. One possibility is to use dependency parsing, a powerful formalism for expressing syntactic and semantic relations between words. Dependency parse information has previously been included in statistical classifiers for GEC. It would be interesting to investigate using dependency trees instead of constituency trees as the basis of our model. Alternatively, dependency information could be incorporated in the constituency syntax-based transducer formulation.

- Though n-gram models are very useful, our system is presently too dependent on the n-gram model. Good modelling choices should improve the performance of our model without using a n-gram model, so that the n-gram model can function only as an additional source of information regarding local word preferences, not as the primary model to make judgements regarding the hypotheses generated by the transducer model.

- It has been shown that SVM classifiers trained with features based on syntax trees are very successful in judging the grammaticality of sentences (Ferraro *et al.*, 2012). Incorporating such a classifier into our system may also improve performance.

- Lastly, it has been shown that models for the selection task can improve the performance of error correction systems (Dahlmeier and Ng, 2011*b*). Therefore combining a tree transducer model for selection with the current tree transducer model trained only on annotated learner text can further improve the performance of our model, as much more training data is available for the selection task.

8.3 Conclusion

Although there is still much work to be done in order to develop high accuracy GEC systems, we believe that our study makes a contribution towards that goal. Many of the models applicable to the complex structures in natural language have been developed only recently. With the rapid development of more complex machine learning methods enabled by the increasing availability of computational resources, one can only expect that significant progress will be made in the development of suitable models in the near future. Grammatical error correction is a challenging and important problem, and we really hope that more research in NLP will be directed towards this problem. This thesis aspires to be a step in that direction.

Appendix A

Tagsets

Tag	Meaning
Clausal level tags	
S	Simple declarative clause
SBAR	Clause introduced by a (possibly empty) subordinating conjunction
SBARQ	Direct question introduced by a wh-word or a wh-phrase
SINV	Declarative sentence with subject-aux inversion
SQ	Subconstituent of SBARQ excluding the wh-word or wh-phrase
RRC	Reduced relative clause
Phrasal level tags	
VP	Verb phrase
ADVP	Adverb phrase
NP	Noun phrase
NX	Head NP of certain complex NPs
ADJP	Adjective phrase
PP	Prepositional phrase
CONJP	Conjunction phrase
QP	Quantifier phrase
UCP	Unlike Coordinated phrase
WHADJP	Wh-adjective Phrase
WHAVP	Wh-adverb phrase
WHNP	Wh-noun phrase
WHPP	Wh-prepositional phrase
INTJ	Interjection
PRT	Particle
LST	List marker
PRN	Parenthetical
FRAG	Fragment
NAC	Not a constituent
X	Unknown, uncertain, or unbracketable

Table A.1: The Penn Treebank syntactic tagset.

Tag	Meaning
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential *there*
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol (mathematical or scientific)
TO	*to*
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-third person singular present
VBZ	Verb, third person singular present
WDT	*wh*-determiner
WP	*wh*-pronoun
WP$	Possessive *wh*-pronoun
WRB	*wh*-adverb

Table A.2: The Penn Treebank POS tagset.

Appendix B

Data Formats

In this appendix we provide brief descriptions of the formats of the datasets used in our model implementations.

B.1 FCE

The FCE corpus[1] (Yannakoudakis *et al.*, 2011) is annotated in XML format. Each XML file corresponds to the examination script of a learner, and contains 2 short essays of 120 to 180 words each. In our implementation we parse the XML with the Python Minidom parser, a lightweight implementation of the DOM standard for XML processing.

Essays are divided into paragraphs, indicated with $\langle p \rangle$ and $\langle /p \rangle$ tags. The essays are also annotated with metadata, including the essay score and the native language and age of the writer. For each suggested edit to the original, the error type, original incorrect word or phrase and corrected word or phrase is given. The error type tags usually consist of two letters, the first denoting the part-of-speech of the error and the second the kind of change needed to perform the correction. The tags follow the error classifications of Tables 2.1 and 2.2. Here is an example of an error-annotated paragraph:

```
<p>Thanks for <NS type="DD"><i>you</i><c>your</c></NS> letter.
I am so <NS type="RJ"><i>exciting</i><c>excited</c> </NS> that
I have won the first prize. I will give you all <NS type="MD">
<c>the</c></NS> information you need and ask some questions.</p>
```

Error corrections in the annotations can be nested. To handle this we need to parse the XML recursively to extract a flattened correction consisting of an incorrect and a correct phrase. It is possible that a single word can be edited more that once, as in this example:

```
... which caused me
<NS type='FN'> <i> <NS type='RN'> <i>trouble</i>
                                  <c>problem</c> </NS> </i>
               <c>problems</c> </NS>.
```

[1]http://ilexir.co.uk/applications/clc-fce-dataset/

In our processing the intermediate correction is ignored. Therefore the correction extracted from the example is *trouble* → *problems.*

B.2 NUCLE

The other learner corpus used is the National University of Singapore Corpus for Learner English (NUCLE) (Dahlmeier *et al.*, 2013). Version 2 of this corpus was released as part of the CoNLL-2013 shared task. The corpus is provided in two formats, "raw" and "pre-processed".

The raw dataset is annotated in SGML (Standard Generalized Markup Language) format. The corpus consists of documents which are learner essays. Each essay is subdivided into paragraphs. The text in each document is followed by annotations of errors with corrections. The annotations are marked by start and end offsets that are indexes of characters in a specified paragraph. For each error annotation the type of error is also marked, using the error tags described in 2.3.

Below is an example extract from a document annotation:

```
<DOC nid="840">
<TEXT>
<P>
Engineering design process can be defined as a process ...
</P>
<P>
Firstly, engineering design ...
</P>
...
</TEXT>
<ANNOTATION teacher_id="173">
<MISTAKE start_par="0" start_off="0" end_par="0" end_off="26">
<TYPE>ArtOrDet</TYPE>
<CORRECTION>The engineering design process</CORRECTION>
</MISTAKE>
...
</ANNOTATION>
</DOC>
<DOC nid="862">
...
```

In the pre-processed version, the original sentences are formatted in a column format traditionally used for CoNLL shared tasks. Sentence and word tokenization are performed with NLTK *punkt* and *word tokenize*, respectively. The constituency and dependency parses of each sentence are also included. The sentences are parsed using the Stanford parser. Below is an example sentence in CoNLL format:

```
NID   PID  SID TOKENID TOKEN      POS  DPHEAD DPREL   SYNT
850    4    2     0     This       DT     3    nsubj   (ROOT(S(NP*)
850    4    2     1     will       MD     3    aux     (VP*
850    4    2    2      directly    RB    3    advmod  (ADVP*)
850    4    2    3      affects     VB    -1   root    (VP*
850    4    2    4      the         DT    5    det     (NP*
850    4    2    5      process     NN    3    dobj    *)
850    4    2    6      on          IN    3    prep    (PP*
850    4    2    7      engineering NN    8    nn      (NP(NP*
850    4    2    8      design      NN    6    pobj    *)
850    4    2    9      of          IN    8    prep    (PP*
850    4    2    10     innovation  NN    9    pobj    (NP*))))
850    4    2    11     especially  RB    12   advmod  (PP*
850    4    2    12     on          IN    3    prep    *
850    4    2    13     raising     VBG   12   pcomp   (S(VP*
850    4    2    14     solutions   NNS   13   dobj    (NP*))))))
850    4    2    15     .            .    -    -       *))
```

NID is the document identifier, PID the paragraph identifier, SID the paragraph identifier and TOKENID the word token identifier. The POS and SYNT columns give the constituency parse. A fragment of the constituency tree directly preceding the word is given in the SYNT table, and the syntactic parse can be reconstructed from it. The dependency parse is given in the columns DPHEAD and DPREL. DPHEAD indicates the index of the dependency head word of each token, while DPREL is the type of dependency relation between the word and it head.

The error annotations for the preprocessed version are contained in a separate SGML file. The character-level offsets of corrections are projected to token-level annotations. Below are the error annotations for the sentence in the above example:

```
<ANNOTATION>
<MISTAKE nid="850" pid="4" sid="2" start_token="1" end_token="4">
<TYPE>Vt</TYPE>
<CORRECTION>directly affects</CORRECTION>
</MISTAKE>
<MISTAKE nid="850" pid="4" sid="2" start_token="6" end_token="7">
<TYPE>Prep</TYPE>
<CORRECTION>of</CORRECTION>
</MISTAKE>
<MISTAKE nid="850" pid="4" sid="2" start_token="7" end_token="11">
<TYPE>WOinc</TYPE>
<CORRECTION>innovative engineering design</CORRECTION>
</MISTAKE>
<MISTAKE nid="850" pid="4" sid="2" start_token="12" end_token="13">
<TYPE>Wci</TYPE>
<CORRECTION>in</CORRECTION>
</MISTAKE>
<MISTAKE nid="850" pid="4" sid="2" start_token="13" end_token="15">
<TYPE>Um</TYPE>
<CORRECTION></CORRECTION>
```

```
</MISTAKE>
</ANNOTATION>
```

The SGML files are parsed with the Python ElementTree XML parser. By adding a root node to a SGML construction, it can be parsed as XML.

B.3 Parse trees

The format of parse trees produced by the Stanford or Berkeley parsers is different from the format of trees used in this thesis. The format of the parser output of a tree t with root node σ is (recursively) $t = (\sigma\ t_1\ t_2 \ldots t_n)$. Below is the parser output representation of the parse tree in Figure 2.1:

> (S (NP (DT the) (NN man) (VP (VBZ helps) (NP (DT the) (NNS children)))))

B.4 M^2 scorer

We use the M^2 scorer[2] (Dahlmeier and Ng, 2012*b*) to evaluate our system output. The system output is accepted as one sentence per line, tokenized, and with the original capitalization. The gold standard file contains the original sentences, together with the gold standard edits of each sentence. The format for an error annotation is:

> A <token start offset> <token end offset>|||<error type>|||<correction1>|| <correction2>||...correctionN|||<required>|||<comment>|||<annotator id>

Below is an example sentence annotation:

```
S This will directly affects the process on engineering design of innovation
especially on raising solutions .
A 1 4|||Vt|||directly affects|||REQUIRED|||-NONE-|||0
A 6 7|||Prep|||of|||REQUIRED|||-NONE-|||0
A 7 11|||WOinc|||innovative engineering design||||REQUIRED|||-NONE-|||0
A 12 13|||Wci|||in|||REQUIRED|||-NONE-|||0
```

[2]http://www.comp.nus.edu.sg/~nlp/software.html

Bibliography

Allauzen, C., Riley, M., Schalkwyk, J., Skut, W. and Mohri, M. (2007). OpenFst: A general and efficient weighted finite-state transducer library. In: *CIAA Revised Selected Papers*, Lecture Notes in Computer Science, pp. 11–23. Springer.

Atwell, E.S. (1987). How to detect grammatical errors in a text without parsing it. In: *Proceedings of EACL*, pp. 38–45.

Bergsma, S., Lin, D. and Goebel, R. (2009). Web-scale *n*-gram models for lexical dismabiguation. In: *Proceedings of IJCAI*, pp. 1507–1512.

Bies, A., Ferguson, M., Katz, K. and MacIntyre, R. (1995). *Bracketing Guidelines for Treebank II style Penn Treebank Project.*

Bird, S., Klein, E. and Loper, E. (2009). *Natural Language Processing with Python.* O'Reilly Media.

Brockett, C., Dolan, W.B. and Gamon, M. (2006). Correcting ESL errors using phrasal SMT techniques. In: *Proceedings of ACL*, pp. 249–256.

Brown, P.F., Pietra, S.D., Pietra, V.J.D. and Mercer, R.L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, vol. 19, no. 2, pp. 263–311.

Büchse, M., May, J. and Vogler, H. (2010). Determinization of weighted tree automata using factorizations. *Journal of Automata, Languages and Combinatorics*, vol. 15, no. 3, pp. 229–254.

Buys, J. and van der Merwe, B. (2013). A tree transducer model for grammatical error correction. In: *Proceedings of CoNLL: Shared task*, pp. 43–51.

Charniak, E. and Johnson, M. (2005). Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In: *Proceedings of ACL*, pp. 173–180.

Chen, S.F. and Goodman, J. (1999). An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, vol. 13, no. 4, pp. 359–393.

Chiang, D. (2007). Hierarchical phrase-based translation. *Computational Linguistics*, vol. 33, no. 2, pp. 201–228.

Chiang, D., Graehl, J., Knight, K., Pauls, A. and Ravi, S. (2010). Bayesian inference for finite-state transducers. In: *Proceedings of HLT-NAACL*, pp. 447–455.

Chiang, D., Knight, K. and Wang, W. (2009). 11,001 new features for statistical machine translation. In: *Proceedings of HLT-NAACL*, pp. 218–226.

Chomsky, N. (1956). Three models for the description of language. *IEEE Transactions of Information Theory*, vol. 2, no. 3, pp. 113–124.

Chomsky, N. (1957). *Syntactic Structures*. Mouton.

Chomsky, N. (1959). On certain formal properties of grammars. *Information and Control*, vol. 2, pp. 137–167.

Chung, T., Fang, L. and Gildea, D. (2011). Issues concerning decoding with synchronous context-free grammar. In: *Proceedings of ACL (Short Papers)*, pp. 413–417.

Cohn, T. and Lapata, M. (2007). Large margin synchronous generation and its application to sentence compression. In: *Proceedings of EMNLP-CoNLL*, pp. 73–82.

Collins, M. (1999). *Head-driven Statistical models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S. and Tommasi, M. (2007). Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`. Release October, 12th 2007.

Dahlmeier, D. and Ng, H.T. (2011*a*). Correcting semantic collocation errors with L1-induced paraphrases. In: *Proceedings of EMNLP*, pp. 107–117.

Dahlmeier, D. and Ng, H.T. (2011*b*). Grammatical error correction with alternating structure optimization. In: *Proceedings of ACL*, pp. 915–923.

Dahlmeier, D. and Ng, H.T. (2012*a*). A beam-search decoder for grammatical error correction. In: *Proceedings of EMNLP-CoNLL*, pp. 568–578.

Dahlmeier, D. and Ng, H.T. (2012*b*). Better evaluation for grammatical error correction. In: *Proceedings of HTL-NAACL*, pp. 568–572.

Dahlmeier, D., Ng, H.T. and Ng, E.J.F. (2012). NUS at the HOO 2012 shared task. In: *Proceedings of the 7th Workshop on Innovative Use of NLP for Building Educational Applications*, pp. 216–224.

Dahlmeier, D., Ng, H.T. and Wu, S.M. (2013). Building a large annotated corpus of learner English: The NUS corpus of learner English. In: *Proceedings of the 8th Workshop on Innovative Use of NLP for Building Educational Applications*, pp. 22–31.

Dale, R., Anisimoff, I. and Narroway, G. (2012). HOO 2012: A report on the preposition and determiner error correction shared task. In: *Proceedings of the 7th Workshop on Innovative Use of NLP for Building Educational Applications*, pp. 54–62.

Dale, R. and Kilgarriff, A. (2010). Helping our own: Text massaging for computational linguistics as a new shared task. In: *Proceedings of the Sixth International Natural Language Generation Conference (INLG)*.

Dale, R. and Kilgarriff, A. (2011). Helping our own: The HOO 2011 pilot shared task. In: *Proceedings of the 13th European Workshop on Natural Language Generation*, pp. 242–249.

De Felice, R. and Pulman, S.G. (2008). A classifier-based approach to preposition and determiner error correction in L2 English. In: *Proceedings of COLING*, pp. 169–176. Association for Computational Linguistics.

De Marneffe, M.-C., MacCartney, B. and Manning, C.D. (2006). Generating typed dependency parses from phrase structure parses. In: *Proceedings of LREC*, vol. 6, pp. 449–454.

Dempster, A.P., Laird, N.M. and Rubin, D.B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, vol. 39, no. 1, pp. 1–38.

DeNeefe, S., Knight, K., Wang, W. and Marcu, D. (2007). What can syntax-based MT learn from phrase-based MT? In: *Proceedings of EMNLP-CoNLL*, pp. 755–763.

Droste, M. and Gastin, P. (2009). Weighted automata and weighted logics. In: Droste *et al.* (2009), chap. 5.

Droste, M., Kuich, W. and Vogler, H. (eds.) (2009). *Handbook of Weighted Automata*. Monographs in Theoretical Computer Science. Springer-Verlag, Berlin Heidelberg.

Dyer, C., Lopez, A., Ganitkevitch, J., Weese, J., Türe, F., Blunsom, P., Setiawan, H., Eidelman, V. and Resnik, P. (2010). cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In: *Proceedings of ACL (System Demonstrations)*, pp. 7–12.

Ehsan, N. and Faili, H. (2013). Grammatical and context-sensitive error correction using a statistical machine translation framework. *Software: Practice and Experience*, vol. 42, no. 2, pp. 187–206.

Eisenstein, J. (2013). What to do about bad language on the internet. In: *Proceedings of HTL-NAACL*, pp. 359–369.

Eisner, J. (2002). Parameter estimation for probabilistic finite-state transducers. In: *Proceedings of ACL*, pp. 1–8.

Eisner, J. (2003). Learning non-isomorphic tree mappings for machine translation. In: *Proceedings of ACL (Companion)*, pp. 205–208.

Engelfriet, J. (1975). Tree automata and tree grammars. Unpublished Lecture Notes. Department of Computer Science, University of Aarhus.

Ferraro, F., Post, M. and Van Durme, B. (2012). Judging grammaticality with count-induced tree substitution grammars. In: *Proceedings of the Seventh Workshop on Innovative Use of NLP for Building Educational Applications*, pp. 116–121.

Foster, J. and Andersen, O. (2009). GenERRate: Generating errors for use in grammatical error detection. In: *Proceedings of the Fourth Workshop on Innovative Use of NLP for Building Educational Applications*, pp. 82–90.

Fülöp, Z. and Vogler, H. (2009). Weighted tree automata and tree transducers. In: Droste *et al.* (2009), chap. 9.

Galley, M., Graehl, J., Knight, K., Marcu, D., DeNeefe, S., Wang, W. and Thayer, I. (2006). Scalable inference and training of context-rich syntactic translation models. In: *Proceedings of ACL*, pp. 961–968.

Galley, M., Hopkins, M., Knight, K. and Marcu, D. (2004). What's in a translation rule? In: *Proceedings of HLT-NAACL*, pp. 273–280.

Gamon, M. (2010). Using mostly native data to correct errors in learners' writing: A meta-classifier approach. In: *Proceedings of HLT-NAACL*, pp. 163–171.

Gamon, M., Gao, J., Brockett, C., Klementiev, A., Dolan, W.B., Belenko, D. and Vanderwende, L. (2008). Using contextual speller techniques and language modeling for ESL error correction. In: *Proceedings of IJCNLP*.

Gécseg, F. and Steinby, M. (1984). *Tree Automata.* Akadéniai Kiadó, Budapest, Hungary.

Gispert, A.D., Iglesias, G., Blackwood, G.W., Banga, E.R. and Byrne, W. (2010). Hierarchical phrase-based translation with weighted finite-state transducers and shallow-n grammars. *Computational Linguistics*, vol. 36, no. 3, pp. 505–533.

Graehl, J., Knight, K. and May, J. (2008). Training tree transducers. *Computational Linguistics*, vol. 34, no. 3, pp. 391–427.

Han, N., Tetreault, J., Lee, S. and Ha, J. (2010). Using an error-annotated learner corpus to develop an ESL/EFL error correction system. In: *Proceedings of LREC*.

Hopkins, M. and Langmead, G. (2010). SCFG decoding without binarization. In: *Proceedings of EMNLP*, pp. 646–655.

Huang, L. and Chiang, D. (2005). Better k-best parsing. In: *Proceedings of the Ninth International Workshop on Parsing Technologies*, pp. 53–64.

Huddleston, R. and Pullum, G. (eds.) (2002). *The Cambridge Grammar of English.* Cambridge: Cambridge University Press.

Huddleston, R. and Pullum, G. (2005). *A Student's Introduction to English Grammar.* Cambridge University Press.

Jones, B.K., Johnson, M. and Goldwater, S. (2012). Semantic parsing with Bayesian tree transducers. In: *Proceedings of ACL*, pp. 488–496.

Jurafsky, D. and Martin, J.H. (2009). *Speech and Language Processing.* 2nd edn. Pearson Education.

Katz, S.M. (1987). Estimation of probabilities from sparse data for the language model of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 3, pp. 400–401.

Klein, D. and Manning, C.D. (2003). Accurate unlexicalized parsing. In: *Proceedings of ACL*, pp. 423–430.

Knight, K. (2007). Capturing practical natural language transformations. *Machine Translation*, vol. 21, no. 2, pp. 121–133.

Knight, K. and Al-Onaizan, Y. (1998). Translation with finite-state devices. In: *Proceedings of the Third Conference of the Association for Machine Translation in the Americas*, pp. 421–437.

Knight, K. and Chander, I. (1994). Automated postediting of documents. In: *Proceedings of AAAI*, pp. 779–784.

Koehn, P., Och, F.J. and Marcu, D. (2003). Statistical phrase-based translation. In: *Proceedings of HLT-NAACL*, pp. 48–54.

Leacock, C., Chodorow, M., Gamon, M. and Tetreault, J.R. (2010). *Automated Grammatical Error Detection for Language Learners.* Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

Lee, J. and Seneff, S. (2006). Automatic grammar correction for second-language learners. In: *Proceedings of Interspeech*, pp. 1978–1981.

MacDonald, N.H., Frase, L.T., Gingrich, P.S. and Keenan, S.A. (1982). The writer's workbench: Computer aids for text analysis. *IEEE Transactions on Communications*, vol. 30, no. 1.

Madnani, N., Tetreault, J. and Chodorow, M. (2012). Exploring grammatical error correction with not-so-crummy machine translation. In: *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*, pp. 44–53.

Maletti, A. (2006). *The Power of Tree Series Transducers.* Ph.D. thesis, Technischen Universität Dresden.

Maletti, A., Graehl, J., Hopkins, M. and Knight, K. (2009). The power of extended top-down tree transducers. *SIAM Journal for Computation*, vol. 39, no. 2, pp. 410–430.

Marcu, D., Wang, W. and Knight, K. (2007). Binarizing syntax trees to improve syntax-based machine translation accuracy. In: *Proceedings of EMNLP-CoNLL*, pp. 746–754.

Marcus, M.P., Santorini, B. and Marcinkiewicz, M.A. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, vol. 19, no. 2, pp. 313–330.

May, J. (2010). *Weighted Tree Automata and Transducers for Syntactic Natural Language Processing.* Ph.D. thesis, University of Southern California.

May, J. and Knight, K. (2006). Tiburon: A weighted tree automata toolkit. In: *Proceedings of CIAA*, pp. 102–113.

Miller, G.A. (1995). WordNet: A lexical database for English. *Communications of the ACM*, vol. 38, no. 11, pp. 39–41. ISSN 0001-0782.

Mohri, M. (2009). Weighted automata algorithms. In: Droste *et al.* (2009), chap. 6.

Mohri, M., Pereira, F. and Riley, M. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, vol. 16, no. 1, pp. 6–88.

Murphy, K.P. (2012). *Machine Learning: A Probabilistic Perspective.* MIT.

Ng, H.T., Wu, S.M., Wu, Y., Hadiwinoto, C. and Tetreault, J. (2013). The CoNLL-2013 shared task on grammatical error correction. In: *Proceedings of CoNLL: Shared task*, pp. 1–12.

Nicholls, D. (2003). The Cambridge learner corpus: Error coding and analysis for lexicography and ELT. In: *Proceedings of the Corpus Linguistics 2003 Conference*, pp. 572–581.

Novak, J.R., Minematsu, N. and Hirose, K. (2011). Open-source WFST tools for LVCSR cascade construction. In: *Proceedings of FSMNLP*, pp. 65–73.

Och, F.J. (2003). Minimum error rate training in statistical machine translation. In: *Proceedings of ACL*, pp. 160–167.

Park, Y.A. and Levy, R. (2011). Automated whole sentence grammar correction using a noisy channel model. In: *Proceedings of ACL*, pp. 934–944.

Petrov, S., Barrett, L., Thibaux, R. and Klein, D. (2006). Learning accurate, compact, and interpretable tree annotation. In: *Proceedings of COLING-ACL*, pp. 433–440.

Petrov, S. and Klein, D. (2007). Improved inference for unlexicalized parsing. In: *Proceedings of HLT-NAACL*, pp. 404–411.

Pirinen, T. and Lindén, K. (2010). Finite-state spell-checking with weighted language and error models – building and evaluating spell-checkers with Wikipedia as corpus. *Proceedings of the LREC Workshop on Creation and use of basic lexical resources for less-resourced languages.*

Rabiner, L.R. and Juang, B.H. (1986). An introduction to hidden Markov models. *IEEE ASSP Magazine*, pp. 4–16.

Rounds, W.C. (1970). Mappings and grammars on trees. *Mathematical systems theory*, vol. 4, no. 3, pp. 257–287.

Rozovskaya, A. and Roth, D. (2011). Algorithm selection and model adaptation for ESL correction tasks. In: *Proceedings of ACL*, pp. 924–933.

Santorini, B. (1990). *Part-of-Speech Tagging Guidelines for the Penn Treebank Project.*

Shannon, C.E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423.

Sipser, M. (2006). *Introduction to the Theory of Computation.* 2nd edn. Thomson Course Technology.

Stolcke, A. (2002). SRILM: An extensible language modeling toolkit. In: *Proceedings of the 7th International Conference on Spoken Language Processing (ICSLP)*, pp. 901–904.

Thatcher, J.W. (1970). Generalized sequential machine maps. *Journal of Computer System Science*, vol. 4, no. 4, pp. 339–367.

Toutanova, K., Klein, D., Manning, C.D. and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In: *Proceedings of HLT-NAACL*, pp. 173–180.

Turner, J. and Charniak, E. (2007). Language modeling for determiner selection. In: *Proceedings of HTL-NAACL*, pp. 177–180.

Wagner, J., Foster, J. and van Genabith, J. (2007). A comparative evaluation of deep and shallow approaches to the automatic detection of common grammatical errors. In: *EMNLP-CoNLL*, pp. 112–121.

Yamada, K. and Knight, K. (2001). A syntax-based statistical translation model. In: *Proceedings of ACL*, pp. 523–530.

Yannakoudakis, H., Briscoe, T. and Medlock, B. (2011). A new dataset and method for automatically grading ESOL texts. In: *Proceedings of ACL*, pp. 180–189.

Zhang, H., Huang, L., Gildea, D. and Knight, K. (2006). Synchronous binarization for machine translation. In: *Proceedings of HLT-NAACL.*