Models of Grammar Evolution: Evolving English

Markus Gerstel
Lincoln College

Supervisors: Dr Stephen Clark
COMPUTING LABORATORY

Prof. Jotun Hein
DEPARTMENT OF STATISTICS

Thesis submitted in partial fulfilment of the requirements for the degree of MSc. in Computer Science

September 2008

I would like to thank in particular my supervisors Stephen Clark and Jotun Hein as well as Rune Lyngsø and Ferenc Huszár and everyone at our presentation sessions for many insightful and helpful comments

This work was supported by stipend D/06/48365 from the German Academic Exchange Service, Bonn, Germany.

Table of Contents

Chapter 1

Introduction

> *"(...) the best grounding for education is the Latin grammar. I say this, not because Latin is traditional and medieval, but simply because even a rudimentary knowledge of Latin cuts down the labour and pains of learning almost any other subject by at least 50 per cent. It is the key to the vocabulary and structure of all the Romance languages and to the structure of all the Teutonic languages (...)"*
> *— Dorothy L. Sayers, The Lost Tools of Learning (1947), Oxford*

1.1. Cross-Language Structure

Anyone who ever considered learning a new language may have come across this or a similar quote. It is quite common to find foreign words incorporated in a language. A lot of German words can be found in the English language (*Lied*, *Bremsstrahlung*,...) and vice versa (*label*, *team*,...). Some of these words are loanwords, borrowed via cultural exchange. Others have a common ancestry, for example hand/Hand. Some words take more complicated ways: '*shop*' in German is borrowed from the English 'shop' which has a common ancestor with the German '*Schuppen*'. Tracing the history of these words is very easy if one has an etymological dictionary. Even more visible than words are borrowed letters or digits, like Umlauts or the Arabic numerals.

However, once it comes to grammatical constructions it gets more difficult. A bilingual speaker will usually only notice the cases in which grammatical constructions differ between languages when thinking up a sentence in his native tongue and translating this sentence word for word into a foreign language, resulting in an awkward or ungrammatical sentence. And even when sentence constructions differ it is not always obvious how exactly the sentence structure has to be changed during translation. On the other hand a fluent speaker might notice and appreciate sentence structures shared between both languages.

The following examples illustrate such a structural relation:

	Ich	habe	dieses	Buch	gelesen.	(German)
	Ik	heb	dit	boek	gelezen.	(Dutch)
(*)[1]	I	have	this	book	read.	(English)

All three sentences express the same fact. They are word for word translations of each other in three West Germanic languages. The German and Dutch sentences are grammatically correct, and if you ever spoke to someone from these countries you may have heard something close to the (ungrammatical) English sentence. Correcting that sentence is easy: Moving the word '*read*' in front of '*this book*' solves the problem. But how would one know which word to pick and where to put it?

1 In linguistics an asterisk in front of a phrase indicates that the following phrase is ungrammatical.

Every sentence has internal syntactic structure, which can be described by a syntax tree. In a syntax tree the words of the sentence can be found at the leaf nodes. The nodes directly connected to leaf nodes are annotated with so-called part of speech tags (POS tags) that describe the function the corresponding word takes in a sentence. Inner nodes, or branch nodes, indicate the function of the phrase beneath. For example in the noun phrase (*NP*) '*this book*' the word '*this*' is a determiner (*DT*) and book is a singular noun (*NN*). The root '*S*' of each tree describes the clause in its entirety as a sentence. The required change between the German and English sentence can now be pinpointed easily by comparing the two syntax trees:

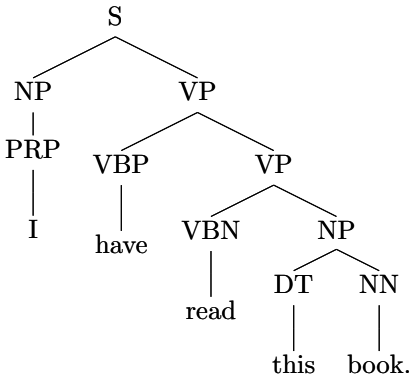

Fig. 1 – Syntax tree for the correct English sentence

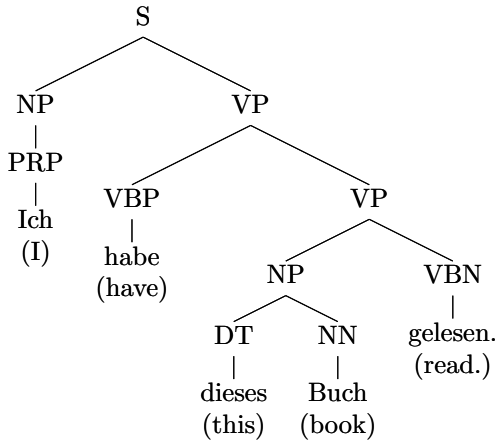

Fig. 2 – Syntax tree for the correct German and incorrect English sentence

Apart from the words both trees differ only in the ordering of the subtrees of the second, lower verb phrase (*VP*) node. A full description of the entire syntax tree annotation can be found in the appendix, section 7.1.

Syntax trees can also be understood as derivation trees: Beginning with a starting symbol, in these cases '*S*' the tree is expanded by derivation rules (e.g. S → NP VP,...) until at the leaves only terminal symbols (words) remain. The tuple of a starting symbol and sets of non-terminals, terminals and these derivation rules constitutes a Context-Free Grammar (CFG). Counting the occurrences of each derivation rule and assigning it a probability results in a Probabilistic Context-Free Grammar.

1.2. Models of Grammar Evolution – Report Structure

This project aims to create a novel evolutionary model for Probabilistic Context-Free Grammars (PCFG) of natural languages.

Chapter 2 will give a more detailed account of the linguistic and computational motivation behind this project. The question of whether such a CFG model can give an accurate representation of the English language, or in fact any other natural language, will be explored.

In chapter 3 the necessary formal techniques will be introduced. Beginning with the formal definition of CFG/PCFGs and an introduction to treebanks as sources for these grammars, this chapter will also reveal weaknesses and limitations of these CFG models, especially in comparative situations. A way around these limitations will be presented with the introduction of n-gram models. A possible distance metric between two languages will be derived and finally an overview of the entire evolutionary model will be provided.

In chapter 4 practical results will be presented. The proposed distance metric will be tested on four language models derived from English, German and Chinese treebanks. The English language will then be subjected to the directed evolutionary model to produce Modified English, a different language. The idea will be to change the English language model in such a way as to move it closer to the German language. Finally the changes made to the English language model will be analysed and the results interpreted.

Chapter 5 will summarize the main findings of this project. It will also give suggestions for continuation projects.

Chapter 2

Background

"What's past is prologue."
— Antonio in William Shakespeare, The Tempest (1611), Act II, scene I

2.1. Linguistic Motivation

Germanic languages like English and German both evolved from a hypothetical common ancestor, Proto-Germanic. In contrast French, Latin, Welsh and Farsi (Persian) are not Germanic languages. All these languages however do have a common root in the (again hypothetical) Proto-Indo-European language. According to linguistic theory this unified Indo-European language existed somewhere between 10,000 BC and 3,000 BC. There is no direct evidence for this language, and not much is known about how it sounded or how it could have looked like in writing.[2]

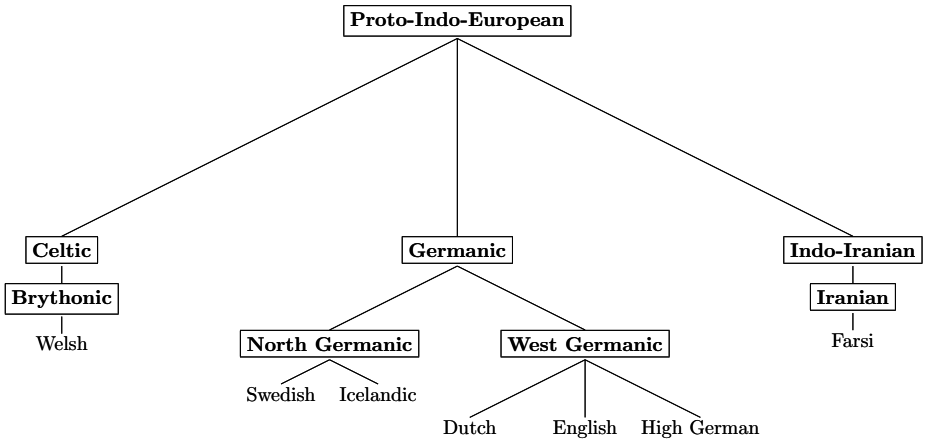

Fig. 3 – Part of the Proto-Indo-European language family tree

2.1.1. Retracing Language Evolution

The two main methods by which knowledge about the Proto-Indo-European language was acquired are called internal reconstruction and the comparative method.

The internal reconstruction method looks at the current state of variation within a language and postulates that exceptions to a general case are newer than the general case. For example in Latin the regular perfect forms of verbs are created by the verb's stem and the added suffix '*si*'. But there are some exceptions to that rule, namely irregular verbs like '*scribere*'. The internal reconstruction method postulates the original verb forms (*)'*figsi*' and (*)'*scribsi*'. It is

carpo	carpere	car**psi**
duco	ducere	du**xi**
figo	figere	fi**xi**, not (∗)fi**gsi**
scribo	scribere	scri**psi**, not (∗)scri**bsi**

Fig. 4 – Latin verb forms

2 The earliest evidence for something that could be considered writing emerged around 6,600 BC. This however was not writing with linguistic content – and this happened in ancient China, not Europe. It may be possible that this Proto-Indo-European language existed but has never been used in writing. There have been two notable attempts to create a text in reconstruted Proto-Indo-European: Schleicher's fable originally by A. Schleicher (1868) and "The king and the god" by S. K. Sen, E. P. Hamp et al. (1994).

then tried to reconstruct the exceptions from their original forms. In this case the postulated change is that devoicing occurs in front of voiceless consonants. This reconstruction has to satisfy two plausibility principles: Firstly, the changes have to be natural, in other words the required sound changes have to be regular, simple and language-universal. Secondly, the postulate has to satisfy Occam's Razor, that there can not be a different postulate that has the same outcome but makes less assumptions (i.e. is simpler).[3] Hypothetical languages created by means of internal reconstruction carry the prefix '*pre-*' (e.g. Pre-Germanic).

The comparative method instead compares variation between two or more languages. If a systematic correspondence of some feature can be established then a hypothetical, common ancestral form can be established. The hypothetical common ancestor of the sound correspondence example in figure 5 could either be '*d*', '*t*' or an unobserved element '*X*'. Again the plausibility principles are applied: A change d → t is quite uncommon, but the change t → d can be observed quite frequently in other languages. A third element '*X*' is not required and would violate Occam's Razor. Hypothetical languages created by means of the comparative method carry the prefix '*proto-*' (e.g. Proto-Indo-European).

day	**d**ies
deus	**d**ivine
diabolus	**d**evil
ten	**d**ecem, not (∗)**t**ecem
two	**d**uo, not (∗)**t**uo

Fig. 5 – Sound correspondences

However, since there is not much information or evidence – certainly not enough for any kind of statistical approach – for any of the Proto-languages the goal of the project will not be to create an accurate model of historic language evolution. While the immediate focus of this thesis lies on grammar evolution, the generic evolution model presented in the following chapters can at a later stage be modified to accommodate accurate historic grammar evolution. A prerequisite for that is the availability of treebanks that contain information about historic grammar evolution, which is currently not the case.

2.1.2. Modelling English Grammar

Before attempting to describe how something changes it is usually desirable to know the current state of affairs. With natural languages this may be more difficult than it seems. Arguably it is impossible to give a complete and accurate grammar for the English language. This is caused by two separate problems:

The first problem lies in the definition of the English language itself. What should be considered 'English'? The difference between American English and British English are just the tip of the iceberg.

3 Example and explanation from A. Lohöfer, UE History of English, Linguistic Reconstruction, Session 2, April 2007

There are differences in the dialects of English spoken in Yorkshire and London. Maybe language should be defined, prescribed instead of described, by some authority that would itself rely only on highest quality works from selected writers. William Shakespeare's works are certainly considered well-formed English – but times have changed, and so has the language. The different settings of place, time and even social hierarchy result in different definitions of English.

The second problem is the inherent richness of language. This can be illustrated by the simple question "How many words are there in the English language?" While it is a myth that the Inuit have 400 different words for snow it is certainly possible to give 400 English compounds with snow (snowflake, snowmobile, snowman, snowball,...). If one considers technical terms of any sort, especially chemical terms, then it is usually possible to construct an infinite amount of compound words. The other extreme would be to only consider words found in a certain dictionary. An approach like this fails of course to keep up with new word creations, for example 'words' like blog, vlog, mlog, blogosphere and blawg. Even when one analyses a copy of the entire Wikipedia with over 2.5 million articles and keeps a list of seen words – that list will always continue to grow.

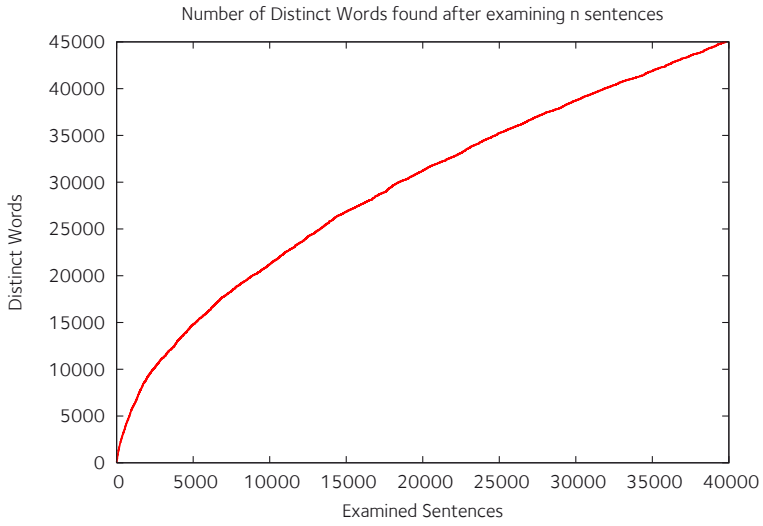

Fig. 6 – Increase in the number of seen distinct English words while analysing the Penn Treebank

Noam Chomsky (1957) argued that English is so complex that it cannot be described by any context-free grammar, probabilistic or not. While there is proof that this property holds for some languages like Swiss German, the question of whether this is true for English is still open (e.g. M. Mohri, R. Sproat, 2006).

Working with a statistical method on something organic and infinitely complex as language clearly requires more pragmatism than perfectionism. The grammar for the English language in this project will be derived from the Penn Treebank, an annotated corpus built from articles in the Wall Street Journal. It will neither be a 100% complete nor a 100% accurate Probabilistic Context-Free Grammar for the English language.

2.2. Existing Language Evolution Simulations

Computer simulation has been used for very early stages of language evolution: The emergence of simple signalling systems allowing shared communication, the appearance of syntax and syntactic universals. The question of a required minimum complexity for any natural language has been studied as well as the question how a language can be learned at all. Simulation on the origins of language usually involves an agent-based view, in which independent entities form a society and interact with and learn from each other. These can be represented by sets of rules or neural networks. Evolution then can be introduced by e.g. genetic algorithms.

Most of these simulations however involve very abstract systems that bear no strong resemblance to any existing language (A. Cangelosi and D. Parisi, 2002; H. Turner, 2002). Others are much closer to an existing language: (M. Hare and J. L. Elman, 1995) applied a machine learning technique to the morphology of English verbs in an effort to simulate evolution on a word level.

It appears that so far there is no existing research on syntax level evolutionary simulation on one or more natural languages.

2.3. Project Aims

A pure linguistic approach to estimate distance between languages usually relies on one or more of these three methods:

The first method is word analysis. The Latin pater became the English father and German Vater. This kind of analysis relies on known rules of how spelling and pronunciation changed over time. Word analysis becomes more difficult the further apart two languages are: The English '*sister*' descends from the same root as the Hindi '*bahan*', but this is not immediately obvious (via Sanskrit '*svasar*', Old English '*sweostor*', cf. A. Lohöfer, 2007).

The second method is estimating the distance by looking at the language family. The language family tree shown on page 9 makes it quite clear that English is much closer related to German than to Welsh. Of course this estimation technique is limited. Nothing can be said about whether Dutch or English is closer to Swedish. This approach also requires a direct connection between two languages.

The final method is the comparisons of parameters in the given languages. These parameters can for example be the number of genders or the number of cases occurring in a language, as well

as more complex properties like word ordering. (Ryder, 2006) attempts to recreate language trees by applying concepts from molecular biology to a set of 109 parameters per language. With these approaches it is still quite difficult to decide whether English is closer to Welsh or to Farsi.

This project enables a novel statistical, semi-automatic approach that will work even between languages that are not connected by the language tree (e.g. English to Chinese). It will also allow to measure an effective distance between two languages: If a language evolved independently from English but had the same grammatical structure the reported difference between these languages would be very small. Later analysis would then show how to manipulate one language to get closer to grammatical structures of the other. A model to match grammars between languages may give practical results that can then be reused in other fields working with language: The most obvious application to benefit from a set of fixed rewriting rules would be statistical machine translation that could incorporate these rules in its model.

Chapter 3

Formalities

"Colorless green ideas sleep furiously."
— Noam Chomsky, Syntactic Structures (1957).

3.1. Modelling a Probabilistic Context-Free Grammar

3.1.1. Context-Free Grammars

A context-free grammar (CFG) is usually defined as a 4-tuple (N,T,Σ,R) with *N* being a set of nonterminal symbols, *T* a set of terminal symbols, Σ a starting symbol that is also an element of *N*, and *R* a (finite) set of derivation rules. These derivation rules have of the form X → Y where *X* is an element of *N*, and *Y* is any combination of elements from *N* and *T*.

$$G_{CFG} := (N, T, \Sigma, R)$$
$$N \cap T = \emptyset,\ \Sigma \in N,$$
$$(X \rightarrow Y) \in R \Rightarrow$$
$$X \in N$$
$$\wedge\, Y \in (N \cup T)^*$$

Fig. 7 – CFG definition

The language corresponding to the grammar is then defined as the (possibly infinite) set of words that can be generated by this grammar. A word is a string that contains only terminal symbols. In the context of this project the set of terminal symbols will be the set of parts of speech (POS) tags. In the resulting context-free language each word, a stream of terminal symbols, will therefore correspond to an entire phrase in the natural language.

The set of nonterminal symbols will be the set of all phrase level and sentence level annotations and a dedicated starting symbol Σ. The use of a dedicated starting symbol together with rules like Σ → S is required, since not every allowed phrase is a sentence. For example headlines or titles ("*Models of Grammar Evolution*") are considered phrases, but not sentences.

3.1.2. Probabilistic Context-Free Grammars

In a probabilistic context-free grammar there is a probability P(Y|X) defined for every rule X → Y. The probabilities of all rules with the same left hand side *X* will always sum up to 1. In a generative view this corresponds to the probability that one of the possible derivation rules is chosen. This also means that it is now possible to calculate the probability of a specific derivation tree by multiplying the probabilities of all *n* involved derivation rules: $P(\mathrm{DerivationTree}) = \prod_{i=1}^{n} P(\mathrm{Y}_i|\mathrm{X}_i)$. To account for ambiguous trees the probability for a specific phrase is then defined to be the sum of the probabilities for all distinct derivation trees leading to that phrase. Since the grammars in this project will be derived from treebanks the probabilities will be calculated by taking counts and using a simple maximum likelihood estimation: $P(\mathrm{Y}|\mathrm{X}) = \frac{\#(\mathrm{X}\rightarrow\mathrm{Y})}{\#(\mathrm{X})}$.

Probabilistic Context-Free Grammars are also used in a range of fields outside of Computer Science and Linguistics, for example RNA structures are routinely described by PCFGs (cf. D. Searls, 1993).

3.1.3. Limitations of Natural Language CFGs/PCFGs

A pure context-free grammar as described so far is very simply structured. Natural languages however can be very intricate and sometimes even appear to defy logic. It therefore will lead to a lot of cases where the grammar diverges from real language usage.

This is most obvious if a PCFG/CFG is used to create sentences consisting of normal words. There is for example no notion of agreement. This means noun phrases like (*)"*one dogs*", (*)"*two dog*" could be generated. One way to deal with this kind of problem is to encode more information into the set of nonterminal symbols: The original rule NP → CD NP where a noun phrase derives to a number plus a noun phrase could be split into NP → CD-singular NP-singular and NP → CD-plural NP-plural.

This means that basically the number rules involving noun phrases doubles. Verbs also may change depending on the number of the subject, so some verb rules will have to be split as well. Since the information from the nouns will have to get 'transported' to the verb, rules like S → NP VP will have to be duplicated. Assume the singular/plural problem is completely solved – the phrase (*)"*much dogs*" could still be generated. So the process starts again duplicating rules for countable and uncountable phrases, etc. Finally there are expressions that are just not idiomatic in a language. "*Merry Christmas!*" is fine, (*)"*Merry Birthday!*" is not. But both "*Happy Christmas!*" and "*Happy Birthday!*" are acceptable. Encoding all this information into a context-free grammar will lead to an incredibly complex model.

Since in this project the PCFG generates POS tags instead of actual words it is not affected by some of these problems. There is for example no notion of countability in POS tags, therefore there can not be any disagreement. Number agreement however does exist for example between third person singular verbs ('*VBZ*') and singular nouns ('*NN*'/'*NNP*').

Another problem is the possibility of repeated derivation. The perfectly fine rule NP → DT NP in "*the book*" could be applied a second time and would then lead to (*)"*the the book*". Again this can be solved by increasing information in the nonterminal symbols, e.g. by creating another symbol for a noun phrase that cannot be derived to a leading determiner.

There are of course consequences to putting more information into the grammar symbols: The number of symbols and rules increases dramatically. This increases resource demand for computa-

tions. And finally any statistical approach will require a lot more data, or data sparsity will become a problem. The grammars used in this project will not be augmented by any additional information beyond the POS tag itself.

3.1.4. Beyond Context-Free

One way to increase the capabilities of a grammar is to reduce the constraints on its set of rules. If context-free grammars are not powerful enough to accurately model natural languages the logical step would be to use context-sensitive grammars. In context-sensitive grammars the derivation rules can take the context of a nonterminal symbol into account. But the increased generative power of context-sensitive grammars also results in increased operation complexity. For example the word problem can be solved in $O(n^3)$ for context-free languages with the CYK algorithm. The same problem for context-sensitive languages is solvable in $O(|N|^n) = O(2^n)$.

In order to avoid the exponential complexity of context-sensitive grammars, research focused on extending context-free grammars instead to reach a new class of so called mildly context-sensitive grammars. These extensions are chosen carefully so to be as powerful as necessary but as weak as possible. Amongst the independently proposed extensions proposed were Combinatory Categorial Grammars (CCG), Head Grammars, Linear Indexed Grammars and Tree Adjoining Grammars. They all describe languages that are more powerful than context-free languages without incurring the large computational overhead of context-sensitive grammars. Interestingly enough it was later proven by (K. Vijay-Shanker and D. Weir, 1994) that these four extensions all describe the same class of languages.

Since there is no authoritative source for an English grammar (in the sense of a formal language) even a CCG/English grammar would only be an approximation. While a mildly context-sensitive language model may approximate the English language better than a plain CFG model it remains unclear whether it is more useful in the context of this project. In section 3.2.2. the CCG treebank will be analysed and its suitability for this project examined.

3.1.5. Inversion Transduction Grammars

Dekai Wu (1997) introduced a novel stochastic inversion transduction grammar (ITG) formalism. This is basically a modified CFG that can simultaneously generate equivalent sentences in two different languages.

This is achieved by extending the CFG definition to a 5-tuple (N,T_1,T_2,Σ,R). T_1 and T_2 are now the sets of terminal symbols in the two respective languages. The definitions of N and Σ stay unchanged.

The derivation rules have either the form X → [Y] or X → <Y>. *X* is any nonterminal symbol in *N* while *Y* is a string that can contain nonterminal symbols and pairs of terminal symbols. These pairs can consist of exactly one symbol from set T_1 and one symbol from T_2, but either (or both) can be replaced by the empty word. Derivations are always applied to a pair of sentences, one sentence in each language, simultaneously. The most important part of the ITG are the bracket operators. The operator *[]* means the derivation is applied to both sentences normally. The inversion operator <> means the derivation is applied to the first sentence as usual, but for the second sentence the rule gets inverted.

$$
\begin{aligned}
&G_{ITG} := (N, T_1, T_2, \Sigma, R) \\
&\quad N \cap (T_1 \cup T_2) = \emptyset,\ \Sigma \in N, \\
&\quad \big((X \to [Y]) \vee (X \to \langle Y \rangle)\big) \in R \Rightarrow \\
&\qquad X \in N \\
&\qquad \wedge\, Y \in N \cup \big((T_1 \cup \{\epsilon\}) \times (T_2 \cup \{\epsilon\})\big)^*
\end{aligned}
$$

Fig. 8 – ITG definition

$$
\begin{array}{lrcl}
(1) & \Sigma & \to & [\text{IN NP}] \\
(2) & \text{NP} & \to & [\text{DT NP}] \\
(3) & \text{NP} & \to & \langle \text{NNP NN} \rangle \\
(4) & \text{IN} & \to & [\text{of/de}] \\
(5) & \text{DT} & \to & [\text{the}/\epsilon] \\
(6) & \text{NNP} & \to & [\text{gallic/gallico}] \\
(7) & \text{NN} & \to & [\text{war/bello}]
\end{array}
$$

Fig. 9 – Sample ITG derivation rules

$$
\begin{array}{lrcl}
 & (& \Sigma & , \quad \Sigma) \\
\to_1 & (& \text{IN NP} & , \quad \text{IN NP}) \\
\to_2 & (& \text{IN DT NP} & , \quad \text{IN DT NP}) \\
\to_5 & (& \text{IN the NP} & , \quad \text{IN NP}) \\
\to_3 & (& \text{IN the NNP NN} & , \quad \text{IN NN NNP}) \\
\to_4 & (& \text{of the NNP NN} & , \quad \text{de NN NNP}) \\
\to_6 & (& \text{of the gallic NN} & , \quad \text{de NN gallico}) \\
\to_7 & (& \text{of the gallic war} & , \quad \text{de bello gallico})
\end{array}
$$

Fig. 10 – Creating the sentence in both languages simultaneously by applying the ITG derivation rules

An ITG can not express all possible word mappings. It is not possible to move the preposition '*of/de*' into the inner noun phrase between '*gallic*' and '*war*' in this example. Theoretically ITGs are of limited use with languages that have a very loose or even completely free word order. Latin is a free-word-order language, but there is usually a tendency to prefer one word ordering, so ITGs will still work to some extent.

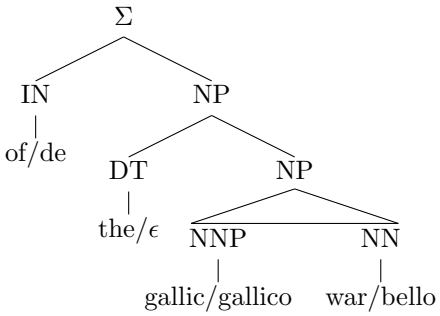

Fig. 11 – A syntax tree showing both sentence parses with the triangle marking the inversion operator

3.2. Treebanks as Grammar Sources

Treebanks are collections of sentences, with a few exceptions monolingual, annotated with syntactic information. They are a very useful tool in computational linguistics to train and test parsers, taggers and other machine learning software working on Natural Language Processing. Treebanks can either be created completely manually or semi-automatically with a parser suggesting a syntactic structure that then has to be checked by a linguist. Either way it is a very labour-intensive and time-consuming process.

3.2.1. Penn Treebank

The Penn Treebank is the best known English language treebank. Work on the Penn Treebank started in 1989 at the University of Pennsylvania. In its latest revision it contains more than 7 million words. This project will work on the sections 02–21 of the Wall Street Journal corpus. These sections contain 1,014,129 words in 39,832 sentences. Every sentence is annotated with a syntax tree, null elements and even a few semantic roles.

```
(SBARQ (WHNP-1 What)
       (SQ (VBZ is)
           (NP-SBJ Tim)
           (VP (VBG eating)
               (NP (-NONE- *T*-1)))
       (. ?))
```

Fig. 12 – Example sentence in Penn Treebank annotation

In the question "*What is Tim eating?*" there is such a null element: Whatever it is that Tim is eating, it should follow the verb 'eating' in a normal sentence. In a question however the food gets replaced by the word '*what*', which then moves to the front of the sentence. The original space of the word is marked by the null element **T**. Both the null element and the corresponding wh-word receive the index number 1. Additionally '*Tim*' is identified as subject and marked with '*-SBJ*'. A complete list of the 36 POS tags (without punctuation), clause and phrase annotation can be found in the appendix, section 7.1. For the grammar model these sentence markers, null elements, functional tags and the words themselves will be filtered. What remains are the three derivation rules: SBARQ → WHNP SQ, SQ → VBZ NP VP and VP → VBG NP.

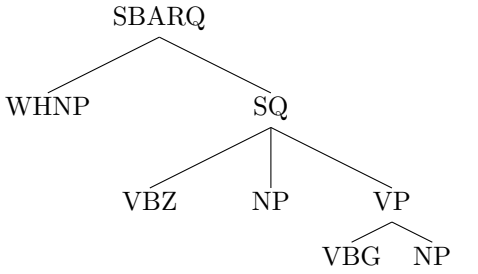

Fig. 13 – Syntax tree of the example sentence

With these simplifications a total of 10,330 distinct derivation rules can be extracted from the analysed treebank sections. By plotting the count of seen distinct derivation rules against the number of analysed sentences it can be easily shown that the rule count is not yet reaching a plateau. This indicates that there are still more new, unseen rules to be learned. The growth behaviour is remarkably similar to that of the number of seen distinct words.

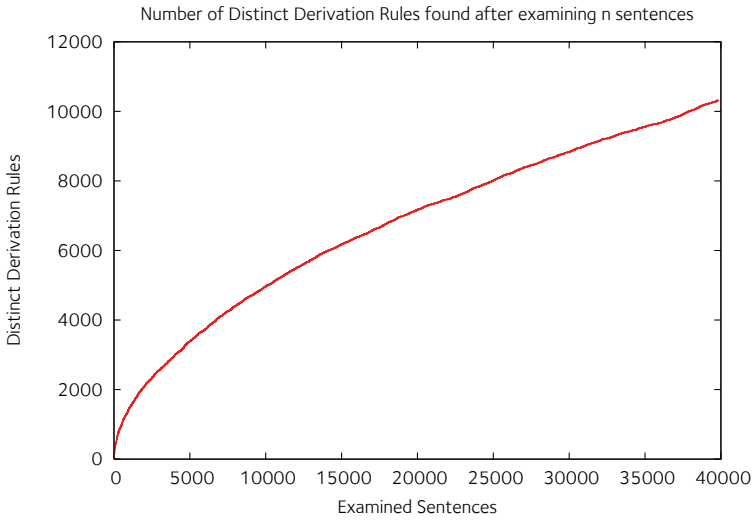

Fig. 14 – Increase in the number of seen distinct derivation rules while analysing the Penn Treebank

Is there an upper bound for the number of distinct rules?

Depending on the number of nonterminals *N* and terminals *T* the upper bound can be formulated as $|\mathrm{N}| \sum_{n=1}^{k} |\mathrm{N} \cup \mathrm{T}|^n$ with *k* being the maximum length of the right hand side of a derivation rule. The Penn Treebank annotation guide does not impose a maximum length. The longest seen derivation rule has length 12 – ignoring one rule with an extraordinary length of 27. The above formula with k=12 gives a number of $1.03*10^{23}$ possible rule combinations.

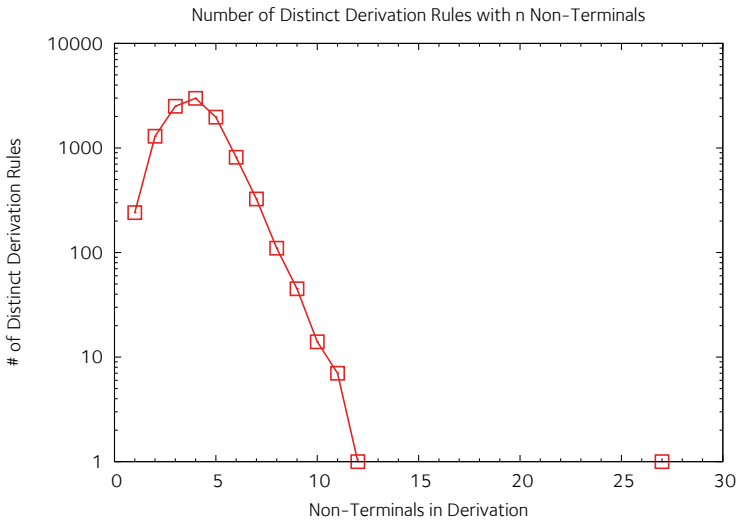

Fig. 15 – Number of distinct derivation rules seen in the Penn Treebank, grouped by length of their right hand side

The reason why the Penn Treebank has very long derivation rules lies in its flat annotation structure. The term "*New York Stock Exchange*" for example would just result in one rule instead of possibly up to three rules. Another example that does not involve a trademark is the rule VP → (VB spin) (PRT off) (NP its textiles operations) (PP to existing shareholders) (PP in a restructuring) (S to boost shareholder value) of length 6.

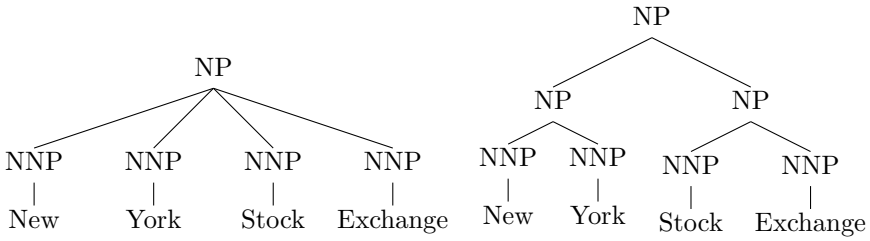

Fig. 16/17 – Example of structural flatness in the Penn Treebank. On the left the syntax tree of the term "New York Stock Exchange" as it appears in the treebank. An alternative syntax tree is shown on the right.

Long derivation rules will inevitably lead to a long tail in the rule distribution: A lot of the rules are only observed a few times. Approximating the real distribution becomes difficult due to data sparsity. Having only short rules however is not necessarily better: A maximum rule length of 2 would result in a maximum of 104,832 distinct rules in the Penn Treebank. Even though approximating the real distribution becomes much easier due to the reduced number of possible rules, the amount of information carried within a single rule diminishes.

3.2.2. CCG Bank

The Combinatory Categorial Grammar (CCG) Bank is based on the Penn Treebank. The translation algorithm described in Hockenmaier and Steedman (2002) creates a categorial derivation tree for each sentence. In a CCG each word is assigned a category, a syntactic type. These types are either a primitive category (e.g. *S*, *NP*, *PP*, *N*) or functions. Function categories are a combination of primitive categories, the forward application operator (/), the backward application operator (\) and appropriate bracketing to establish operator precedence. In the sentence '*Tim laughed.*' *Tim* will be assigned the primitive category *NP*. The transitive verb *laughed* is an entity that will expect one argument (*NP*), specifically to its left. Once this argument is applied the result will be a sentence (*S*). Therefore the verb will be tagged *S\NP*, read: a sentence that is missing an *NP* to its left.

$$
\begin{array}{llll}
(1) & X/Y & Y & \Rightarrow X \\
(2) & Y & X\backslash Y & \Rightarrow X \\
(3) & X/Y & Y/Z & \Rightarrow X/Z \\
(4) & X & & \Rightarrow Y/(Y\backslash X)
\end{array}
$$

Fig. 18 – Sample function applications

A selected set of the possible function applications is shown in figure 18. The forward application (*1*) means that a function (*X/Y*) takes an argument (*Y*) to its right and returns type (*X*). By applying these rules the complete derivation tree can be built from the bottom up. *X* and *Y* themselves stand for any possible category. Derivation rules can be deduced by reading these rules from right to left: X → X/Y Y, etc.

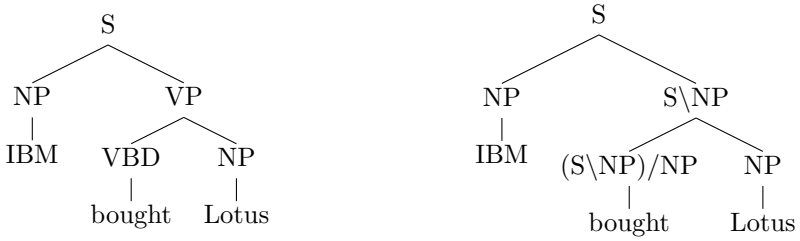

Fig. 19/20 – Example sentence with regular syntax tree on the left.
To the right the same sentence annotated with CCG

In the sentence '*IBM bought Lotus*' the intransitive verb bought expects two arguments: Something that is buying and something that is being bought. Both subject and object are again of type *NP*. '*Bought*' expects one argument to its right and one to its left. As in a normal syntax tree bought '*Lotus*' is considered a verb phrase on its own. This closer connection dictates operator precedence: '*Lotus*' has to be applied first. The outcome of this application will result in a category that expects another *NP* to its left. This type is the same *(S\NP)* as that of the transitive verb encountered previously. Now the category of '*bought*' can be defined as *(S\NP)/NP*.

CCG has the advantage that by following function application semantic structure within the sentence can be extracted. The set of primitive categories is very small and there are only a few function application rules. By design CCG derivation trees can only have a branching factor of 1 or 2.

But there are two major drawbacks making CCG unsuitable for this project. Firstly the information is stored at the word level. The derivation tree is defined by the categories, not the other way around. This also causes categories to become increasingly complex in larger sentences. An example from the CCG bank is the category *(((S\NP)/((S\NP)/NP))/NP)\((((S\NP)/((S\NP)/NP))/NP)/NP)* assigned to the phrase 'BPC residents'. The number of possible rules therefore is again unbounded, which leads to data sparsity problems. Too much encoded information will also make changing the grammar in a consistent way across many sentences quite difficult.

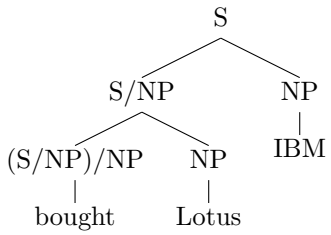

Fig. 21 – Word order change

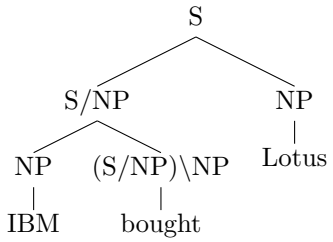

Fig. 22 – Semantic change modifying two categories

The second drawback is that modifying the language is much more complicated than with a simple CFG. Since the derivation tree is completely dependent on the word level categories those have to be modified. Furthermore in more complex sentences there are interdependencies between words. If one category is changed others have to be changed as well so that a valid tree can still be built. One small change can cause a ripple effect throughout the tree. This will make it very difficult to change the grammar in a controlled way to e.g. get closer to another language model.

3.2.3. Penn Chinese Treebank

The Penn Chinese Treebank was also created at the University of Pennsylvania. It contains approximately 550,000 tokens in 19,000 sentences, and is compiled from sources such as Xinhua newswire, Sinorama news magazine and Hong Kong News. The treebank uses 33 POS tags (without punctuation) and uses a format similar to that of the Penn Treebank.

```
(NP-PRD (NP (PP (P 对)
                (NP (NN 外)))
            (NP 开放))
        (NP (NN 城市)))
```

Fig. 23 – Example from the Chinese Penn Treebank

3.2.4. German Treebank TIGER

The TIGER Treebank was a joint project of institutes at the Saarland University, the Universität Stuttgart and the University of Potsdam. In its latest version 2.1 it contains 890,000 tokens in 50,000 sentences from the German newspaper Frankfurter Rundschau. It is freely available for research purposes.

The treebank was semi-automatically POS-tagged and annotated with syntactic structure. Its 54 POS tags represent a slightly modified version of the STTS tagset, a standard tagset used by all the major German treebanks. STTS was developed at the University of Stuttgart and the University of Tübingen. Nearly all words in the TIGER treebank are additionally annotated with detailed information about their case, number, gender, person, degree, tense and mood.

```
<graph root="s4231_VROOT" discontinuous="true">
 <terminals>
  <t id="s4231_1" word="In" lemma="in" pos="APPR" morph="--"/>
  <t id="s4231_2" word="Japan" lemma="Japan" pos="NE" morph="Dat.Sg.Neut"/>
  <t id="s4231_3" word="wird" lemma="werden" pos="VAFIN" morph="3.Sg.Pres.Ind"/>
  <t id="s4231_4" word="offenbar" lemma="offenbar" pos="ADJD" morph="Pos"/>
(..)
 <nonterminals>
  <nt id="s4231_500" cat="PP">
   <edge label="AC" idref="s4231_1"/>
   <edge label="NK" idref="s4231_2"/>
  </nt>
```

Fig. 24 – Example from the TIGER Treebank in XML format

3.2.5. German Treebank TüBa-D/S

The Tübingen Treebank of Spoken German (TüBa-D/S) is a syntactically annotated corpus based on spontaneous spoken dialogues. It consists of around 38,000 sentences with about 360,000 words, annotated with the STTS tagset. It does however not come with the additional information available in the TIGER treebank. It is also freely available for scientific purposes.

3.3. Compatibility of Language Models

Working with data from more than one treebank inevitably leads to the problem that every treebank likes to use its own format, annotation guidelines and tagset. While overcoming the format barrier is merely a programming exercise, working with disparate tagsets can be tedious.

To be able to cross the tagset barrier a translation, a mapping from the tags in one language to the tags in the other, is required. The way this mapping will be accomplished in this project is by means of an intermediate (common) POS tagset and two surjective and total translation functions. This common tagset will be kept as close to the English tagset as possible, the only difference to the English tagset will be that elements that cannot be reached from the foreign language will be removed, or merged with reachable elements, or as a last resort collected in a special catch-all tag that will be used for all tags that have no correspondent.

The tagset used in the Chinese Treebank was clearly inspired by the Penn Treebank tagset, but it contains several important differences. While the Penn Treebank offers 36 POS tags (ignoring punctuation tags) the Chinese Treebank has 33 POS tags. However out of these 33 POS tags only 31 tags

actually appear in the corpus, out of which 8 are particles and 6 are classified 'other'. The remaining 17 tags are sometimes quite different from the English set. The Chinese word 毁灭 can, depending on context, be translated to *destroy, destroys, destroyed, destroying* or *destruction*. Consequently the Chinese treebank only has 4 POS tags to indicate different verb forms, but these do not map directly to any of the 7 POS tags of the English treebank. There is a suggested mapping from Chinese tags to English in Xia (2000), but it is neither total nor surjective nor a function.

A small excerpt of the mapping procedure is presented to the right. The Chinese tags *VA* and *VV* can both mark verbs equivalent to the English *VB*. Additionally *VV* can mark modal verbs, *MD* in English. Since it is impossible to distinguish whether a tagged Chinese word corresponds to *VB* or *MD* a single common tag will be assigned to the entire group. The two translation functions will then map every involved tag to this new common tag. Likewise Chinese noun tags lack number information, so they will be mapped to the English singular noun *NN*. The English plural noun *NNS* stands isolated. Depending on the view the mapping has to be surjective and total, so isolated tags can not be not allowed. *NNS* is closest in meaning to the Chinese *NN*, so it is added to the noun cluster. Once all isolated tags are assigned to their closest equivalent (or a catch-all common tag for tags without an appropriate equivalent) the entire connected component is assigned a tag in the new common tagset.

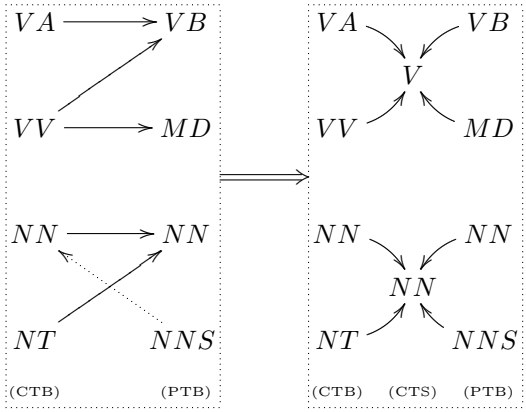

Fig. 25 – Example of POS tag mapping functions

One possible mapping between English and Chinese tags created by this method is presented in the appendix, chapter 7.2. In this case the common tagset consists of 18 tags. Generally a higher number of common tags is expected to give more reliable results later on, as a higher number suggests a higher precision in what this cluster represents. A lower number of common tags represents less information, and will make the languages appear closer together than they really are. If the common tagset were to contain only a single tag (i.e. *WORD*) then the involved languages would look identical.

Trying to map CCG categories to POS tags in other languages poses huge difficulties. Due to the possibly infinite number of distinct categories an automatic mapping process would be needed. Together with the already mentioned problems of modifying the grammar it was decided not to pursue the use of CCG in this project any further.

The German STTS tagset is also quite different from the English tagset but it is arguably a lot closer than the Chinese tagset. In fact the additional annotation in the TIGER treebank can be used together with the German POS tag to find appropriate English tags to map to. Most of the time a very exact matching, in some cases 1:1, is possible. The common tagset between English/Penn Treebank and German/TIGER contains 32 tags.

Since TüBa-D/S is lacking the additional annotation of the TIGER treebank some information is lost. Like with the Chinese treebank singular and plural cannot be distinguished. The third person verb forms have to be unified with the regular ones, differentiation between different degrees of adjectives is no longer possible. The resulting common tagset between English/PTB and German/TüBa-D/S contains only 26 tags.

3.4. Comparability of Language Models

With the introduction of a common tagset it should now be possible to compare two languages. Comparing languages is required to do directed evolution. Ideally one has a function that returns a number indicating whether two language models are equal, or how far apart they are. Howev-

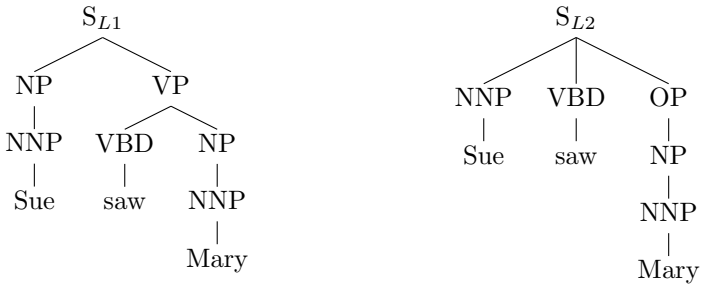

Fig. 26/27 – The same sentence in two different annotations

er comparing two CFG language models is harder than it seems. Strictly speaking the equivalence problem for Context-Free Grammars is even undecidable.

With these two sentences from two language models it is not possible to deduce whether the two models describe the same language. One way of comparing the two languages is to compare the derivation trees of the sentences. The node annotation is a bit different and would have to be translated to a common tagset as it was the case with the POS tags. It can be seen easily that a noun phrase is a hyponym, a sub-type of an object phrase (*OP*). But except from the derivation NP → NNP for Mary the derivation trees have nothing in common. Another way of comparing is by

looking at the output. The actual words are of course not included in either CFG, but they would be of no help since the two CFGs are describing different real languages and hardly any words would match. So the POS tag sequence *NNP VBD NNP* has to be used instead.

It is very simple to decide whether this sequence can occur in the other language. The word problem for Context-Free Grammars can be solved in $O(n^3)$ with the CYK algorithm. The algorithm would however only return true or false – not exactly helpful to determine the distance between two languages. A better variation is not to determine if a sequence can be generated, but to determine the likelihood that this sequence is generated. The probability of a sequence is the sum of all the probabilities of its possible parses, the probability of a specific parse is the product of the probabilities of all involved derivation rules.

A few problems remain with this approach: If there is no possible parse for the sequence then the sequence will have a probability of zero. Especially with flat grammar structures this could be caused by data sparsity. It was shown in chapter 3.2.1. that there are 1.03*10^{23} possible rule combinations for Penn Treebank derivation rules with a maximum length of 12. Data sparsity is expected to be a big problem, especially because the two language models will not be based on translations of the same sentences.

3.5. Introducing N-Gram Models

There are two ways to avoid or mitigate the data sparsity problem. One is to apply some kind of smoothing to the PCFG model to assign unseen derivations a low but non-zero probability. Smoothing would likely be a quite complicated process, and it is uncertain how successful smoothing on the PCFG can be.

There is another widely used model in natural language processing, called the *n-gram* model. The probability of a sentence *S* occurring in a language *L* can also be rewritten using the chain rule and conditional probabilities. These probabilities can then be approximated by making the independence

$$\begin{aligned} P(S|L) &= P(t_1 t_2 t_3 t_4 t_5 \dots t_n|L) \\ &= P(t_1|L) \times P(t_2|L,t_1) \times P(t_3|L,t_1,t_2) \times P(t_4|L,t_1,t_2,t_3) \times P(t_5|L,t_1,t_2,t_3,t_4) \times \dots \\ &\overset{\text{3-gram}}{\approx} P(t_1|L) \times P(t_2|L,t_1) \times P(t_3|L,t_1,t_2) \times P(t_4|L,t_2,t_3) \times P(t_5|L,t_3,t_4) \times \dots \\ &\overset{\text{2-gram}}{\approx} P(t_1|L) \times P(t_2|L,t_1) \times P(t_3|L,t_2) \times P(t_4|L,t_3) \times P(t_5|L,t_4) \times \dots \times P(t_n|L,t_{n-1}) \end{aligned}$$

Fig. 28 – Probability of a sentence occuring in a language according to the chain rule, a trigram and a bigram model.

assumption that the probability of any specific tag occurring only depends on the (n-1) preceding tags. Each of these sub-sequences containing (n-1) tags plus the following tag constitutes an n-gram. Probabilities are estimated by the usual maximum likelihood estimation.

The described view is that of an (n-1)-th order Markov model. As in any other Markov model the past beyond a certain point does not influence the current state. Disregarding parts of the history of a sentence of course means that in some cases long range dependencies cannot be taken into account.[4] Precision can be balanced against data sparsity issues by choosing an appropriate value for *n*.

This project will use 5-gram models. Of course there is a very high number of 5-grams that will only be seen a few times or not at all, so again smoothing is crucial. There are some established smoothing methods for n-gram models. The simplest method is to treat any n-gram like it was seen at least once. However this technique allocates a relatively high probability to the entire mass of unseen n-grams, which is why it is usually not used in practice.

A far more sophisticated solution is the Katz back-off smoothing technique, presented in (S. M. Katz, 1987). This technique leaves probability estimates of very common n-grams as they are, as the estimation is deemed to be quite reliable due to a lot of evidence. Probability estimates of rarely seen (less than *k* times) n-grams are reduced in favour of the probability of unseen n-grams. The probability of unseen n-grams is estimated recursively by looking at the Katz smoothed (n-1)-gram model of the data. For this project *k* was chosen to have a value of 9 in the English model, 5 in the Chinese model and 10 in the German models.

With n-gram models the problems of directly comparing PCFGs can be avoided. Using n-gram models instead of PCFG parsing probabilities has three more practical advantages: It is not necessary to map phrase and clause level annotations between language models, it is not necessary to implement a parser to calculate the probabilities, and finally it is not necessary to create PCFG models for the foreign languages as the n-gram models can be extracted directly from the corpora.

4 The practical viability of n-gram models was for example shown in the practical of the lecture Computational Linguistics. The assignment was to program a Parts of Speech tagger that was to be trained on the words and tags of a part of the Penn Treebank, and then had to assign POS tags to untagged text. It was possible to correctly annotate between 88% (on texts with previously unseen words) and 98% (on texts without unseen words) of the untagged text with the help of a bigram model.

3.6. Comparing N-Gram Models

With this definition of a sentence probability the next logical step is to define a distance between two sentences. Three example sentences are shown on the right. For each sentence fictional, but within the example plausible, probabilities of it appearing in English texts and in German texts are stated. Due to the used definition of sentence probability shorter sentences will usually have a much higher probability than longer sentences. So while it makes sense to compare the probabilities of sentences of the same length between languages, it does not make sense to compare probabilities between sentences of different lengths.

S					$P(S\|L_E)$	$P(S\|L_G)$
I (PRP)	have (VBP)	this (DT)	book (NN)	read. (VBN)	0.00001	0.00095
I (PRP)	have (VBP)	read (VBN)	this (DT)	book. (NN)	0.00095	0.00001
I (PRP)	went (VBD)	home. (NN)			0.015	0.015

Fig. 29 – Sentence probabilities in English and German language

One way to measure how far apart two probability distributions over the sentence space are, is the so called relative entropy or Kullback-Leibler (KL) divergence. Intuitively the total difference between two entire models is the sum of all the differences on a sentence level. The KL divergence of two language models L_1 and L_2 is defined as $d(L_1||L_2) = \sum_i P(i|L_1) \log \frac{P(i|L_1)}{P(i|L_2)}$, which is just a weighted sum of the probability differences between all sentences. The weighting of the sum causes the KL divergence to become asymmetric, which is the reason why it is called KL divergence rather than KL distance.

This definition is of course rather problematic for any real world scenario. The real sentence probability function P is unknown and approximated by the Katz smoothed n-gram model K. Hence the real KL divergence can only be approximated with $d(L_1||L_2) \approx \sum_i K_{L_1}(i) \log \frac{K_{L_1}(i)}{K_{L_2}(i)}$. The next problem is that the KL divergence is a sum over the entire sentence space. The weighted sum of course means that any sentences with a zero probability can be ignored, but the Katz smoothing of the n-gram model ensured that there are no sentences with a zero probability. Since calculating the probabilities of an infinite number of sentences is out of the question another simplification is needed. The best way to approximate a weighted sum is to only consider the most important summands with very high weights. Sentences with a high probability K_{L_1} can be found easily – in the L_1 corpus.

While technically any sentences from the corpus could be used for the approximation of the KL distance it seemed prudent to use a set of sentences that is separate from the set of sentences that was used to create the PCFG or n-gram models. For the English Penn Treebank section 00 (containing 1,921 sentences) is used for this task. Similarly in the Chinese and German treebanks the first 1,921 sentences are put aside and are not used to create the language model.

The KL divergence approximation is now defined as $d(L_1||L_2) \approx \sum_{i=1}^{n} K_{L_1}(s_i^{L_1}) \log \frac{K_{L_1}(s_i^{L_1})}{K_{L_2}(s_i^{L_1})}$ with s being the set of sentences put aside for the divergence calculation. After removing an infinite chunk of the original weighted sum it is a natural step to normalize the remaining weights so that the sum of the weights adds up to 1 again. But there is good reason to change the weighting entirely: As it was mentioned before, using the sentence probability to compare between sentences of different lengths is not very meaningful. Since the sentences in the set s are probably not of a uniform length using the n-gram probability at this point will certainly bias the results toward short sentences.

One solution is to re-estimate the sentence probability. In the weighted sum only the probability relative to the other sentences in the set s is required. A simple way to estimate this probability is by using a maximum likelihood estimation on the sentences in s. In other words each sentence in the set s will be predicted as equally likely, which finally results in the KL divergence approximation $d(L_1||L_2) \approx \sum_{i=1}^{n} \frac{1}{n} \log \frac{K_{L_1}(s_i^{L_1})}{K_{L_2}(s_i^{L_1})}$. The original KL divergence was guaranteed to be nonnegative. While this will usually still be the case it can due to the approximations no longer be guaranteed.

To get from the asymmetrical divergence to a real distance between two language models the divergence and the inverted divergence can be added up: $d(L_1, L_2) := d(L_1||L_2) + d(L_2||L_1)$.

3.7. Distance Learning

A controlling instance can now close the loop. It modifies the original English Probabilistic Context Free Grammar extracted from the Penn Treebank. The output of the PCFG, sentences in POS tag form, is then fed into a new n-gram model. The new KL divergence is then calculated. Depending on whether the modified English got closer to the foreign language the modification is kept or reverted. The process then starts anew.

In a lot of cases more than one change to the original grammar is needed to get closer to a foreign grammar. In the intermediate steps the KL divergence will rise. Hence the value of greedy algorithms will be limited. This optimization problem can for example be tackled by a simulated annealing algorithm or by genetic algorithms.

3.8. Language Evolution Model – An Overview

The schema below provides an overview for one complete possible (directed) evolution model for natural languages. Numbers next to the elements reference the relevant chapter.

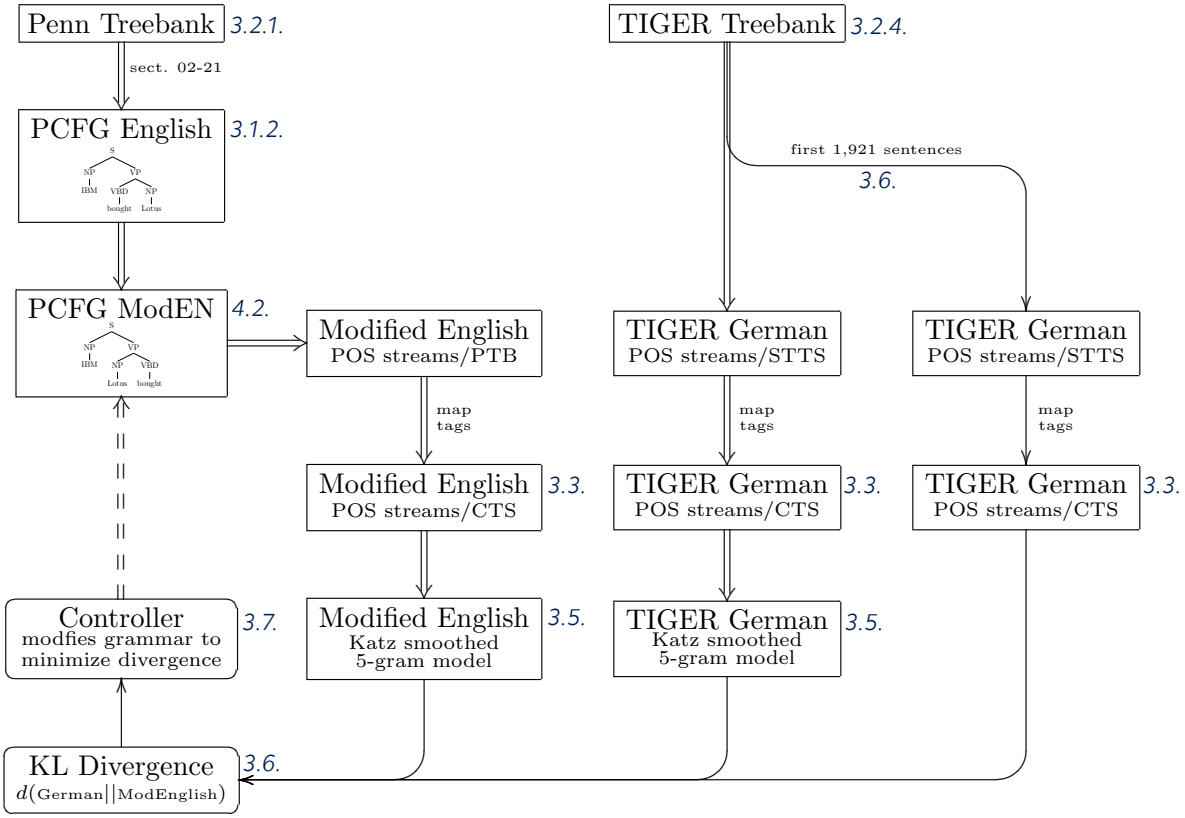

Fig. 30 – The entire directed evolution model

Chapter 4

Details & Results

"Those who know nothing of foreign languages know nothing of their own."
— Johann Wolfgang von Goethe

4.1. Measuring Distances

foreign language L_F	$d(L_E\|\|L_F)$	$d(L_F\|\|L_E)$	$d(L_E, L_F)$	|CTS|
Penn Chinese Treebank	31.91	45.29	77.20	18
German Treebank TIGER	17.21	14.38	31.59	32
German Treebank TüBaD/S	27.89	14.56	42.45	26

Fig. 31 – KL divergences and distances of English to the three foreign language models

In a first step the existing divergence between the n-gram model of original English and the other languages is determined. The first two columns show the KL divergence and the inverse KL divergence between the languages. The third column contains the KL distance, the sum of the KL divergences, and the fourth column holds the size of the used common tagset. As was already mentioned in chapter 3.3. a smaller common tagset will cause the distance between languages to appear smaller than it actually is. It is all the more surprising and reassuring that the calculated divergences between the languages behave as expected.

The divergences between English and Chinese and vice versa are the highest across-the-board, although the associated common tagset is the smallest with only 16 tags. The model based on TIGER gets closest to English, with TüBaD/S coming in second. This result was to be expected considering that TIGER is, like the used sections of the Penn Treebank, a collection of newspaper text. TüBaD/S instead is a collection of spoken dialogue. Its model should inherently be a lot more chaotic as people tend to insert interjections, get interrupted mid-sentence, and generally not adhere to a strict grammar when speaking. A rather interesting result is that the divergence $d(L_F||L_E)$ is nearly identical for both German language models. This direction of the divergence is calculated by calculating the probabilities using sentences from the German corpora, and apparently both the written and spoken German sentences are equally distant from the English language model.

4.2. Modifying the English Language

Within the language evolution model as shown in section 3.8. and explained in section 3.5. the English language PCFG has to be translated into an n-gram model. For this model a number of POS tag sequences is required. There are three ways of doing this: The mathematically correct way is to start with the PCFG and generate all possible sentences. These can then be fed into the n-gram model, and the distribution of POS tag sequences corresponds exactly to the language defined by the PCFG.

Creating an infinite number of sentences is inherently impractical, so an alternative is to use random sampling. By creating random sentences according to the PCFG the distribution of the POS tag sequences will converge to the distribution of sentences in the Probabilistic Context-Free Language. Generating random sentences is reasonably simple. Care has to be taken that the length of the generated sentences has a similar distribution as the original language, as there is no sentence length information encoded in a PCFG, and having very long sentences will bias the n-gram model. Another problem with this randomized algorithm is that the output will not be consistent, so the KL divergences obtained before and after a language modification cannot be compared directly.

Finally there is the realistic and pragmatic approach that avoids generating sentences altogether: Since the PCFG is generated from real sentences from a corpus an obvious choice is to use these sentences to create the n-gram model. In order to do this the syntax trees of all those sentences have to be kept. The PCFG can be derived easily from those syntax trees. The POS tag sequences required to create the n-gram model can be recovered from the syntax tree by means of an in-order depth first search. The advantage of having the syntax trees is that the extracted POS tag sequences mirror the PCFG distribution exactly, as they come from the same source. Due to the dependency of the PCFG on the syntax trees it also means that the PCFG does not have to be generated explicitly. The drawback to this approach is that manipulating the PCFG directly is no longer possible – otherwise the link to the syntax trees is lost, and it again becomes necessary to generate POS tag sequences for the n-gram model. Any modifications to the grammar have to be made by changing the stored syntax trees.

In the program used in this project to simulate language change the syntax tree model was used. The consequences of this choice and the possible operations on the language will be explored in the following sections.

4.2.1. Adding and Removing Rules

The most basic operations on a Context Free Grammar are to add or remove derivation rules. In a Probabilistic CFG this corresponds to assigning a positive or zero probability to a derivation rule and subsequently re-normalizing the probabilities of all derivation rules with the same left hand side, so that the sum of these probabilities equals 1.

These operations were not used in this project. In the syntax tree model sentences are manipulated directly. This means adding rules involves introducing new POS tags into existing sentences, while removing rules deletes existing POS tags. If a rule that is to be deleted has sub-trees then

these have to be dealt with, either by deleting them, too, or re-attaching them to the tree in a different place. Moreover, the linguistic motivation behind these sentence modifications seems inconclusive. It is also unclear how these new rules and the places they are applied to are chosen.

In a pure PCFG model adding rules can be justified somewhat more easily because these rules are applied automatically while generating sentences. It is also possible to adjust the probabilities of existing rules, making a certain existing construct more probable or obsolete. How new rules are chosen if of course again undefined.

4.2.2. Change Across Rules

More complicated operations on grammars can involve changing more than one rule at the same time. In the syntax tree model these are tree operations. A sub-tree can be promoted by removing its root node and reconnecting all its child nodes directly to its parent. Similarly the inverse operation, demotion, works by introducing a new intermediate parent node for a group of siblings. Sub-tree raising leaves all node connections within a sub-tree intact, but assigns a new root – similar to an AVL rebalancing operation.

While the shown example operations do change the syntax tree they do not change the produced sentence. They do have an effect in combination with other modifications. Tree operations are currently not used in the simulation.

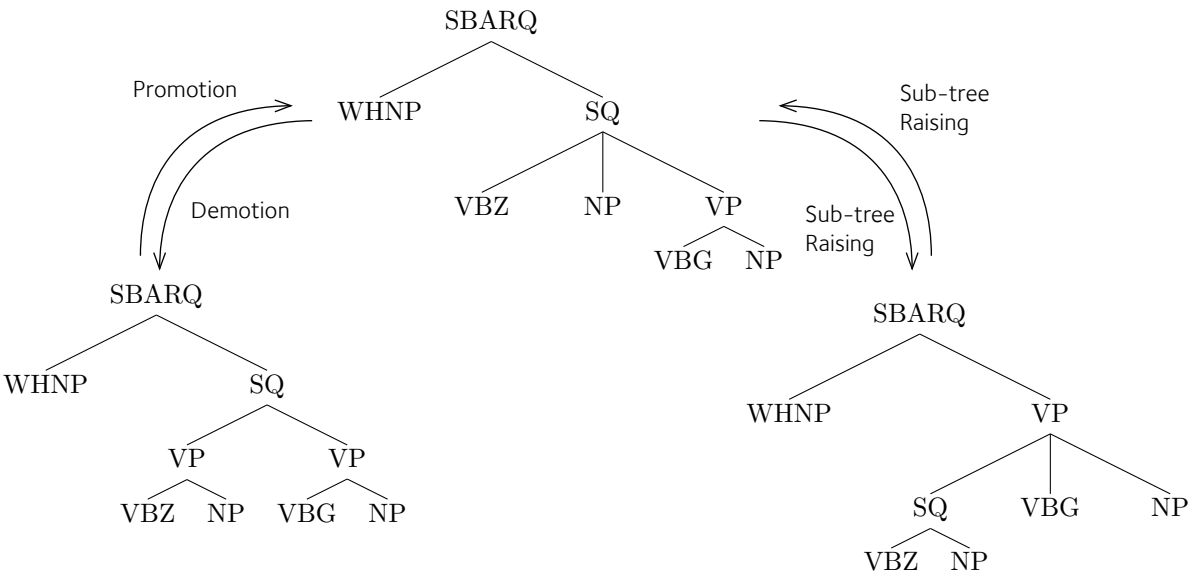

Fig. 32 – Possible tree operations to modify a language model

4.2.3. Change Within Rules

Another possibility to modify an existing grammar is to change existing rules. Changing rules does not only include introducing or removing symbols from the right hand side of a derivation rule, but also means changing the order of existing symbols. The latter is particularly interesting in the context of this project.

Swapping pairs of symbols within production rules of a grammar corresponds to reordering subtrees of a node in a syntax tree. A swap was all that was needed to align the sentences "I have read this book"/"Ich habe dieses Buch gelesen" in the first chapter. Since all possible permutations of a derivation rule can be reached by swapping pairs. This operation alone gives the evolutionary model the expressiveness of the previously mentioned Inversion Transduction Grammars.

An analysis of the PCFG extracted from the Penn Treebank showed that 60% of the derivation rules do not have any of their possible permutations appearing in the treebank. This includes rules with a very high frequency, for example VP -> TO VP occurs 12,958 times, VP -> VP TO does not appear in the treebank. SBAR -> IN S starts a relative clause (... that key officials may fail) and occurs 9,488 times, its inverse counterpart fails to appear once. These numbers suggest that there is indeed a lot of information stored in some simple derivation rules.

Computing swaps or complete rule inversions is very simple. Since swaps do not directly affect sub-trees or other derivation rules it is also a lot easier to later optimize a model for a low KL divergence.

4.3. Directed Evolution

The perl program presented in the appendix uses the following method to simulate directed evolution: The syntax trees of the English treebank are loaded. Then the program picks any binary rule that appears at least 300 times in the entire corpus and then swaps all occurrences of this rule in all the syntax trees. Then the n-gram models are recalculated and the change in the KL divergence is observed. If the divergence improves, or at least does not grow beyond a certain threshold above the best known divergence, then the rule change is kept and recorded. Otherwise the rule change is reverted.

The constraint on which rules are chosen will be relaxed after some time. After 30 simulated generations rules that occur at least 100 times will also be considered for manipulation. The threshold for keeping grammar modifications is reduced by a small amount in each generation to slowly enforce reaching a possibly local KL divergence minimum.

Even with heavy use of efficient data structures, intermediate result caching and partially lazy evaluation of the n-gram model, simulating language change is a very time-demanding and memory-intensive process. Currently between 20 and 30 seconds are required for each generation and around 1 GB of RAM is needed for the language models. Most of the processing time is spent for creating the modified Katz-smoothed n-gram models.

The longest running simulation so far was on English and German/TIGER. It spanned 369 generations and took slightly over three hours. In the end 25 swaps were deemed useful, or at least not too detrimental, and were kept. The KL divergence between German and English was reduced to 11.00 from the original 14.38.

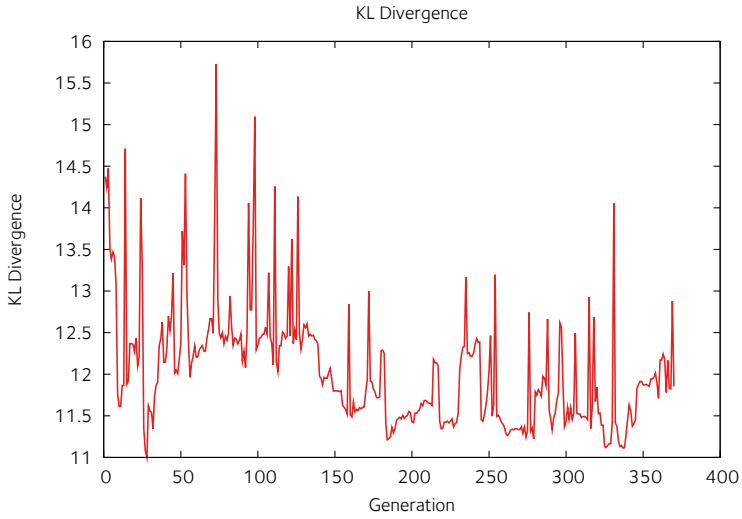

Fig. 33 – Change of the KL divergence during the simulation

By monitoring the change of the KL divergence during the simulation is becomes obvious that the restrictions on which rules are eligible for swaps play an important role. Most progress in reducing the KL divergence was made in the first 30 generations, with the minimum reached in the 27th generation. Afterwards the KL divergence appears to be just moving around randomly.

4.4. Interpreting Results

The simulation run to modify English grammar to get closer to the model of German obtained from the TIGER treebank returned the following 25 rule sets in this order:

NP → [CD NNP]	VP → [VB VP]	NP → [ADJP NNS]	VP → [MD VP]
VP → [PP VBG]	VP → [NP VBN]	NP → [NNP NNP]	VP → [PP VBN]
VP → [SBAR VBD]	NP → [NNP NNP]	NP → [NP NP]	NP → [NP SBAR]
NP → [CD NNP]	NP → [NN PRP$]	ADVP → [PP RB]	VP → [ADVP VB]
VP → [S VBD]	VP → [PP VBP]	NP → [NNP NNPS]	
VP → [VBD VP]	VP → [NP VBN]	VP → [S VBP]	
VP → [S VBN]	VP → [PP VBG]	ADJP → [CD NN]	

Fig. 34 – Swapping these 25 rule pairs moves English closer to German

The notation with brackets is meant to convey that each rule stands for a set of affected rules. So every occurence of NP → CD NNP will be changed to NP → NNP CD and vice versa. Interestingly this swap improved the KL divergence. This swap is reversed a bit later and improved the result again, which shows the need for simulation methods like simulated annealing that can avoid getting stuck in local KL minima.

This set of rules can be reduced by removing these duplicate rules as well as symmetric swaps like NP → NNP NNP. Eventually the set of swaps can be reduced by randomly removing swaps while monitoring the effect on the KL divergence. With the remaining 11 rule changes KL divergence is improved to 10.58:

ADVP → [PP RB]	VP → [S VBD]	VP → [SBAR VBD]	VP → [VB VP]
NP → [NP SBAR]	VP → [S VBN]	VP → [PP VBN]	VP → [VBD VP]
VP → [MD VP]	VP → [S VBP]	VP → [PP VBP]	

Fig. 35 – A smaller subset of 11 rules gives even better results

The (DT)	new (JJ)	rate (NN)	will (MD)	be (VB)	payable (JJ)	Feb. 15 (NNP CD)
The (DT)	new (JJ)	rate (NN)	be (VB)	payable (JJ)	Feb. 15 (NNP CD)	will (MD)
Die (DT)	neue (JJ)	Rate (NN)	wird (VBP)	am 15. Feb. (DT CD NNP)	fällig (JJ)	werden (VB)

Fig. 36 – Effect of these 11 rules on an English sentence compared to a German translation of the same sentence

Most of these rules are concerned with verbs. By applying these swaps to a sample text their effect can be observed and interpreted (Fig 38). In this case the rule changes have the effect that a verb will move to the end of the sentence or after another constituent. Generally moving verbs to the end is a good start to get closer to the German language. In the shown example moving the modal verb to the end of the sentence is one of the required changes to achieve the same POS tag sequence. The other required changes are swapping NP → [CD NNP], which was actually found earlier, and ADJP → [JJ NP]. The remaining differences, the introduction of a second determiner and the replacement of the verb '*will*' (*MD*) with its German equivalent '*werden*' (*VB*) can not be achieved by swapping.

It is obvious that there are limits how close one can get to another language. With swaps alone it is not possible to reach the POS tag distribution of the foreign language. So if one language uses more verbs than the other, it will lead to a lower bound on the KL divergence. Then there are limits that are caused by the PCFG model itself. German for example is a V2 language, which means that in a declarative sentence the second constituent is always a verb. It is however not possible to express such a constraint in a PCFG (without resorting to tricks that lead to a sharp increase in the number of non-terminals and an equally sharp decrease in the usefulness of the grammar). One should therefore not expect to find a finite set of swapping rules that would lead to a KL divergence of zero.

Chapter 5

Conclusion & Outlook

> *"There is nothing so trivial as a grammar, and scarce any thing so rare as a good grammar."* — *G. Miège*

In this thesis a novel statistical method to compare language grammars was devised. This method would work without any particular knowledge about the languages' heritage. This method was then developed into a complete formal model for directed evolution of natural languages. The model was implemented and put to the test with the help of annotated Chinese, English, and German treebank data. Results show that it is possible to find meaningful grammatical differences that could be used in other fields of natural language processing.

While working out the intricacies of the evolutionary model, several problems with the suitability of treebank data for cross-language research in general and comparative grammar research in particular were identified. In the case of mapping different POS tagsets a complete workable solution was demonstrated, in other cases viable workarounds were presented. The advantages and drawbacks of using probabilistic context free grammars to reproduce natural language change were investigated.

The conclusion is that statistical approaches to model grammar change are workable. There is however a distinct lack of historical data to create historically accurate evolutionary models. There is also a distinct need for a common tagset or even a sufficiently general and simple common feature annotation that is consistent and consistently used across languages. Existing tagsets are usually created with one specific language in mind, and strongly reflect specific language patterns and may to a certain point even convey the author's mentality. These properties in turn deter others from reusing an existing tagset or annotation.

The project PROIEL[5] at the University of Oslo aims to create a parallel corpus in a variety of ancient Indo-European languages (Greek, Latin, Old Armenian, Old Church Slavonic and Gothic). This may become a very useful resource for cross-language research and it is certainly sensible to keep an eye on it.

The specific implementation of a directed language evolver presented in this Thesis has yet to be brought to its full potential. There are still a lot of options to optimize for speed (e.g. do not recalculate the Katz n-gram model from scratch, instead change it according to the changes in the POS tag

5 http://www.hf.uio.no/ifikk/proiel/index.html

sequences) to get a lower simulation time per generation. It will then become viable to do simulation runs on a larger scale that may get a lot closer to the foreign language, and give a larger and more detailed set of proposed changes.

The presented grammar evolver is a completely new and innovative model. Further research should go into determining exactly how close this model can evolve a language towards another. The grammar evolver should be extended to allow different changes to the syntax trees. To speed up simulation, and subsequent interpretation, rules could be changed according to some pattern, e.g. for all derivation rules containing a verb POS tag at the front and a noun phrase, the verb could be swapped with the right-most instance of a noun phrase. It may also be worthwhile to investigate and implement promotion, demotion and sub-tree raising operations. Replacing the evolution controller with a more sophisticated implementation of a simulated annealing algorithm will also certainly improve the overall results.

References

"There are worse crimes than burning books. One of them is not reading them."
— Joseph Brodsky

Cangelosi, A., and Parisi, D., 2002, Computer simulation: A new scientific approach to study of language evolution, in Simulating language evolution (A. Cangelosi and D. Parisi, Eds.), London: Springer-Verlag, pp. 3–28.

Chomsky, N., 1957, Syntactic Structures. The Hague: Mouton.

Collins, M., 2003, Head-Driven Statistical Models for Natural Language Parsing. Comput. Linguist. 29, 4 (Dec. 2003), 589-637.

Dryer, M., Haspelmath, D. Gil, Comrie, B., 2005, World Atlas of Language Structures.

Hare, M., Elman, J. L., 1995, Learning and morphological change. Cognition, 56(1):61–98.

Hockenmaier, J., Steedman, M., 2002, Acquiring Compact Lexicalized Grammars from a Cleaner Treebank, In Proceedings of the Third LREC Conference, Las Palmas, Spain. Katz, S. M., 1987, Estimation of probabilities from sparse data for the language model component of a speech recogniser. IEEE Transactions on Acoustics, Speech, and Signal Processing, 35(3), 400–401

Kayne R. S., 2005, Some notes on Comparative Syntax, in The Oxford Handbook of Comparative Syntax, pp. 3–69

Lewis, P. M., Stearns, R. E., 1968, Syntax-Directed Transduction. J. ACM 15, 3 (Jul. 1968), 465-488.

Lohöfer A., 2007, UE History of English, Philipps-Universität Marburg, 23/04/07, Session 2: Historical Linguistics and Linguistic Reconstruction, http://www.staff.uni-marburg.de/~lohoefea/Sessions/S02_Slides.pdf

Lohöfer, A., 2007, UE History of English, Philipps-Universität Marburg, 30/04/07, Session 3: From PIE to Proto-Germanic, http://www.staff.uni-marburg.de/~lohoefea/Sessions/S03_Slides.pdf

Martín-Vide, C., 2003, Formal Grammars and Languages, in R. Mitkov, ed., Oxford Handbook of Computational Linguistics: 157–177. Oxford University Press, Oxford.

Mitchell, P. M., Santorini, B., Marcinkiewicz, M. A., 1993, Building a Large Annotated Corpus of English: The Penn Treebank, in Computational Linguistics, Volume 19, Number 2 (June 1993), pp. 313–330

Mohri, M., Sproat, R., 2006, On a Common Fallacy in Computational Linguistics, A Man of Measure: Festschrift in Honour of Fred Karlsson on his 60th Birthday.

Picone, J., Lecture 33: Smoothing N-Gram Language Models, ECE 8463: Fundamentals of Speech Recognition, Mississippi State University, http://www.ece.msstate.edu/research/isip/publications/courses/ece_8463/lectures/current/lecture_33/index.html

Ryder, R. J., 2006, Grammar and Phylogenies, September 15, 2006

Searls, D., 1993, The computational linguistics of biological sequences. In Artificial Intelligence and Molecular Biology (Hunter, L., ed.), pp. 47–120, AAAI Press.

Turner, H., 2002, An introduction to methods for simulating the evolution of language, in Simulating language evolution (A. Cangelosi and D. Parisi, Eds.), London: Springer-Verlag, pp. 29–50.

Vijay-Shanker, K., Weir, D., 1994, The equivalence of four extensions of context-free grammar. Mathematical Systems Theory, 27:511–546.

Wu, D. 1997, Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. Computational Linguistics, 23:377–404.

Xia, F., 2000, The Part-Of-Speech Tagging Guidelines for the Penn Chinese Treebank (3.0) October 17, 2000.

Xue, N., Chiou, F.-D., Palmer, M., 2002, Building a Large-Scale Annotated Chinese Corpus, , Proceedings of the 19th. International Conference on Computational Linguistics (COLING 2002), Taipei, Taiwan.

Appendix

7.1. The Penn Treebank Tagset

7.1.1. Phrase Level Annotation

S	Simple declarative clause
SBAR	Clause introduced by (possibly empty) subordinating conjunction
SBARQ	Direct question
SINV	Inverted declarative sentence, subject follows tensed verb or modal
SQ	Inverted yes/no question, or main clause of a wh-question

7.1.2. Clause Level Annotation

ADJP	Adjective Phrase
ADVP	Adverb Phrase
CONJP	Conjunction Phrase
FRAG	Fragment
INTJ	Interjection
LST	List marker. Includes surrounding punctuation.
NAC	Not a Constituent (within an NP)
NP	Noun Phrase.
NX	Used within NPs to mark the head of the NP.
PP	Prepositional Phrase.
PRN	Parenthetical.
PRT	Particle. Category for words that should be tagged RP.
QP	Quantifier Phrase within NP.
RRC	Reduced Relative Clause.
UCP	Unlike Coordinated Phrase.
VP	Verb Phrase.
WHADJP	Wh-adjective Phrase.
WHAVP	Wh-adverb Phrase.
WHNP	Wh-noun Phrase.
WHPP	Wh-prepositional Phrase.
X	Unknown, uncertain, or unbracketable.

7.1.3. Word Level Annotation

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
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
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-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

Additional parts of speech tags are used for currency signs, brackets and punctuation including quotation marks.

7.2. Mapping Tagsets: Chinese/CTB to English/PTB

Only word level annotation is covered.

CTB tag	Mapping to PTB tags
AD	RB
AS	X
BA	X
CC	CC
CD	CD
CS	IN
DEC	VBG
DEG	POS
DER	V
DEV	Does not occur on its own, see VC, VE, VP and VV
DT	DT
ETC	CC
FW	FW
IJ	UH
JJ	JJ
LB	X
LC	IN
M	NN
MSP	TO
NN	NN
NP	NN. This tag should, according to the documentation, not exist on a word level. However it does, and is probably closest to a noun.
NR	NNP
NT	NN
OD	CD
P	IN
PN	PRP
PU	If occurring at the end of the sentence it will mapped to (.) In any other place it will be mapped to (:)
SB	X
SP	X

CTB tag	Mapping to PTB tags
VA	If VA is followed by DEV then both are replaced with RB If VA is not followed by DEV then it is mapped to JJ
VC	If VC is followed by DEV then both are replaced with RB If VC is not followed by DEV then it is mapped to V
VE	If VE is followed by DEV then both are replaced with RB If VE is not followed by DEV then it is mapped to V
VP	This tag should, according to the documentation, not exist on a word level. However it does, and is probably closest to a verb. If VP is followed by DEV then both are replaced with RB If VP is not followed by DEV then it is mapped to V
VV	If VV is followed by DEV then both are replaced with RB If VV is not followed by DEV then it is mapped to V
X	X

PTB tags of the English text have to be mapped as follows (second translation function):

EX → X and LS → X and RP → X and SYM → X	particles without counterpart
JJR → JJ and JJS → JJ	comparative/superlative adjectives
RBR → RB and RBS → RB and WRB → RB	
NNS → NN and NNPS → NNP	plural forms not marked in CTB
PDT → CD	
WDT → DT	
MD → V and VB → V and VBD → V and VBN → V and VBP → V and VBZ → V	
WP → PRP and WP$ → PRP and PRP$ → PRP	
$ → NN and # → NN	currency symbols

all other tags remain unchanged.

7.3. Mapping Tagsets: German/STTS to English/PTB

Only word level annotation is covered.

STTS tag	Usage	Mapping to PTB tags
ADJA	attributive adjective [das] große [Haus]	ADJA corresponds to the PTB tags JJ, JJR and JJS. Contextual information in the TIGER treebank allows a fine-grained mapping.
ADJD	adverbial or predicative adjective [er fährt] schnell, [er ist] schnell	This distinction is not made in the PTB. Mapped to JJ, JJR, JJS like ADJA.
ADV	adverb schon, bald, doch, oft	Adverbs should be mapped to RB, RBR and RBS. The TIGER corpus does however not provide contextual information for varying degrees like oft, öfter (often, more often). Adverbial comparatives and superlatives are quite rare in German and are usually replaced by adjectives. 'Am öftesten' (most often) for example is nonsensical and the adjectival superlative 'am häufigsten' would be used instead. Therefore all adverbs will be mapped to RB, and the distinction between RB, RBR and RBS is lost.
APPR	preposition, circumposition, left part in [der Stadt], ohne [mich], von [jetzt an]	IN
APPRART	preposition with implicit article im [Haus], zur [Sache]	IN
APPO	postposition [ihm] zufolge, [der Sache] wegen	has a similar function to IN
APZR	circumposition, right part [von jetzt] an	has a similar function to IN
ART	definite or indefinite article der, die, das, ein, eine	DT
CARD	cardinal number zwei [Männer], [im Jahre] 1994	CD
FM	foreign material [Er hat das mit "] A big fish [" übersetzt]	FW
ITJ	interjection mhm, ach, tja	UH
KOUI	subordinating conjunction with zu-infinitive um [zu leben], anstatt [zu fragen]	IN
KOUS	subordinating conjunction with sentence weil, daß, damit, wenn, ob	IN
KON	coordinating conjunction und, oder, aber	CC
KOKOM	comparative conjunction als, wie	IN (like)

STTS tag	Usage	Mapping to PTB tags
NN	common noun Tisch, Herr, [das] Reisen	Depending on contextual information this tag is mapped either to NN or to NNS. This STTS tag also encompasses gerund constructions (VBG) which generally act as nouns (das Reisen = travelling).
NE	proper noun Hans, Hamburg, HSV	Depending on contextual information this tag is mapped either to NNP or to NNPS.
NNE	combination of common and proper noun Theodor-Heuss-Stiftung, Niger-Delta	The Penn Treebank treats these combinations as proper nouns. TIGER does not provide contextual information, but these constructs are usually singular and are thus mapped to NNP.
PDS	substitutive demonstrative pronoun dieser, jener	DT
PDAT	attributive demonstrative pronoun jener [Mensch]	DT
PIS	substitutive indefinite pronoun keiner, viele, man, niemand	NN
PIAT	attributive indefinite pronoun kein [Mensch], irgendein [Glas]	DT (any)
PIDAT	attributive indefinite pronoun with determiner [ein] wenig [Wasser], [die] beiden [Brüder]	NN (a bit/NN of water). A mapping to DT would also be possible (both/DT brothers), but NN seems to be closer to the function of most of these words.
PPER	irreflexive personal pronoun ich, er, ihm, mich, dir	PRP
PPOSS	substitutive possessive pronoun meins, deiner	PRP
PPOSAT	attributive possessive pronoun mein [Buch], deine [Mutter]	PRP$
PRELS	substitutive relative pronoun [der Hund,] der	This tag is mapped to WP, but could also be mapped to WDT
PRELAT	attributive relative pronoun [der Mann,] dessen [Hund]	WP$ (whose)
PRF	reflexive pronoun sich, einander, dich, mir	PRP (myself)
PWS	substitutive interrogative pronoun wer, was	WP
PWAT	attributive interrogative pronoun welche [Farbe], wessen [Hut]	WP$
PWAV	interrogative adverb or adverbial relative pronoun warum, wo, wann, worüber, wobei	WRB (why, when, where)
PAV / PROAV (TIGER)	pronominal adverb dafür, dabei, deswegen, trotzdem	RB (hence, therefore)

STTS tag	Usage	Mapping to PTB tags
PTKZU	"zu" before infinitive zu [gehen]	This tag is a perfect match to the Penn Treebank tag TO. However to is a much more common word than zu.
PTKNEG	negating particle nicht	RB
PTKVZ	separated verb particle [er kommt] an, [er fährt] rad	These constructions do appear in the English language: being better off, to break down. Particles are tagged RP.
PTKANT	answer particle ja, nein, danke, bitte	These answer particles are not tagged consistently in the Penn Treebank: Yes/UH, Please/UH, Thanks/NNS, Thank/VB you, Please/VB, No/DT. This tag will be mapped to UH
PTKA	particle with adjective or adverb am [schönsten], zu [schnell]	Is mapped to RB. (too/RB) Other valid mappings would be more/JJR and most/JJS.
TRUNC	truncated element An- [und Abreise]	Similar constructions are annotated differently in the Penn Treebank. The suspended hyphen is considered a separate sentence element: third -/: and fourth-quarter growth TRUNC will be mapped to (:)
VVFIN	finite full verb [du] gehst, [wir] kommen [an]	Will be mapped to VBZ, VBP or VBD according to number and tense information
VVIMP	imperative, full verb komm [!]	VB
VVINF	infinitive, full verb gehen, ankommen	VB
VVIZU	infinitive with zu, full verb anzukommen, loszulassen	VB
VVPP	past participle, full verb gegangen, angekommen	VBN
VAFIN	finite auxiliary verb [du] bist, [wir] werden	Will be mapped to VBZ, VBP or VBD according to number and tense information
VAIMP	imperative, auxiliary verb sei [ruhig !]	VB
VAINF	infinitive, auxiliary verb werden, sein	VB
VAPP	past participle, auxiliary verb gewesen	VBN
VMFIN	finite modal verb dürfen	MD
VMINF	infinitive, modal verb wollen	MD
VMPP	past participle, modal verb gekonnt, [er hat gehen] können	MD

STTS tag	Usage	Mapping to PTB tags
XY	nonword, with special characters 3:7, H_2O, D2XW3	SYM
$,	comma	(,)
$.	final punctuation . ? ! ; :	(.)
$(	other punctuation marks - [,]()	(:)

PTB tags of the English text have to be mapped as follows (second translation function):

VBG → NN
EX (→ PPER) → PRP
LS → SYM
PDT (→ PIAT) → DT
POS (→ PPOSAT) → PRP$
WDT (→ PRELS) → WP
RBR → RB and RBS → RB comparative and superlative forms of adverbs
$ → NN and # → NN currency symbols

all other tags remain unchanged.

7.4. Code: evolution.pl

```
#!/usr/bin/perl
#
#
#  "Perl terrifies me, it looks like an explosion in an ascii factory."
#                                                  -- svunt(916464), Slashdot

use strict;
use ThsFunctions;
use ThsCorpora;

use constant OTHERLANGUAGE => 0;  # To which other language should English evolve?
                                  # 0 = German/TIGER
                                  # 1 = German/TueBadS
                                  # 2 = Chinese/PTB

use constant SAMPLELANGUAGE => 1; # From where should the 1921 sample sentences be taken?
                                  # 0 = English/PTB    (section 00)
                                  # 1 = other selected language

my (@treebank, @samplesentences, %derivations, @derivationslist);
my $MEnglishModel;  # Model of the modified English language, obtained from leaves of the trees

# A) Load sample sentences (eg sect 00 PTB), POS-Streams
if (SAMPLELANGUAGE == 0) {
  my $wsj = read_sentences_from_file('wsj.mrg-sect0');
  @samplesentences = @{(parse_PTB_file($wsj))[0]};
  print "Read ". (0+ @samplesentences). " English sampling sentences.\n";
}

# B) read GER-Treebank, extract POS-streams, create Katz-model
my $GermanTIGERModel;
if (OTHERLANGUAGE == 0) {
  my @TIGERstreams = @{ get_TIGER_pos_streams() };

  if (SAMPLELANGUAGE == 1) {
    for (my $n = 0; $n <= 1920; $n++) {
      $samplesentences[$n] = shift @TIGERstreams;
    }
    print "Set aside ". (0+@samplesentences) ." German/TIGER sentences for sampling, ". (0+@TIGERstreams)
                                                                                    . " remain.\n";
  } else {
    for (my $n = 0; $n < @samplesentences; $n++) {
      $samplesentences[$n] = prepare_sentence_for_TIGER $samplesentences[$n];
    }
    print "Converted ". (0+@samplesentences) ." sample sentences to German/TIGER format.\n";
  }

  $GermanTIGERModel = create_katz_model(\@TIGERstreams, 10);
}

# D) dito DE-2
my $GermanTueBadSModel;
if (OTHERLANGUAGE == 1) {
  my @TueBadSstreams = @{ get_TueBadS_pos_streams() };

  if (SAMPLELANGUAGE == 1) {
    for (my $n = 0; $n <= 1920; $n++) {
      $samplesentences[$n] = shift @TueBadSstreams;
    }
    print "Set aside ". (0+@samplesentences) ." German/TueBadS sentences for sampling, ".
                                                                    (0+@TueBadSstreams). " remain.\n";
  } else {
    for (my $n = 0; $n < @samplesentences; $n++) {
      $samplesentences[$n] = prepare_sentence_for_TueBadS $samplesentences[$n];
    }
    print "Converted ". (0+@samplesentences) ." sample sentences to German/TueBadS format.\n";
  }

  $GermanTueBadSModel = create_katz_model(\@TueBadSstreams, 10);
}

# D) dito CN
my $ChineseTBModel;
if (OTHERLANGUAGE == 2) {
  my @Chinesestreams = @{ get_CTB_pos_streams() };

  if (SAMPLELANGUAGE == 1) {
    for (my $n = 0; $n <= 1920; $n++) {
      $samplesentences[$n] = shift @Chinesestreams;
    }
    print "Set aside ". (0+@samplesentences) ." Chinese sentences for sampling, ". (0+@Chinesestreams)
                                                                          .  " remain.\n";
  } else {
    for (my $n = 0; $n < @samplesentences; $n++) {
      $samplesentences[$n] = prepare_sentence_for_CTB $samplesentences[$n];
    }
```

```
    print "Converted ". (0+@samplesentences) ." sample sentences to Chinese format.\n";
  }

  $ChineseTBModel = create_katz_model(\@Chinesestreams, 5);
}


# E)
# 1. load WSJ
# 2. extract syntax trees
{
  my $wsj = read_sentences_from_file('wsj.mrg');
  my (@wsjsentences, @wsjwords);
  {
    my @parseptb = parse_PTB_file($wsj);
    @wsjsentences = @{$parseptb[1]};
    @wsjwords = @{$parseptb[4]};
  }

  foreach (@wsjsentences) {
    my (%sentencedescr, @sentencetree, $nodenumber);
    do {

      # Process innermost layer, replace brackets by their descriptors
      s/\(([^()\s]+)\s([^()]+)\)/
        my ($descr, $deriv) = ($1, $2);
        while ($deriv =~ s|\s*-(NONE\|LRB\|RRB)-\s*| |g) {};
        while ($deriv =~ s|(^\.\|\.$\|\s\.\s)| |g) {};
        while ($deriv =~ s|(^\,\|\,$\|\s\,\s)| |g) {};
        while ($deriv =~ s|(^``\|``$\|\s``\s)| |g) {};
        while ($deriv =~ s|(^''\|''$\|\s''\s)| |g) {};
        $deriv =~ s|(^\s+\|\s+$)||go;
        $descr = substr($descr, 0, index($descr, '-')) if (index($descr, '-') > 0);
        $descr = substr($descr, 0, index($descr, '=')) if (index($descr, '=') > 0);
        if ($deriv ne '') {
          $nodenumber++;
          push @{$sentencetree[$nodenumber]}, $descr;
          my $rule = '';
          foreach (split ' ', $deriv) {
            push @{$sentencetree[$nodenumber]}, $_;
            $rule .= ' '. ($sentencetree[$_])->[0] if ($_ > 0);
            $rule .= ' '. $_                       unless ($_ > 0);
          }
          my $sortedderivation = $descr . ' -> [' . (join ' ', sort split ' ', $rule) . ']';
          push @{$sentencedescr{$sortedderivation}}, $nodenumber;
          $derivations{$sortedderivation}++ if (($sortedderivation =~ tr: : :) == 3);
                                                                    # Only list binary derivations
          $descr = $nodenumber;
        }
        ($deriv eq '') ? '-NONE-' : $descr/geo;

      # Process root nodes, brackets that contain only one descriptor
      s/\s*\(([^()\s]+)\)\s*/$sentencetree[0] = $1; " "/geo;
    } until /^\s?$/o;

    $sentencedescr{'tree'} = \@sentencetree;

    if (@wsjwords > 0) {
      $sentencedescr{'orig'} = shift @wsjwords;
      my @sentencewords = split ' ', $sentencedescr{'orig'};
      my (@wordtree, @stack);

      $wordtree[0] = $sentencetree[0];
      for (my $i = 1; $i < @sentencetree; $i++) {
        my @s = @{$sentencetree[$i]};
        for (my $j = 1; $j < @s; $j++) {
          $s[$j] = '.' if ($s[$j] <= 0);
        }
        $wordtree[$i] = \@s;
      }

      push @stack, $wordtree[0];
      do {
        my $node = pop @stack;
        if ($node > 0) {
          push @stack, ($_ > 0 ? $_ : -$node) foreach (reverse @{$sentencetree[$node]});
          pop @stack; # last element is [0] = parent node
        } else {
          for (my $i = 0; $i < @{$wordtree[-$node]}; $i++) {
            if ($wordtree[-$node]->[$i] eq '.') {
              $wordtree[-$node]->[$i] = '-'. (shift @sentencewords);
              $i = 1 + @{$wordtree[-$node]};
            }
          }
        }
      } while (@stack > 0);

      $sentencedescr{'word'} = \@wordtree;
    }
```

```
    push @treebank, \%sentencedescr;
  }
  @derivationslist = keys %derivations;
}

# 3. Generate POS-streams
# 4. Generate Katz-smoothed model

  sub extract_leaves {
    my (@stack, $leafstream);
    my $tree = @_[0];
    push @stack, $tree->[0];
    print $stack[0];
    do {
      $_ = pop @stack;
      if ($_ > 0) {
        push @stack, $_ foreach (reverse @{$tree->[$_]});
        pop @stack; # last element is [0] = parent node
      } else {
        $leafstream .= ' ' . $_;
      }
    } while (@stack > 0);
    return $leafstream;
  }

 sub update_model {
  print "Updating POS-model\n";

  my @POSstreams;
  for (my $n = 0; $n < @treebank; $n++) {
    if ($treebank[$n]->{'pos'} eq '') {
      my $POSstream = extract_leaves($treebank[$n]->{'tree'});
#     print "sentence $n stream : $POSstream\n";
      $POSstream = 'START'.$POSstream.' STOP';

      $POSstream = prepare_sentence_for_TIGER($POSstream)   if (OTHERLANGUAGE == 0);
      $POSstream = prepare_sentence_for_TueBadS($POSstream) if (OTHERLANGUAGE == 1);
      $POSstream = prepare_sentence_for_CTB($POSstream)     if (OTHERLANGUAGE == 2);
      $treebank[$n]->{'pos'} = $POSstream;

      if (defined $treebank[$n]->{'word'}) {
        my $words = extract_leaves($treebank[$n]->{'word'});
        while ($words =~ s/ -/ /go) {};
        $treebank[$n]->{'words'} = $words;
      }
    }
    push @POSstreams, $treebank[$n]->{'pos'};
  }

  $MEnglishModel = create_katz_model(\@POSstreams, 9);
  print "         POS-model updated.\n";
}

# Optimization: Precompute foreign language sentence probabilities
my @samplesentenceprobabilities;
{
  print "Precomputing sentence probabilities...";
  for (my $n = 0; $n < @samplesentences; $n++) {
    my $lpL2;

    $lpL2 = sentence_katz_probability($GermanTIGERModel, $samplesentences[$n], 5)
            if (OTHERLANGUAGE == 0);
    $lpL2 = sentence_katz_probability($GermanTueBadSModel, $samplesentences[$n], 5)
            if (OTHERLANGUAGE == 1);
    $lpL2 = sentence_katz_probability($ChineseTBModel, $samplesentences[$n], 5)
            if (OTHERLANGUAGE == 2);

    $samplesentenceprobabilities[$n] = $lpL2;
  }

  # and free some RAM
  undef($GermanTIGERModel);
  undef($GermanTueBadSModel);
  undef($ChineseTBModel);
  print "done\n";
}

#update_model;
#print_treebank \@treebank;

#
# F)
# 1. Select a binary derivation
sub pick_derivation {
  my $minpick = shift;

  print "Picking one random derivation out of ". (0+@derivationslist). ": ";
  my $pick;
```

```
  do {
    $pick = $derivationslist[rand @derivationslist];
  } while ($derivations{$pick} < $minpick);
  print "$pick  (occurs ". $derivations{$pick} ." times)\n";
  return $pick;
}

# 2. Swap
sub mutate_treebank {
  my $pick = shift;
  for (my $n = 0; $n <= $#treebank; $n++) {
    if (defined $treebank[$n]->{$pick}) {
      foreach (@{$treebank[$n]->{$pick}}) {
          ($treebank[$n]->{'tree'}->[$_]->[1], $treebank[$n]->{'tree'}->[$_]->[2])
        = ($treebank[$n]->{'tree'}->[$_]->[2], $treebank[$n]->{'tree'}->[$_]->[1]);

          ($treebank[$n]->{'word'}->[$_]->[1], $treebank[$n]->{'word'}->[$_]->[2])
        = ($treebank[$n]->{'word'}->[$_]->[2], $treebank[$n]->{'word'}->[$_]->[1])
          if (defined $treebank[$n]->{'word'});
      }
      $treebank[$n]->{'pos'} = '';
    }
  }
}

# 3. Update Katz-smoothed model
#update_model;
#print_treebank \@treebank;

# 4. Test on sample sentences
sub kldivergence {
  my ($sumpos, $sumneg);
  for (my $n = 0; $n < @samplesentences; $n++) {
    my $lpL1 = sentence_katz_probability($MEnglishModel, $samplesentences[$n], 5);
    my $lpL2 = $samplesentenceprobabilities[$n];

    $lpL2 = $lpL1 - $lpL2;

    my $KL = $lpL2 * (exp $lpL1);
    $sumpos += $lpL2 if ($lpL2 > 0);
    $sumneg -= $lpL2 if ($lpL2 < 0);
  }
  my $kld = $sumpos - $sumneg;
  print "KL DIVERGENCE = $sumpos - $sumneg = $kld = avg. ". $kld / @samplesentences ."\n";
  return $kld / @samplesentences;
}


my $kld; # = kldivergence;
my $kll = $kld;

my $generation = 0;
my $evolutionfreedom = 2;
my $evolutionpressure = 0.002;
while ($generation < 2000) {
  ++$generation;

  my $mutation;
  if ($generation < 30) {
    $mutation = pick_derivation 300;
  } elsif ($generation < 100) {
    $mutation = pick_derivation 100;
  } elsif ($generation < 600) {
    $mutation = pick_derivation 10;
  } else {
    $mutation = pick_derivation 1;
  }
  mutate_treebank $mutation;
  update_model;
  my $kld2 = kldivergence;
  if ($kld2 > $kld) {
    $kld = $kll = $kld2;
    print "G$generation - This is much better! keep $mutation, KLD=$kld\n";
  } elsif ($kld2 > $kll) {
    $kll = $kld2;
    print "G$generation - This is better! keep $mutation, KLD=$kld, EF=$evolutionfreedom\n";
  } elsif ($kld2 > $kld - $evolutionfreedom) {
    $kll = $kld2;
    print "G$generation - This is within reason. keep $mutation, KLD=$kld, EF=$evolutionfreedom\n";
  } else {
    print "G$generation - Doesn't help. Reverting...\n";
    mutate_treebank $mutation;
  }
  $evolutionfreedom -= $evolutionpressure;
  $evolutionfreedom = 0 if ($evolutionfreedom < 0);
}
# 6. If closer to foreign language then keep, otherwise revert (or if within threshold do random walk)
# 7. Contd. at F1
```

7.5. Code: ThsCorpora.pm

```
#!/usr/bin/perl
use strict;

package ThsCorpora;
require Exporter;
our (@ISA, @EXPORT);
@ISA = qw(Exporter);
@EXPORT = qw(read_sentences_from_file parse_PTB_file
             get_CTB_pos_streams prepare_sentence_for_CTB
             get_TIGER_pos_streams prepare_sentence_for_TIGER
             get_TueBadS_pos_streams prepare_sentence_for_TueBadS
            );

use constant DEBUG => 0;

sub read_sentences_from_file {
  my ($filename) = @_;
  print "Reading $filename\n";

  # Slurp input file
  $_ = do { local(@ARGV, $/) = "<$filename"; <> };
  print 'Input    : '. length() ." bytes\n";

  # Cleaning up:
  #  - Remove redundant whitespace near brackets
  #  - Collapse consecutive whitespace including line endings
  s/\s\s+/ /g;
  s/\s+\)/\)/g;
  s/\(\s+/\(/g;
  tr/\n/ /;
  print 'Cleaned  : '. length() ." bytes\n";

  return $_;
}

sub parse_PTB_file {
  # Extract sentences:
  #  - Find all substrings with a matching number of opening
  #    and closing parentheses
  my @sentences;
  {
    my $wsj = @_[0];
    my $bufright = 0;
    do {
      my $buffer;
      my $openbrackets = 1;
      my $bufleft = $bufright;

      do {
        $bufright = 1 + index $wsj, ')', $bufright for 1 .. $openbrackets;
        $buffer = substr $wsj, $bufleft, $bufright - $bufleft;
        $openbrackets = ($buffer =~ tr/((/) - ($buffer =~ tr/))/);
      } while ($openbrackets > 0);
      push @sentences, $buffer if ($bufright > 0);

  #   print $bufleft . "-" . $bufright . ": ". $buffer. "\n";
    } while ($bufright < length($wsj)) && ($bufright > 0);
  }
  print "Extracted: ". (0 + @sentences) ." sentences\n";

  my ($nx, %terminal, %terminaldistributions, @POSTagStreams, @sentencetrees, @WordStreams);

  foreach (@sentences) {
    ++$nx;
    print 'Sentence '. ++$nx . ':' . $_ ."\n" if (DEBUG);

    my ($POSStream, $WordStream);
    # Process innermost layer of brackets, containing all terminal derivations
    # Replace whole brackets with their non-terminal descriptor
    # Record distribution of words for POS-tags
    s/\(([^()\s]+)\s([^()]+)\)/$terminal{$1.' -> '.$2}++ if (($1 ne '-NONE-')
                                                          && ($1 ne '-LRB-')
                                                          && ($1 ne '-RRB-'));
                                ($terminaldistributions{$1}->{$2})++;
                                $terminaldistributions{$1.'-count'}++;
                                if (($1 ne ',') && ($1 ne '.')
                                               && ($1 ne '\'\'')
                                               && ($1 ne '``')
                                               && ($1 ne '-NONE-')
                                               && ($1 ne '-LRB-')
                                               && ($1 ne '-RRB-')) {
                                  $POSStream .= "$1 ";
                                  $WordStream .= "$2 ";
                                };
                                $1/geo;

    # Store the stream of seen POS tags to create an n-gram model later on
```

```
    push @POSTagStreams, 'START '. $POSStream .'STOP';
    push @WordStreams, $WordStream;

    # Store the tree structure to create a PCFG later on
    push @sentencetrees, $_;
  }

  return (\@POSTagStreams, \@sentencetrees, \%terminal, \%terminaldistributions, \@WordStreams);
}


# get_CTB_pos_streams
#
# Returns a reference to an array containing all streams of POS tags
# from the Chinese (Penn) tree bank
sub get_CTB_pos_streams {
  my $filename = 'chtb.fid';
  $_ = do { local(@ARGV, $/) = "<$filename"; <> };
  tr/\n/ /; s/\s\s+/ /g;

  my (@sentences, $numsent, $numtags);
  s/<S [^>]*>(.*?)<\/S>/push @sentences, $1; ++$numsent;/geo;

  $_ = <<'CTBMAP';
        AD              RB
        AS              X
        BA              X
        CC              CC
        CD              CD
        CS              IN
        DEC             VBG
        DEG             POS
        DT              DT
        ETC             CC
        FW              FW
        IJ              UH
        JJ              JJ
        LB              X
        LC              IN
        M               NN
        MSP             TO
        NN              NN
        NN-OBJ          NN
        NN-SBJ          NN
        NN-SHORT        NN
        NP              NN
        NR              NNP
        NR-PN           NNP
        NR-SHORT        NNP
        NT              NN
        NT-SHORT        NN
        OD              CD
        P               IN
        PN              PRP
        SB              X
        SP              X
        VC              V
        VE              V
        VP              V
        VV              V

        VA              VA
        DER             DER
        PU              PU
        DEV             DEV
        X               X
CTBMAP
  tr/\n\t/  /; s/\s+/ /g; split ' ';
  my %CTBmap;
  $CTBmap{(pop)} = pop while (@_);

  my @posstreams;
  foreach (@sentences) {
    my $posstream = '';
    s/\(([^ ()]+) [^ ()]+\)/$posstream .= ' ' . $CTBmap{$1} if ($1 ne '-NONE-'); ++$numtags;/geo;
    $posstream .= ' ';
    while ($posstream =~ s/ V DEV / RB /go) {};
    while ($posstream =~ s/ VA DEV / RB /go) {};
    while ($posstream =~ s/ DER / V /go) {};
    $posstream =~ s/ PU $/ . /go;
    while ($posstream =~ s/ PU / : /go) {};
    while ($posstream =~ s/ VA / JJ /go) {};
    push @posstreams, 'START'. $posstream .'STOP';
  }
  print "ChineseTB: read $numsent POS streams with a total of $numtags tags\n";
  return \@posstreams;
}
```

```
# prepare_sentence_for_CTB   $sentence
#
# Replaces PTB tags that have no CTB tags mapped to them
sub prepare_sentence_for_CTB {
  my ($sentence) = @_;

  while ($sentence =~ s/ EX / X /go) {};
  while ($sentence =~ s/ LS / X /go) {};
  while ($sentence =~ s/ RP / X /go) {};
  while ($sentence =~ s/ SYM / X /go) {};

  while ($sentence =~ s/ JJR / JJ /go) {};
  while ($sentence =~ s/ JJS / JJ /go) {};

  while ($sentence =~ s/ RBR / RB /go) {};
  while ($sentence =~ s/ RBS / RB /go) {};
  while ($sentence =~ s/ WRB / RB /go) {};

  while ($sentence =~ s/ NNS / NN /go) {};
  while ($sentence =~ s/ NNPS / NNP /go) {};

  while ($sentence =~ s/ PDT / CD /go) {};

  while ($sentence =~ s/ WDT / DT /go) {};

  while ($sentence =~ s/ MD / V /go) {};
  while ($sentence =~ s/ VB / V /go) {};
  while ($sentence =~ s/ VBD / V /go) {};
  while ($sentence =~ s/ VBN / V /go) {};
  while ($sentence =~ s/ VBP / V /go) {};
  while ($sentence =~ s/ VBZ / V /go) {};

  while ($sentence =~ s/ WP / PRP /go) {};
  while ($sentence =~ s/ WP\$ / PRP /go) {};
  while ($sentence =~ s/ PRP\$ / PRP /go) {};

  while ($sentence =~ s/ \$ / NN /go) {}; # dollar
  while ($sentence =~ s/ # / NN /go) {};  # pound

  return $sentence;
}


# get_TIGER_pos_streams
#
# Returns a reference to an array containing all streams of POS tags
# from the TIGER tree bank
sub get_TIGER_pos_streams {
  print "Loading TIGER...\n";
  my $filename = "TIGER/corpus/tiger_release_aug07.xml";
  $_ = do { local(@ARGV, $/) = "<$filename"; <> };
  tr/\n/ /; s/\s\s+/ /g;

  my ($numsent, $numtags) = (0, 0);
  my @sentences;
  s/<terminals>(.*?)<\/terminals>/push @sentences, $1; ++$numsent;/geo;

  $_ = <<'STTSMAP';
        ADJA-Pos        JJ
        ADJA-Comp       JJR
        ADJA-Sup        JJS
        ADJA-*.*.*.*    JJ
        ADJD-Pos        JJ
        ADJD-Comp       JJR
        ADJD-Sup        JJS
        ADV             RB
        APPR            IN
        APPRART         IN
        APPO            IN
        APZR            IN
        ART             DT
        CARD            CD
        FM              FW
        ITJ             UH
        KOUI            IN
        KOUS            IN
        KON             CC
        KOKOM           IN
        NN-*            NN
        NN-Sg           NN
        NN-Pl           NNS
        NE-*            NNP
        NE-Sg           NNP
        NE-Pl           NNPS
        NNE             NNP
        PDS             DT
        PDAT            DT
        PIS             NN
        PIAT            DT
```

```
        PIDAT           NN
        PPER            PRP
        PPOSS           PRP
        PPOSAT          PRP$
        PRELS           WP
        PRELAT          WP$
        PRF             PRP
        PWS             WP
        PWAT            WP$
        PWAV            WRB
        PAV             RB
        PROAV           RB
        PTKZU           TO
        PTKNEG          RB
        PTKVZ           RP
        PTKANT          UH
        PTKA            RB
        TRUNC           :
        VVFIN-3.Sg.Pres.Ind     VBZ
        VVFIN-3.Sg.Pres.Subj    VBZ
        VVFIN-Pres      VBP
        VVFIN-Past      VBD
        VVIMP           VB
        VVINF           VB
        VVIZU           VB
        VVPP            VBN
        VAFIN-3.Sg.Pres.Ind     VBZ
        VAFIN-3.Sg.Pres.Subj    VBZ
        VAFIN-Pres      VBP
        VAFIN-Past      VBD
        VAIMP           VB
        VAINF           VB
        VAPP            VBN
        VMFIN           MD
        VMINF           MD
        VMPP            MD
        XY              SYM
        $,              ,
        $.              .
        $(              :
STTSMAP
 tr/\n\t/  /; s/\s+/ /g; split ' ';
 my %STTSmap;
 $STTSmap{(pop)} = pop while (@_);

 my $regex = "";
 $regex .= ' '.$_.'="([^">]+)"([^>]*)' foreach ('pos', 'morph', 'number', 'degree', 'tense');

  my @posstreams;
  foreach (@sentences) {
    my $posstream = '';
    s/$regex/$posstream .= ' '; $posstream .= $STTSmap{$1}
                                          || $STTSmap{$1.'-'.$3}
                                          || $STTSmap{$1.'-'.$5}
                                          || $STTSmap{$1.'-'.$7}
                                          || $STTSmap{$1.'-'.$9}
                                          || "unknown:$1:$3:$5:$7:$9 "; ++$numtags;/geo;
    $posstream =~ s/ [.,]//go;  # filter . and ,
    push @posstreams, 'START'. $posstream .' STOP';
  }
  print "TIGER: read $numsent POS streams with a total of $numtags tags\n";
  return \@posstreams;
}


# prepare_sentence_for_TIGER   $sentence
#
# Replaces PTB tags that have no TIGER tags mapped to them
sub prepare_sentence_for_TIGER {
  my ($sentence) = @_;

  while ($sentence =~ s/ VBG / NN /go) {};
  while ($sentence =~ s/ EX / PRP /go) {}; # EX -> PPER -> PRP
  while ($sentence =~ s/ LS / SYM /go) {};
  while ($sentence =~ s/ PDT / DT /go) {}; # PDT -> PIAT -> DT
  while ($sentence =~ s/ POS / PRP\$ /go) {}; # POS -> PPOSAT -> PRP$
  while ($sentence =~ s/ WDT / WP /go) {}; # WDT -> PRELS -> WP
  while ($sentence =~ s/ RBR / RB /go) {};
  while ($sentence =~ s/ RBS / RB /go) {};
  while ($sentence =~ s/ \$ / NN /go) {}; # dollar
  while ($sentence =~ s/ # / NN /go) {};  # pound
  return $sentence;
}


# get_TueBadS_pos_streams
#
# Returns a reference to an array containing all streams of POS tags
# from the TueBadS tree bank
```

```
sub get_TueBadS_pos_streams {
  my $filename = "TueBadS/tuebads.xml";
  $_ = do { local(@ARGV, $/) = "<$filename"; <> };
  tr/\n/ /; s/\s\s+/ /g;

  my ($numsent, $numtags) = (0, 0);
  my @sentences;
  s/<sentence([^>]*)>(.*?)<\/sentence>/push @sentences, $2; ++$numsent;/geo;

  $_ = <<'STTSMAP';
        ADJA    JJ
        ADJD    JJ
        ADV     RB
        APPR    IN
        APPRART IN
        APPO    IN
        APZR    IN
        ART     DT
        BS      SYM
        CARD    CD
        FM      FW
        ITJ     UH
        KOUI    IN
        KOUS    IN
        KON     CC
        KOKOM   IN
        NN      NN
        NE      NNP
        NNE     NNP
        PDS     DT
        PDAT    DT
        PIS     NN
        PIAT    DT
        PIDAT   NN
        PPER    PRP
        PPOSS   PRP
        PPOSAT  PRP$
        PRELS   WP
        PRELAT  WP$
        PRF     PRP
        PWS     WP
        PWAT    WP$
        PWAV    WRB
        PAV     RB
        PROAV   RB
        PROP    RB
        PTKZU   TO
        PTKNEG  RB
        PTKVZ   RP
        PTKANT  UH
        PTKA    RB
        TRUNC   :
        VVFIN   VERB
        VVIMP   VB
        VVINF   VB
        VVIZU   VB
        VVPP    VBN
        VAFIN   VERB
        VAIMP   VB
        VAINF   VB
        VAPP    VBN
        VMFIN   MD
        VMINF   MD
        VMPP    MD
        XY      SYM
        $,      ,
        $.      .
        $(      :
STTSMAP
 tr/\n\t/  /; s/\s+/ /g; split ' ';
 my %STTSmap;
 $STTSmap{(pop)} = pop while (@_);

  my @posstreams;
  foreach (@sentences) {
    my $posstream = '';
    s/<word ([^>]*)pos="([^">]*)"([^>]*)>/$posstream .= ' '; $posstream .= $STTSmap{$2}
                                          || "unknown:$2"; ++$numtags;/geo;
    $posstream =~ s/ [.,]//go;  # filter . and ,
    push @posstreams, 'START'. $posstream .' STOP';
  }
  print "TueBadS: read $numsent POS streams with a total of $numtags tags\n";
  return \@posstreams;
}

# prepare_sentence_for_TueBadS   $sentence
#
# Replaces PTB tags that have no TueBadS tags mapped to them
sub prepare_sentence_for_TueBadS {
  my ($sentence) = @_;
```

```
  while ($sentence =~ s/ VBG / NN /go) {};
  while ($sentence =~ s/ EX / PRP /go) {}; # EX -> PPER -> PRP
  while ($sentence =~ s/ LS / SYM /go) {};
  while ($sentence =~ s/ PDT / DT /go) {}; # PDT -> PIAT -> DT
  while ($sentence =~ s/ POS / PRP\$ /go) {}; # POS -> PPOSAT -> PRP$
  while ($sentence =~ s/ WDT / WP /go) {}; # WDT -> PRELS -> WP
  while ($sentence =~ s/ RBR / RB /go) {};
  while ($sentence =~ s/ RBS / RB /go) {};
  while ($sentence =~ s/ VBD / VERB /go) {};
  while ($sentence =~ s/ VBP / VERB /go) {};
  while ($sentence =~ s/ VBZ / VERB /go) {};
  while ($sentence =~ s/ JJR / JJ /go) {};
  while ($sentence =~ s/ JJS / JJ /go) {};
  while ($sentence =~ s/ NNS / NN /go) {};
  while ($sentence =~ s/ NNPS / NNP /go) {};
  while ($sentence =~ s/ \$ / NN /go) {}; # dollar
  while ($sentence =~ s/ # / NN /go) {};  # pound

  return $sentence;
}
```

7.6. Code: ThsFunctions.pm

```
#!/usr/bin/perl
use strict;

package ThsFunctions;
require Exporter;
our (@ISA, @EXPORT);
@ISA = qw(Exporter);
@EXPORT = qw(get_top_n_results print_array print_places get_random_hash_element create_katz_model
             sentence_katz_probability
             print_treebank
            );
# @EXPORT_OK = qw(funcTwo $varTwo);

sub get_top_n_results {
  my ($hash, $n) = @_;

  my (%counts, @output);
  while (my ($entity, $count) = each(%$hash) ) {
    push(@{$counts{$count}}, $entity);
  }

  my @topcounts = reverse sort {$a <=> $b} keys %counts;

  for (my ($place, $results); ($place <= $#topcounts) && (($n == 0) || ($results < $n)); $place++ ) {
    foreach (@{$counts{$topcounts[$place]}}) {
      if ((($n == 0) || (++$results <= $n)) && ($topcounts[$place] > 0)) {
        push @output, ($topcounts[$place], $_);
      }
    }
  }
  return @output;
}

sub print_array {
  while (@_) {
    printf "%3dx %s\n", shift @_, shift @_;
  }
}

sub print_places {
  my $n = 0;
  while (@_) {
    printf " Place %3d: %3dx %s\n", ++$n, shift @_, shift @_;
  }
}

sub get_random_hash_element {
  my ($probabilitysum, $rndstruct) = @_;
  my $rndnumber = rand($probabilitysum);

  keys %$rndstruct; # reset iterator to beginning of hash

  my $entity;
  while (($entity, my $probability) = each(%$rndstruct) ) {
    $rndnumber -= $probability;
    if ($rndnumber <= 0) {
      last;
    }
  }

  keys %$rndstruct; # reset iterator to beginning of hash

  return $entity;
}


# create_katz_model  \@array   of streams of POS tags
#                    $K        K constant in katz-smoothing
#
# Creates a 1-, 2-, 3-, 4-, 5-gram model with Katz-smoothing
# Returns a reference to a hash containing the counts of all
# n-grams with 1 < n <= 5 as well as individual word counts
sub create_katz_model {
  my ($TagStreams, $K) = @_;

  my %gramcount;
  my %wordcount;
  my $wordtotal;

  # Count occurences
  foreach (@{$TagStreams}) {
    split ' ';
    ++$wordcount{$_} foreach (@_);
    $wordtotal += @_;

    for (my $i = 1; $i < @_; $i++) {
      for (my $n = 1; $n <= 4; $n++) {
```

```
        if ($n <= $i) {
          ++$gramcount{join ' ', @_[$i-$n..$i-1]}->{@_[$i]};
          ++$gramcount{join ' ', @_[$i-$n..$i-1]}->{''};
        }
      }
    }
  }

  my %inversecount;
  foreach my $gc (keys %gramcount) {
    my $len = 2 + $gc=~tr/ / /;
    foreach (keys %{$gramcount{$gc}}) {
      $inversecount{$len}->{$gramcount{$gc}->{$_}}++ if ($_ ne '');
    }
  }

  foreach (keys %wordcount) {
    $inversecount{1}->{$wordcount{$_}}++;
  }

  my %model;
  $model{'gc'} = \%gramcount;
  $model{'wc'} = \%wordcount;
  $model{'ic'} = \%inversecount;
  $model{'N'} = $wordtotal;
  $model{'K'} = $K;
# $model{'p'}  = probability cache
# $model{'lp'} = log probability cache

  return \%model;
}

# katz_probability  \%katz    reference to the katz model
#                   $input    input so far
#                   $output   next element in stream
#
# Returns the probability of '$output' occuring if '$input' was
# the last observation
sub katz_probability {
  use constant KDBG => 0;
  my ($katz, $input, $output) = @_;

  # result cached?
  return $katz->{'p'}->{$input}->{$output}
    if defined($katz->{'p'}->{$input}->{$output});

 print "ug,$output,wc=".$katz->{'wc'}->{$output}.',' if (KDBG);
  # simple probability for unigrams
  return $katz->{'p'}->{$input}->{$output} =
           $katz->{'wc'}->{$output} / $katz->{'N'}
    if ($input eq '');

  my $r = $katz->{'gc'}->{$input}->{$output};

 print "r=$r,gc=".$katz->{'gc'}->{$input}->{''}.',' if (KDBG);
  my $K = $katz->{'K'};
  # if r > k then return traditional estimate
  return $katz->{'p'}->{$input}->{$output} =
           $r / $katz->{'gc'}->{$input}->{''}
    if ($r > $K);

  my $n = 2 + $input =~ tr/ / /;
 print "n=$n," if (KDBG);

  if ($r > 0) {
    my $p = $r / $katz->{'gc'}->{$input}->{''};
    my $rstar = ($r + 1) * $katz->{'ic'}->{$n}->{$r + 1}
                         / $katz->{'ic'}->{$n}->{$r};
    print "observed $n-grams occuring $r+1 times: ".$katz->{'ic'}->{$n}->{$r + 1}."\n" if (KDBG);
    print "observed $n-grams occuring $r   times: ".$katz->{'ic'}->{$n}->{$r}."\n" if (KDBG);
    print "observed $n-grams occuring $K+1 times: ".$katz->{'ic'}->{$n}->{$K+1}."\n" if (KDBG);
    print "observed $n-grams occuring 1 times: ".$katz->{'ic'}->{$n}->{1}."\n" if (KDBG);

    my $d  = ($rstar / $r);
       $d -= ($K + 1) * $katz->{'ic'}->{$n}->{$K + 1}
                      / $katz->{'ic'}->{$n}->{1};
       $d /= 1 - (($K + 1) * $katz->{'ic'}->{$n}->{$K + 1}
                           / $katz->{'ic'}->{$n}->{1});
   print "\nD-P: $p  R*: $rstar  D-D: $d  D: ". ($p * $d)."\n" if (KDBG);
    return $katz->{'p'}->{$input}->{$output} = $p * $d;
  }

  my $backoff;
  $backoff = '' if ($n <= 2);
  $backoff = substr($input, index($input, ' ')+1) if ($n > 2);

  my ($alphat, $alphab, @knownderivations);
 print "Input: $input   Backf: $backoff   Output: $output\n" if (KDBG);
  @knownderivations = keys %{$katz->{'gc'}->{$input}};
```

```
  foreach (@knownderivations) {
    print "  knownderiv: $input -> $_\n" if (KDBG);
    $alphat += katz_probability($katz, $input, $_) if ($_ ne '');
    $alphab += katz_probability($katz, $backoff, $_) if ($_ ne '');
   print "  $input -> $_ ($katz->{'gc'}->{$input}->{$_}, $alphat)\n" if ($_ ne '') && KDBG;
   print "  $backoff => $_ ($katz->{'gc'}->{$backoff}->{$_}, $alphab)\n" if ($_ ne '') && KDBG;
  }

  my ($alpha, $ps);
  $alpha = (1 - $alphat) / (1 - $alphab) if ($alphab != 1);
 print "A = $alpha\n" if (KDBG);
  $ps = katz_probability($katz, $backoff, $output) if ($alpha != 0);
 print "PS ($backoff -> $output) = $ps\n" if (KDBG);

  return $katz->{'p'}->{$input}->{$output} = $alpha * $ps;
}


# sentence_katz_probability  \%hash    containing the katz model
#                            $sentence a stream of POS tags
#                            $depth    max n in n-gram
#
# Returns the probability of the given sentence in the given model
sub sentence_katz_probability {
  my ($katz, $sentence, $depth, $k) = @_;

  my $probability;
  split ' ', $sentence;

  --$depth;
  $depth = @_-1 if ($depth >= @_);

  for (my $i = 1; $i <= $depth; $i++) {
    my $input = join ' ', @_[0..$i-1];
    my $output = @_[$i];

    if (!defined $katz->{'lp'}->{$input}->{$output}) {
      $katz->{'lp'}->{$input}->{$output} = katz_probability($katz, $input, $output);

#      print "ERR: $input -> $output\n"                if ($katz->{'lp'}->{$input}->{$output} == 0);
#      print "ERR!:$input -> $output\n"                if ($katz->{'lp'}->{$input}->{$output} < 0);
      $katz->{'lp'}->{$input}->{$output} = 0.000000875 if ($katz->{'lp'}->{$input}->{$output} <= 0);
      $katz->{'lp'}->{$input}->{$output} = log $katz->{'lp'}->{$input}->{$output};
    }

    $probability += $katz->{'lp'}->{$input}->{$output};
#   print " + PR [ $output | $input ] = $katz->{'lp'}->{$input}->{$output}\n";
  }

  for (my $i = 1; $i < @_ - $depth; $i++) {
    my $input = join ' ', @_[$i..$i+$depth-1];
    my $output = @_[$i+$depth];

    if (!defined $katz->{'lp'}->{$input}->{$output}) {
      $katz->{'lp'}->{$input}->{$output} = katz_probability($katz, $input, $output);
#      print "ERR: $input -> $output\n"                if ($katz->{'lp'}->{$input}->{$output} == 0);
#      print "ERR!:$input -> $output\n"                if ($katz->{'lp'}->{$input}->{$output} < 0);
      $katz->{'lp'}->{$input}->{$output} = 0.000000875 if ($katz->{'lp'}->{$input}->{$output} <= 0);
      $katz->{'lp'}->{$input}->{$output} = log $katz->{'lp'}->{$input}->{$output};
    }

    $probability += $katz->{'lp'}->{$input}->{$output};
#   print " + PR [ $output | $input ] = $katz->{'lp'}->{$input}->{$output}\n";
  }

  return $probability;
}


sub print_treebank {
  my @treebank = @{@_[0]};
  for (my $n = 0; $n < @treebank; $n++) {
    print "Sentence $n:\n";
    foreach (keys %{$treebank[$n]}) {
      print "$_ :: ";
      if (($_ eq 'tree') || ($_ eq 'word')) {
        print "[0] = $treebank[$n]->{$_}->[0]\n";
        for (my $i = 1; $i < @{$treebank[$n]->{$_}}; $i++) {
          print "        [$i] = ";
          for (my $j = 0; $j < @{$treebank[$n]->{$_}->[$i]}; $j++) {
            print '['.$treebank[$n]->{$_}->[$i]->[$j].'] '.($j == 0 ? '-> ' : '');
          }
          print "\n";
        }
      } elsif (($_ eq 'pos') || ($_ eq 'post') || ($_ eq 'words') || ($_ eq 'orig')) {
        print $treebank[$n]->{$_}."\n";
      } else {
        print '[ ';
        print "$_ " foreach (@{$treebank[$n]->{$_}});
```

```
        print "]\n";
      }
    }
  print "\n";
  }
}
```