Diplomarbeit

Unsupervised Morpheme Segmentation using Formal Analogies

Georg Jähnig

Matrikelnr. 721634

Berlin, im Juli 2011

Universität Potsdam, Department Linguistik

1. Gutachter: Dr. Gerlof Bouma
2. Gutachter: Dr. Sebastian Varges

Georg Jähnig:
Unsupervised Morpheme Segmentation using Formal Analogies

Contents

1. Introduction

Morpheme Segmentation is a well known problem in computational linguistics concerning the task of segmenting words into meaningful units (morphemes). It is often one of the first steps in natural languages processing, right after sentence splitting and word tokenization. Splitting a word into morphemes simplifies further steps like Part-of-Speech tagging, since the number of morphemes in a language is usually smaller than the number of word forms (which often may also be infinite). Thus, morpheme segmentation allows further processing steps to use a smaller lexicon or makes them even possible at all.

Unsupervised morpheme segmentation (in contrast to *supervised*) refers to approaches that do not take into account knowledge about the specific language they are segmenting, i.e. these approaches are language-independent. Because of this, such approaches are suitable for morphologically not well-studied languages. And such languages then usually suffer from a low number of speakers, and thus also from a low availability of (digital) data. This makes it hard for quantitatively oriented segmenting approaches to build a model. But also in data of well-studied languages previously unseen words can occur which a segmenter needs to deal with. For such cases, even supervised segmenters may find aspects from unsupervised ones useful. In the annual competition *Morpho Challenge*[1], new approaches of unsupervised morpheme segmentation are regularly competing.

Formal analogies, the other part of this thesis topic, is a notion based on the concept of analogy, having its roots in ancient philosophy. An analogy describes a relation between four elements as "A is to B as C is to D", for example: "an electron is to the nucleus asa planet is to the sun" (Lepage, 1998). Formal analogies can be seen as a subset of analogies where the analogy is based on the *form*. In case of words, this form is described by their characters, e.g. "$lay : lays :: say :: says$" (writing ":" for "is to"

[1] http://www.cis.hut.fi/morphochallenge2010/

and "::" for "as"). The big advantage of formal analogies is that they can be validated automatically by a computer without any knowledge about the language of the words but only by looking at the set and order of their characters. This checking algorithm divides the words into factors and these factors do often resemble morphemes, an observation leading to the idea that finding and checking formal analogies within a given wordlist may produce good morpheme candidates.

In my diploma thesis (Diplomarbeit) I want to study the potential of Unsupervised Morpheme Segmentation using formal analogies. I will examine previous approaches and implement a segmenter based on formal analogies.

The overall structure is as follows: In chapter 2, I will give on overview on unsupervised morpheme segmentation, in chapter 3 I will do the same for formal analogies. Chapter 4 provides a outline on particular segmentation approaches. In chapter 5, I present in detail my approach which is going to be evaluated in chapter 6. The thesis ends with a Conclusion in chapter 7.

2. Unsupervised Morpheme Segmentation

2.1. Problem description

The problem addressed in this thesis can be described as follows, see also Figure 2.1:

- **Input:** Unlabeled text data from a single natural language, tokenized on word level, called *input data*.
- **Output:** The words in the input data split into their morphs, called *segmented data* or *output data*.

In other words: we want to construct a program that takes text data (a corpus or only a list of words) as input and outputs the same words segmented on a morphological level, i.e., into their morphemes, or more accurate *morphs*, as will be pointed out in section 2.3.

We may note that our system always needs *a whole word list at once* as input in order to generate an output. A single word can be segmented only if it was already part of a previous input word list. This is different from other systems that take (labeled) training data as input, create a model which may be then applied on previously unseen words.

An example for the segmenter's input and output is depicted in Figure 2.2.

unlabeled words $\rightarrow$ **segmenter** $\rightarrow$ words split into morphemes

Figure 2.1.: Scheme of the thesis problem

For the given task, we assume the following restrictions, described in detail in the rest of this chapter:

- We split words into their smallest meaning-bearing units, i.e., into *morphs* (and not morphemes).
- We do not recognize allomorphs, i.e., morphs with the same meaning. That means, we step back from finding morphemes, see the following section for clarification.
- We do not categorize the found morphs. We do not provide rules for combining morphs, i.e., a morphology "grammar".
- We allow an unlimited number of morphs within a word.[1]
- Our input data is unlabeled.
- We require the input data to be part of a single language.
- We require our system to work with the same parameters or models for any language.
- We assume a concatenative morphology (respectively, we ignore non-concatenative morphology if present).

In the following, we will take a closer look on these restrictions and on other aspects of morpheme segmentation – just after we pointed out its general relevance and usefulness.

Unlabeled words	**segmented words**
says	say + s
realizes	real + ize + s
toys	toy + s
anti-globalization	anti- + global + iz + ation
feet	feet
tiny	tiny

Figure 2.2.: Example input and output of constructed segmenter

[1]However, as we operate on character level, the natural limit for the number of morphs in a word is the number of its characters.

2.2. Relevance

The relevance of the research field around unsupervised morpheme segmentation can be divided in two parts.

One motivation is a rather practical one: It serves as one of the first steps in computational analysis of natural language. After having detected sentence and word boundaries, word form analysis is an important step to narrow down the often huge amount of word forms to make further processing possible at all: Following steps of natural language analysis like Part-of-Speech tagging, spell checking or machine translation usually require a smaller set of input forms than the number of word types that usually occur in language data.

However, this might also be solved by supervised morpheme segmentation. The second motivation for unsupervised approaches is a scientific one: Having a functioning system acquiring the morphology of languages could also serve as a model for human language acquisition and so contribute to linguistic theory. (Hammarström, 2009, page 2). This is why it is useful to define the input of *unsupervised* morpheme segmentation strictly as *(linguistically) unlabeled and language-independent.*

Having such a functioning language-independent segmenter it is likely to have captured general morphological rules valid beyond many languages – and so to have captured parts of something like an *Universal Grammar.*

Besides, also in practical tasks unsupervised and language-independent approaches could save a lot of work, since one and the same system could be run on data from different even not well-studied languages to obtain a segmentation – without linguists who would need to carefully study the morphology of every single language.

Additionally, even supervised approaches may benefit from ideas of unsupervised ones since they also need to handle previously unseen words. Or one could imagine a two-step morphology acquisition system where morpheme candidates are generated by an unsupervised approach and then are refined by supervised work of a linguist.

2.3. Morphemes and morphs

2.3.1. Definition

As we want to find a segmentation on a morphological level, it is indicated to take a closer look on the concepts and units of morphology. Starting with the entry for **morpheme** in (Bussmann et al., 1996, page 767), we find:

> "Theoretical basic element in structural language analysis, analogous to **phoneme**: the smallest meaningful element of language that, as a basic phonological and semantic element cannot be reduced into smaller elements, e.g., book, three, it, long. Morphemes are abstract (theoretical) units. They are represented phonetically and phonologically by **morphs** as the smallest meaningful, but unclassified, segments of meaning. If such morphs have the same meaning and a complementary distribution or if they stand in free variation, then they are said to be allomorphs."

And for **morph**, we find:

> "The smallest meaningful phonetic segment of an utterance on the level of parole which cannot yet be classified as a representative of a particular **morpheme** (on the level of **langue vs parole**). If two or more morphs have the same meaning but a different distribution then they belong to the same morpheme and are called allomorphs of that morpheme. Thus -able, as in conceivable, and -ible, as in edible constitute two different phonetic representations of an abstract suffix meaning roughly 'able'."

In other words, morphemes can be considered as semantic atoms of language. However, they are a *theoretical construct* as they are summarizing one ore more instances of morphs having the same meaning. Thus, one has to decide which morphs are allomorphs of the same morpheme.

2.3.2. Allomorphy (vs. Synonymy)

According to the definition above, morphs are allomorphs of a morpheme, if

- they share the same meaning and

- have a complementary distribution or stand in free variation.

Based on this definition, it seems reasonable to consider *(drink-)able* and *(ed-)ible* or *color* and *colour* as allomorphs of one morpheme. But looking at the definition again, why could not synonyms like *sick* and *ill* be also allomorphs of one morpheme? Both *sick* and *ill* share the same meaning and stand in free variation. Or asking the question on the background of constructing an unsupervised morphology learner: When *color* and *colour* should be mapped to the same morpheme, why not *sick* and *ill* as well?

It seems as if the definition of allomorphs (and so also of morphemes) needs to be more precise. The following ideas could help in distinguishing allomorphy and synonymy:

- We divide morphemes into **bound** and **free** and allow allomorphy only for bound morphemes. Bound morphemes may not appear alone, they always have to be connected to at least one other morpheme (Kroeger, 2005, page 32). Again, the plural marker *s* as in *(kid-)s* is an example, as it may not appear alone. Then, free morphemes may do so, as the examples of *sick* and *ill* show. However, we should then ask ourselves what to do with phenomena like *color* and *colour*. As both carry the same meaning, where should we "merge" them: on the morphological level or on some extra defined "language variation" level?

- We divide morphemes into **roots** and **affixes** and allow allomorphy only for affixes. Affixes are morphemes carrying grammatical meaning (Kroeger, 2005, page 33). Examples are *(kid-)s* as a marker for plural or *(danc-)ed* as a marker for a verb in the past tense. On the other hand, roots carry lexical meaning, as for instance *kid(-s)* and *danc(-ed)* do. However, if only affixes may be allomorphs, we would exclude *eat* and *ed(-ible)* from being allomorphs of *eat*.

- We may require a **similarity** on the surface form (i.e., the characters) of the morphemes to allow allomorphy. However, this would require us to find a measure for similarity and to set up a threshold manually. We would still have no clear definition and it seems hard to find a similarity threshold low enough to capture the allomorphy of *eat* and *ed(-ible)* without finding too much false positives.

Summing up, it seems to be hard to find a clear distinction between allomorphy and synonymy.

2.3.3. Theoretical construct vs. Useful output

Apparently, it is difficult to find an exact definition of allomorphy and so also of morphemes. While this problem may still remain for theoretical linguists and morphologists, it may be useful to see morpheme segmentation from a more practical view.

As pointed out in section 2.2, one purpose of of morpheme segmentation is to have a more suitable input for the following steps in Natural Language Processing, e.g., for Part-of-Speech tagging, Semantic analysis or Machine translation. One of the constraints for the input of these tasks is usually to narrow down the set of input items. However, this constraint is not sufficient: We could also obtain a smaller set of items by considering every *character* as a morpheme. We still need our output items to be meaning-bearing.

A set of the smallest meaning-bearing units – this definition is already fulfilled by morphs. So we decide to step back and concentrate only on finding morphs. We pass on finding allomorphs of a morpheme. This does not mean that we should give up on the problem of allomorphy. It should just be dealt on a lexical level, as it is apparently so similar to synonymy.

Our still hard enough problem remains to find the smallest meaning-bearing units in a stream of language data.

2.3.4. Phonemes vs. Graphemes

Eventually, we should note that in contrast to the given definitions above, we do not work on *phonemes* but on *graphemes*, as we have much more written data than annotated spoken available. Thus, we assume to be able to capture the morphs on the graphemic level as well as we would on the phonemic one.

2.4. Unsupervised vs. Supervised

As the title of this thesis states, we want to do *unsupervised* morpheme segmentation. Since we have already defined the term *morpheme*, it is also indicated to define *unsupervised*.

It is surprisingly hard to find an exact definition of *unsupervised* in the context of mor-

pheme segmentation. The term with its contrast to *supervised* suggests that there is a binary distinction between two manners of analysis. However, looking at the task definition of Morpho Challenge, an annual competition of unsupervised morpheme analysis approaches, we get the impression of a more vague meaning:

> "The task is the unsupervised morpheme analysis of every word form contained in a word list supplied by the organizers for each test language. [...] Solutions, in which a large number of parameters must be 'tweaked' separately for each test language, are of little interest. This challenge aims at the unsupervised (or very minimally supervised) morpheme analysis of words."[2]

It seems there is a degree of supervision rather than a binary distinction. So apparently when using the term *unsupervised*, one should make clear up to which degree an approach still counts as *unsupervised*. Additionally, the question is *which* knowledge is allowed to be taken into account and which not.

To find a clearer definition, we may look into other areas where the terms *supervised* and *unsupervised* are used – as for instance in machine learning. According to (Jurafsky and Martin, 2000, page 117),

> "[...] a **supervised** algorithm is one which is given the correct answer for some of this data, using these answers to induce a model which can generalize to new data it hasn't seen before. An **unsupervised** algorithm does this purely from data. While unsupervised algorithms don't get to see correct labels for the classifications, they can be given hints about the nature of the rules or models they should be forming. [...] Such hints are called a **learning bias**."

Transferring this definition from machine learning to morpheme segmentation, we could call an approach supervised, if it is taking words split into morphemes as input and inducing a segmentation model out of these. An unsupervised approach is then one using only raw words as they appear in unprocessed language data. Our later decision to use formal analogies is our *learning bias*.

But even unsupervised approaches that only take unlabeled data often use parameters, thresholds and models that are set and selected manually. Eventually, one might ask if requiring the input data to be from a single language is not already a supervised

[2] http://research.ics.tkk.fi/events/morphochallenge2007/rules.shtml

pre-selection of the input.

Besides, our definition coming from machine *learning* does not help us at classifying segmenters that do not learn at all but that are built out of handcrafted, language-specific rules created by the hard work of one ore more linguists finding a model for the morphology of the given language and then formalizing it. Here, the term *supervised morpheme segmentation* seems also convenient.

So in order to clarify the expected input of the approaches I am discussing, I want to give the following definition:

> The input data for unsupervised morpheme segmentation may not contain any linguistic or language-specific knowledge at all – except that all of the input must be part of a single language.

In other words: The same system using the same parameters must work for different languages.

2.5. More Restrictions

2.5.1. Number of morphs

We do not limit the number of possible morphs in a word. So since we operate on character level, every word may contain up to as many morphs as it contains characters.

Imposing a morpheme limit would mean to pick an appropriate number of maximum morphemes per word. Such a number would very probably be language-dependent, thus violating our constraint of language independence. And especially the performance of our segmenter on agglutinative languages would suffer from a morpheme number limit.

2.5.2. Concatenative vs. Non-concatenative

A language may have concatenative or non-concatenative morphology, or even a combination of both. In the first one, "a word is composed of a number of morphemes concatenated together", without any change to the morphemes themselves. An example for concatenative morphology is the singular form of *finger* concatenated with *s* to

fingers in the plural.

In contrast, non-concatenative morphology combines morphemes in a more complex way, intermingling the morphemes. See for example *foot* becoming *feet* in the plural (Jurafsky and Martin, 2000, page 60).

Since our task here concerns only segmenation of words into morphs without any further modification, we restrict ourselves in recognizing only concatenative morphology.

2.5.3. Single words vs. Context

Approaches of unsupervised morpheme segmentation may also differ on the question if their input is only a list of words or if they use a corpus of sentences. In the latter case we may use the context of the words to group them into different categories and then do for a category specific morpheme segmentation.

For the scope of this thesis we assume to have only a list of words as input.

2.6. Summary

This chapter described the relevance of (unsupervised) morpheme segmentation. Its main practical purpose can be seen in the reduction of input forms to simplify further steps of Natural Language Processing, such as Part-of-Speech tagging.

We want to do this in an unsupervised manner: We require the input data to be unlabeled – but from a single language. The segmenter must be able to split words into an unlimited number of morphemes (with the word's character count being the natural limit). We restrict ourselves to recognize only concatenative morphology.

As a crucial point, the problems in the definition of allomorphy were pointed out. It seems as it is an issue rather to be handled in lexical than in morphological analysis. Therefore, we step back from trying to merge allomorphs into morphemes and restrict ourselves to find morphs.

3. Formal Analogies

3.1. Introduction

3.1.1. History

The notion of analogy has a long history in Western thought, an example of mentioning the concept can be already found the work of Plato. The word is based on the "Greek term for geometric or numeric proportions, ratios or symmetries. That is, it referred to the arrangement of two sets of numbers or geometric forms, such that the numbers or forms within each set are related by the same mathematical operator, transformation, or scale (ana logos = 'same logic' or 'according to a ratio')." (Hoffman, 1995, page 5)

It can be basically seen as a relation between four items as in 3.1 where : denotes the just mentioned "same mathematical operator". An example would be the relation in 3.2 where ":" stands for the operation "times two":

$$A : B :: C : D \tag{3.1}$$

$$1 : 2 :: 2 : 4 \tag{3.2}$$

3.1.2. Analogies in language

Analogies in language, which we will be concerned with, appear in the writings of de Saussure, e.g., in (de Saussure, 1959, page 161), giving an example from Latin:

$$oratorem : orator :: honorem : honor \tag{3.3}$$

an analogy, which led the previously used form *honos* change to *honor*. Here, the operator ":" describes an operation changing *oratorem* to *orator* and *honorem* to *honor*.

Figure 3.1.: Analogical relations on two domains (left) and on one domain (right)

It applies the same operation to both words left of it – in this case, it is deleting *em*.

While de Saussure writes then many paragraphs about the role of analogy in language evolution, he provides no exact definition of analogies. For stating whether *oratorem* : *orator* :: *honorem* : *honor* is an analogy and *oratorem* : *orator* :: *honorem* : *emhonor* is not, the reader apparently must rely on his own intuition.

3.1.3. Towards a Definition via Solving Analogical equations

The need for providing an exact formal definition was raised eventually in practice when trying to construct analogy *solvers*, i.e., a program that given the first three items generates the fourth one:

$$oratorem : orator :: honorem : X \Rightarrow X = honor \tag{3.4}$$

Such a program would obviously need an exact algorithm. While there has been a lot of work in the Artificial Intelligence (AI) community (Hall, 1989), one of the first works providing a such one for analogies on words is (Lepage, 1998), giving also the basics of a formal definition I will use in this thesis.

Lepage points out a cornerstone important for solving analogies on words: All of the four items need to be from one domain, that of words (Lepage, 1998, page 2). This is a crucial difference to AI approaches, where the items may be part of two domains, as in the left example of Figure 3.1, taken from (Pirelli and Federici, 1994).

Here, the analogy connects two domains: words in graphemic and in phonemic representation. This leads to the need of three relations, as we have three different combinations of input and output types (i.e., domains and ranges):

1. graphemic to phonemic: f

2. graphemic to graphemic: g

3. phonemic to phonemic: h

The problem is that we cannot unify g and h, as symmetry would suggest, since the type of changes that both functions perform may be very similar to each other but not exactly the same: g is changing the *graphemes* v to s while h is changing the *phonemes* v to s.

In contrast to that, analogies on words operate only within one domain, i.e., that of words, as seen on the right in Figure 3.1 (Lepage, 1998, page 2). Because of this, we come down from three to two relations. We still need two and not one, since the operation from *reactor* to *factor* is a different one as to *reaction*. However, we may swap f and g or swap *reaction* and *factor* and still have a valid analogy – an important property we will need later.

Lepage calls such analogies on words *morphological analogies*, a promising notion for the task of this thesis. Moreover, regarding our goal to do *unsupervised* morpheme segmentation, "morphological analogies can be regarded as simple equations independent of any knowledge about the language in which they are written. This standpoint eliminates the need for any knowledge base or dictionary." (Lepage, 1998, page 2)

3.2. Definition

While Lepage (1998) formulates an algorithm for solving analogical equations on words, we still need a clear formal definition stating the constraints that four words need to fulfill to be an analogy. A such one provide (Yvon et al., 2004, page 7):

> "Analogy is a quaternary relationship defined over $(\Sigma^*)^4$. We say that $(x, y, z, t) \in (\Sigma^*)^4$ stand in analogical relationship, noted $x : y :: z : t$ if and only if $\exists\{(x_i, y_i, z_i, t_i) \in (\Sigma^*)^4\}_{i\in[1,n]}$ s.t.:
>
> $$x_1 \cdots x_n = x, y_1 \cdots y_n = y, z_1 \cdots z_n = z, t_1 \cdots t_n = t \text{ and} \tag{3.5}$$
> $$\forall i \in [1, n], (y_i, z_i) \in \{(x_i, t_i), (y_i, z_i)\}. \tag{3.6}$$
>
> The smallest integer n for which this analogy holds is termed the *degree* of the analogy. "

i	1	2	1	2
x	la	y	a	a
y	la	id	a	b
z	sa	y	b	a
t	sa	id	b	b

Figure 3.2.: Factorization of *lay, laid, say, said* fulfilling the analogy definition

Putting this in prose, it must be possible to factorize every of the four given words in a way that every factor quadruple is of the form (a, a, b, b) or (a, b, a, b). Such a factorization with the example of $say : said :: lay : laid$ is depicted in Figure 3.2, where the first column is of the form (a, a, b, b) and the latter one of the form (a, b, a, b). In terms of our definition, $(y_1, z_1) = (x_1, t_1)$ and $(y_2, z_2) = (t_2, x_2)$[1]. As there is no factorization with less than 2 factors, 2 is also the *degree* of this analogy[2].

With this definition, we can also create a solver[3] for analogical equations like:

$$x : y :: z : X \Rightarrow X = \{t_1, \ldots, t_n\} \tag{3.7}$$

3.3. Properties

Given the previous definition, the following properties hold.

[1] A factor may also be the *empty word*, e.g., for $say : says :: lay : lays$ a possible factorization is $(say, say, lay, lay)(\epsilon, s, \epsilon, s)$.

[2] However, this is not the only minimal factorization, since also $(l, l, s, s)(ay, aid, ay, aid)$ fulfills the analogy equation. This ambiguity will also concern us later.

[3] As indicated, such a solver does not necessarily have an unique solution, there may be even more than one solution – also more than one with the lowest degree. See for instance: $a : ab :: ac : X \Rightarrow X = \{abc, acb, \ldots\}$

3.3.1. Equivalences

Regarding the order of the items, the following equivalences exist (Yvon et al., 2004, pages 6 and 8):

$$
\begin{aligned}
x : y :: z : t &\Leftrightarrow z : t :: x : y && \text{Symmetry of the } as \text{ relation} && (3.8)\\
&\Leftrightarrow x : z :: y : t && \text{Exchange of the means} && (3.9)\\
&\Leftrightarrow y : x :: t : z && \text{Inversion of ratios} && (3.10)\\
&\Leftrightarrow t : y :: z : x && \text{Exchange of the extremes} && (3.11)\\
&\Leftrightarrow t : z :: y : x && \text{Symmetry of reading (1)} && (3.12)\\
&\Leftrightarrow y : t :: x : z && \text{Symmetry of reading (2)} && (3.13)\\
&\Leftrightarrow x : z :: y : t && \text{Symmetry of reading (3)} && (3.14)
\end{aligned}
$$

In short, if we imagine an analogy as a square with the edges as words, we can rotate and flip it as we want – but we cannot start folding or ripping it because: $x : y :: z : t \not\Leftrightarrow x : y :: t : z$.

Furthermore, the following two trivial equations must have an unique solution which they have according to our definition(Yvon et al., 2004, pages 6 and 8):

$$
\begin{aligned}
x : x :: z : X &\Leftrightarrow X = z && \text{Determinism (1)} && (3.15)\\
x : y :: x : X &\Leftrightarrow X = y && \text{Determinism (2)} && (3.16)
\end{aligned}
$$

3.3.2. Character count property

A very important property for practical applications is the character count property (Langlais and Yvon, 2008, page 3) – called there the "T-Trick":

$$x : y :: z : t \Rightarrow \forall c \in \Sigma : |x|_c + |t|_c = |y|_c + |z|_c \tag{3.17}$$

Given the example $[say : said :: lay : laid]$, for instance the count of y in *say* and *laid* must be equal to the one in *said* and *lay*. The same accounts for non-occurring characters, e.g., m – here we just have 0 on both sides of the equation.

3.3.3. Boundary character property

Another property of analogies may be called the boundary character property, presented in Langlais and Yvon (2008, page 4) as the "S-Trick":

$$x : y :: z : t \Rightarrow (x_1 \in \{y_1, z_1\}) \vee (t_1 \in \{y_1, z_1\}) \wedge \tag{3.18}$$

$$(x_{|x|} \in \{y_{|y|}, z_{|z|}\}) \vee (t_{|t|} \in \{y_{|y|}, z_{|z|}\}) \tag{3.19}$$

For instance, looking again at $[say : said :: lay : laid]$, the first character of *say*, *s*, must be also the first character in *said* or *lay*. Or the first character of *laid*, *l*, must be also the first character in *said* or *lay*. In this case, both requirements are fulfilled, only one would be sufficient. The same accounts for the last characters of the words.

These equivalences and properties can be used in reducing the computational problem we encounter when searching for analogies within a large item list. Subsection 3.4.3 will show this problem briefly within the application of analogical learning; in section 5.2, a filter will be described in detail using these properties for the presented segmenation system of this thesis.

3.4. Applications: Analogical learning

3.4.1. Basic idea

Having seen the formal foundation of analogies, we want to take a brief look on common applications of them. A much-used one can be subsumed under the framework of *Analogical learning*, discussed first by (Pirrelli and Yvon, 1999).

Given a supervised learning task of mapping an input item t_I to an output item t_O (possibly yet to be generated) and given only other mappings as the knowledge base, we proceed with the following steps:

1. find one or more triples (x_I, y_I, z_I) such that $x_I : y_I :: z_I : t_I$
2. map x_I, y_I, z_I to their counterparts x_O, y_O, z_O using the knowledge base
3. generate t_O solving the analogical equation $x_O : y_O :: z_O : X \Rightarrow X = t_O$

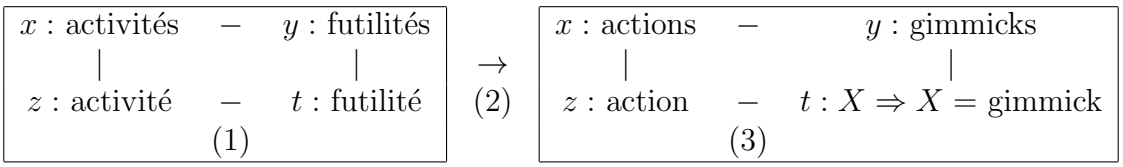

Figure 3.3.: Translating the French word *futilié* to English using a lexicon that only contains *futilités, activités, activité* and their translations. In step (1), we search for appropriate triples and find *(activités, activité, futilités)*. In (2), we map the items from the input to output domain, i.e., we translate them according to their lexicon entries. Eventually, we generate our solution, i.e. our translation in step (3). This is a simplified example taken from (Langlais and Patry, 2007).

3.4.2. Examples

For instance, the input domains may be words and the output domain their Part-of-Speech tags – and we would have a tagger. Or we have words or sentences from one language as input and from another language as output, thus having a machine translation system. Figure 3.3 depicts how the latter one works on an example.

Research work on applications of this framework has been done by Lepage and Denoual who built a machine translation system with this framework based on analogies on sentence-level, Langlais and Patry (2007) used it for translating unknown words, Pirrelli and Yvon (1999) give the example of words as input and their pronunciations as output domain.

Relating to morpheme analysis, the work of Stroppa and Yvon (2005) is worth mentioning: With the title "An analogical learner for morphological analysis", they have implemented and evaluated the above mentioned POS tagger, using a list of tagged words and trying to tag remaining words. In contrast to the task of this thesis, this approach is supervised, since it needs labeled examples as input.

However, one can imagine further applications of this framework: For instance, one could create syntax trees or semantic representations out of sentences given a list of sentences and their trees or representations. Actually, any processing from one linguistic level into another, be it graphemic to phonemic, or syntactic to semantic, could possibly be done using the analogical learning framework.

3.4.3. Problems

Nevertheless, the approach has problems. One of them is computational: While the mapping step is trivial and the generating (solving) step was examined and apparently well solved by many approaches (Lepage, 1998; Yvon, 2003; Yvon et al., 2004), the first finding step is a priori computationally very expensive. It requires to test all possible triples of the input items in the lexicon, i.e., given a lexicon of size n, this means $\binom{n}{3}$ steps. However, Langlais and Yvon (2008) present an interesting approach using the count property to minimize the computational cost. This property will also be used by the filter used for the segmentation system, described in section 5.2.

Other problems related to the machine translation approach by Lepage and Denoual are described in detail by Somers et al. (2009).

3.5. Summary

This chapter gave an overview on formal analogies and in particular, on analogies in language. While the general notion was already mentioned by Plato, de Saussure examined the influence of analogal relations in language change.

We then took a closer look on the definition for formal analogies, as given by (Yvon et al., 2004). Since formal analogies rely only on the form, that is, the surface of the words, just the set and order of characters is relevant, not their meaning. This enables us to find analogical relations without any further (linguistic) knowledge.

We learned about equivalences and two necessary conditions that formal analogies must fulfill: the character count property and the border character property. Both will help us in searching analogies within a large input space.

Finally, we explored briefly a common application of analogies: the analogical learning framework, a generalized approach using analogies for solving various linguistic problems, from Part-of-Speech tagging to machine translation.

4. Previous Work

4.1. Introduction

Now we want to take a look on previous approaches of Unsupervised morpheme segmentation. Various work in this field has appeared in the recent years, probably encouraged by the annual Morpho Challenge conference. An overview of the best participating algorithms along with evaluation results can be found in (Kurimo et al., 2010). A very comprehensive and categorized list of previous works in this area can be found in (Hammarström, 2009, page 16 ff.).

One system is *Morfessor* (Creutz and Lagus, 2005), a purely quantitative approach, that achieves state-of-the-art results and thus often used as a benchmark algorithm as in (Kurimo et al., 2009). Another one is *Linguistica* (Goldsmith, 2006), a bit older than Morfessor which learns morphological paradigms and already uses the notion of analogy – however, not in the formal and generalized way as defined in the previous chapter.

Both approaches use the Minimum Description Length (MDL) framework to evaluate their generated segmentation models, so before describing the approaches, I will give a brief introduction into the MDL framework.

Finally, I will describe *Rali-Ana / Rali-Cof* by Lavallée and Langlais (2009a), a very recent approach very similar to my one, as it uses formal analogies to generate the basic morpheme candidates. However, it does not use MDL for evaluation.

The performance of the three main presented approaches will be compared in section 4.7, the systems will be discussed in the last section 4.8.

But to start, I want to give an overview of the oldest approach: the one of Harris (1955). It looks for peaks in letter successor frequencies to find morpheme border candidates.

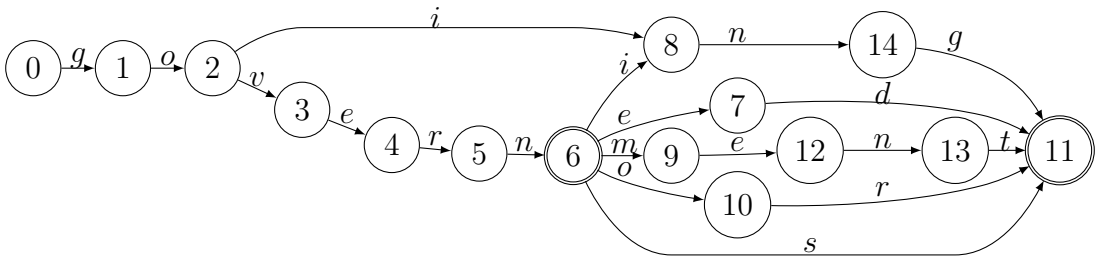

Figure 4.1.: Trie representing the words *governed, governing, government, governor, governs, govern* and *going*. According to Harris' approach, position 6 indicates a morpheme boundary because of its big "fertility".

4.2. First Approach: Peaks in Successor Frequency

One of the first works on Unsupervised morpheme segmentation is the approach of Harris (1955). His idea is also well described in (Goldsmith, 2006, page 5 ff.).

4.2.1. Idea

Harris' approach was basically this: Given a list of words with the same prefix, for instance *gover*, and looking at the very next letter of that prefix, the *successor*, we would ask how many different letters appear as successors. In the case of *gover*, we would probably only find *n*, so the *successor frequency* would be 1. However, looking at the successors of *govern*, we would find words as *governed, governing, government, governor, governs and govern* – 6 different letters, thus a successor frequency of 6. Hence, we would assume a morpheme boundary after *govern*, based on the assumption that a high successor frequency indicates a morpheme boundary.

The idea may be implemented in a word tree structure, called a *trie*, with a root node where all words are starting and edges to following nodes representing the letters of the words. Only one edge with the same letter leaving a node is allowed.

In such a trie, as depicted in Figure 4.1, every node's left path represents a list of words with a common prefix. All leaving edges represent possible successors of the words. Thus, very "fertile nodes" with many edges would indicate morpheme boundaries.

4.2.2. Problems

Goldsmith (2006) notes the problems of that idea. Firstly, the variety of successors is very big at the beginning of words, especially within the first 3 letters, thus making it difficult to find morpheme boundaries there. Additionally, the idea has a bias regarding to vowels and consonants:

> "Since there are more consonants than vowels, and since vowels tend to follow consonants, just as consonants tend to follow vowels, there is a strong tendency for the successor frequency to be larger after a vowel than after a consonant within the first 3 letters of a word, and hence for this algorithm to find a (spurious) morpheme break after any vowel in the first 3 letters of a word."(Goldsmith, 2006, page 6)

Additionally, one must choose a manual threshold for the successor frequency representing a morpheme boundary – or rely on local maxima. Neither works well in English, according to (Hafer and Weiss, 1974), a paper presenting an improved approach based on this one. Also Goldsmith (2006) tries to improve the approach with various heuristics, however eventually stating in (Goldsmith, 2007):

> "[...] such a purely local method does not work, and some more global characteristics of the overall grammar need to be taken into consideration."

An approach considering the "overall grammar" is *Linguistica*, described in section 4.4, which uses MDL for evaluating its candidates. As MDL is also used by *Morfessor*, I will first give a separate introduction into MDL.

4.3. Minimum Description Length

4.3.1. Idea

The Minimum Description Length framework (MDL) was first presented in (Rissanen, 1989) and is also described in (Goldsmith, 2007, page 4 ff.). It deals with the following question:

> Given any data, how can we compress it in a model such that

1. the model can generate as much as possible of the data (is not underfitted) and
2. the model is as small as possible (is not overfitted).

4.3.2. Mathematical Formulation

We can also see the MDL task as trying to find the model that produces the highest probable model given the data – and then applying Bayes' rule (Jurafsky and Martin, 2000, page 146):

$$\arg\max_{model} P(model|data) = \arg\max_{model} \frac{P(data|model) \cdot P(model)}{P(data)} \tag{4.1}$$

Since we are not looking for the actual probability, but only for the model generating it, we may leave out the hard-to-calculate denominator $P(data)$, as the data is constant accross the models being tested, and so its probability would always be the same:

$$\arg\max_{model} P(model|data) = \arg\max_{model} P(data|model) \cdot P(model) \tag{4.2}$$

We may now take the negative log of both sides of the equation. With this operation, we switch from looking for the highest probable model to looking for the optimally encoded (i.e., the shortest) model. The first one is a Maximum a posteriori (MAP), the latter the Minimum Description length formulation of the same problem. Both are equivalent, as shown by (Chen, 1996, page 64).

As the negative log of an increasing number is decreasing, we are now looking for $\arg\min_{model}$:

$$\arg\min_{model} -\log(P(model|data)) = \arg\min_{model}(-\log(P(data|model))) + (-\log(P(model))) \tag{4.3}$$

4.3.3. Meaning of the Formulas

At this point, we can see the two conditions mentioned before under which we want to find the optimal model:

1. $\log(P(model))$ represents the size of the model that should be as small as possible, i.e., no overfitting – I will also use $length(model)$. This value decreases, the less and the shorter morphemes are that we store in the model (usually we start with the maximum with every word being a morpheme). For instance, $[lay, say, s]$ is shorter than $[lay, say, lays, says]$.

2. $\log(P(data|model))$ stands for generating the data as well as possible, i.e., no underfitting – I will also use $length(data|model)$. This value decreases, the higher the probabilities it assigns to the given data.

The sum of both shall be as small as possible, hence *Minimum Description Length*.

4.3.4. Relevance for Morpheme Segmentation

As already mentioned, the idea of using MDL on Morpheme Segmentation is to create a model of morphemes and their distribution, then to try to represent the data using the model while measuring the size of both added.

We may note that thus the basic assumption by using MDL here "is that meaning can be ignored in the process of inferring or inducing the morphological structure of a word or a language" (Goldsmith, 2007, page 9) – an assumption that Goldsmith even calls "controversial". On the other hand, he sees the "reuse of a grammatical object (such as a morpheme, a context, or anything else) is the best kind of evidence we can have of the linguistic reality of the object."(Goldsmith, 2007, page 12).

4.3.5. Evaluation of a Model vs. Search for a Model

We may now have a procedure how to *evaluate* a given model, however, we still need a way how *search* for one, i.e., how to generate a candidate model and then change it according to the temporary result that MDL gives. This is why even if given a sound mathematical model, we still need linguistic theory, or as (Goldsmith, 2007, page 8) puts it [1]:

> "[. . .] the purpose of linguistic theory is to serve as a set of heuristics to help the linguistic scientist come up with a tight, snug grammar, given a set of

[1] using "grammar" for "model"

> data. MDL can determine which of a set of grammars is the best one, given the data; no feasible process can search all possible grammars, so there is no guarantee that another linguist will not come along tomorrow with a better grammar for the data."

In the following sections, we will learn about the search procedures of *Linguistica* and *Morfessor*, two approaches using MDL for evaluation. Basically, both search for redundancy in the data but differ in how this redundancy is expected; thus, how the model is structured.

4.4. Linguistica

4.4.1. Introduction

The approach of *Linguistica* is described in the papers of (Goldsmith, 2001, 2006, 2007). While the first two give the mathematical foundations, the latter one gives the best introduction into the motivation and the general idea. The approach is implemented in software that is freely available on the Internet [2].

Linguistica takes as input an unannotated list of words of a language. Then, it creates so-called *signatures*, each containing a list of stems and suffixes representing the morphological paradigm of a group of words. These signatures can be used to simply split the words into stem and affix, i.e., to solve the task of morpheme segmentation – but they also give us information about the morphological structure of a language, as they group words of the same morphological paradigm.

As the signatures only contain *stems* and *suffixes*, we may get the impression that a word may contain at most two morphemes. But in fact, the stems are allowed to be recursive, they may point not only to an actual stem but also to another signature, that contains a stem and a suffix (Goldsmith, 2001, page 13).

We may note that although a recursive structure of signatures is allowed, the input words are expected to have a limited number of morphs:

> "While some of the discussion is relevant to the unrestricted set of languages,

[2] http://linguistica.uchicago.edu/

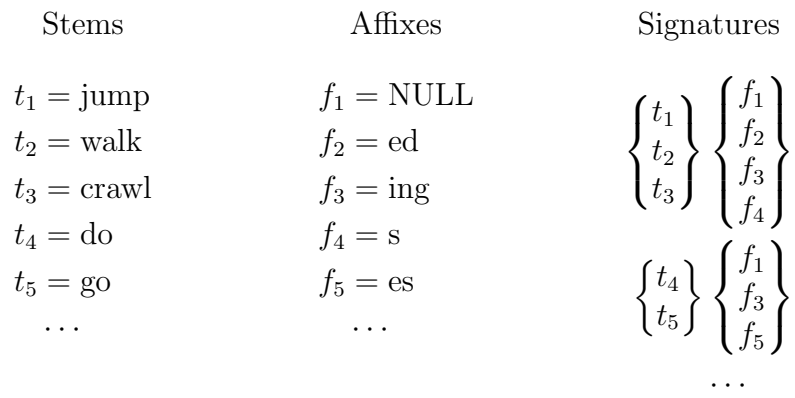

Figure 4.2.: Model structure of *Linguistica*. The first signature captures *crawl, jump* and *walk*, the second one *do* and *go* with their suffixes. From (Goldsmith, 2006, page 4).

> some of the assumptions made in the implementation restrict the useful application of the algorithms to languages in which the average number of affixes per word is less than what is found in such languages as Finnish, Hungarian, and Swahili, and we restrict our testing in the present report to more widely studied European languages. Our general goal, however, is the treatment of unrestricted natural languages." (Goldsmith, 2001, page 1)

An approach dealing with words containing many morphemes is the later published Morfessor, presented in the section 4.5.

4.4.2. Model structure

The model of *Linguistica* tries to find morphological paradigms of stem and suffix alternations. An example structure can be seen in Figure 4.2. It is based on three components (Goldsmith, 2006, page 3):

1. a list of stems $t_{1 \cdots |T|}$,
2. a list of suffixes $f_{1 \cdots |F|}$,
3. a list of signatures with pointers to stems and suffixes $\sigma_{1 \cdots |\Sigma|}$

As we note, only the list of stems and suffixes contain their actual strings, everyone of them only once. The signatures then contain pointers to the entries in the stem and

suffix lists. Every stem (but not every suffix) is linked to exactly one signature. Hence, every stem receives only one pointer from a signature. In contrast, a suffix may receive several pointers from different signatures. There are no different signatures with the same *set* of stem or suffix pointers, as they would be merged into one signature.

A signature may also be pointing not to a *simple* but to a *complex* stem. Such a pointer does not link to an entry in the stem list but to another signature. Thus, the system allows a recursive structure.

4.4.3. Probabilities / Lengths

Now we will learn how the contents of these three components emerge and change, given some data in form of a corpus or word list. We will also take a look on how the system assigns probabilities, hence how we calculate the $length(model)$ and $length(data|model)$. As the model allows various special cases like complex stems (a stem pointing to another stem and suffix), the description here will just give an overview. How the probabilities are calculated in detail, can be followed in (Goldsmith, 2001, page 17).

Notational preliminaries

As already stated, we use T, F, Σ for the set of stems, suffixes and signatures, additionally W for the set of all words in the data. Lowercase letters as t, f, σ, w will be elements of their sets, hence a particular stem, suffix etc. $[W]$ stands for the number of word tokens, $\langle W \rangle$ for types in the data. The same accounts for T, F, Σ etc. (Goldsmith, 2001, page 14). We also need to encode the length of the lists and will do this using the function $\lambda(x)$ where x is the size of a given list, e.g., $\langle T \rangle$.[3]

For calculating the length of a string (within the stem or affix list), we will use the function $L_{typo}(x)$ returning in its simplest definition the length of x times $\log_2 26$, since we assume an alphabet with 26 letters, all with the same probability [4].

[3] The function returns around $\log_2 x$.

[4] A more sophisticated definition assigns different weights to the letters according to their frequency in the data.

Model

Let us start with the probability of the model, i.e., $length(model)$. It is the sum of he stem, the suffix and the signature list – and all three are basically calculated in a very similar way (Goldsmith, 2001, page 15):

$$length(model) = \lambda(\langle T \rangle) + \sum_{t \in T} L_{typo}(t) \tag{4.4}$$

$$+ \lambda(\langle F \rangle) + \sum_{f \in F} L_{typo}(f) \tag{4.5}$$

$$+ \lambda(\langle \Sigma \rangle) + \sum_{\sigma \in \Sigma} L(\sigma) \tag{4.6}$$

In all three cases, we first need the encoded length of the given list calculated by the λ function. Besides, we take the sum of the length of all items of the lists: for stems and suffixes this will be the length of their strings, calculated by the function L_{typo}.

The length of a signature $L(\sigma)$ is a bit more complicated. We will denote $T(\sigma)$ (resp. $F(\sigma)$) as the set of all stems (resp. suffixes) that a signature σ is pointing to. $w(f)$ (resp. $w(\sigma)$) are all words with the suffix f (resp. signature σ).

$$L(\sigma) = \lambda(\langle T(\sigma) \rangle) + \lambda(\langle F(\sigma) \rangle) + \sum_{t \in T(\sigma)} \log(\frac{[W]}{[t]}) + \sum_{f \in F(\sigma)} \log(\frac{[W]}{w(f) \cap w(\sigma)}) \tag{4.7}$$

While this formula looks a bit messy on first sight, it is actually quite symmetric. Firstly, we need the encoded lengths of both the stem and the suffix list of the current signature, i.e., the first two addends. The following two log sums then are the *pointers* to the actual strings in the stem and suffix lists. A pointer gets its length assigned by the relative frequency of the item it is pointing to. As we want the *negative* log probability, we swap numerator and denominator[5].

In the same fashion, the second log sum is constructed. It is the sum of all pointers to suffixes that the current signature is using. But while every stem is associated to exactly one signature, this is not the case for suffixes. Hence, we need a more complicated denominator in the last addend (Goldsmith, 2001, page 15).

[5] since $-\log(\frac{a}{b}) = \log(\frac{b}{a})$

Data given the model

We now come to the second part of the MDL formula, the probability of the data given the model, i.e., $length(data|model)$. It is the sum of the assigned negative log probabilities of all the words in the corpus (Goldsmith, 2001, page 15) [6]:

$$length(data|model) = \sum_{w \in W, w=t+f} [w] \cdot ((-\log P(\sigma(w))) \tag{4.8}$$

$$+ (-\log P(t|\sigma(w))) \tag{4.9}$$

$$+ (-\log P(f|\sigma(w)))) \tag{4.10}$$

Looking at the formula in top-down fashion, we are summing over the lengths of all words. For every word token w, we calculate its length according to the probability

1. of the signature generating w, i.e., $\sigma(w)$,
2. of the stem t of w given the signature $\sigma(w)$,
3. of the stem f of w given the signature $\sigma(w)$.

These probabilities are again all relative counts[7]:

$$\sum_{w \in W, w=t+f} [w] \cdot (\log \frac{[W]}{[\sigma(w)]} + \log \frac{[\sigma(w)]}{[t(w)]} + \log \frac{[\sigma(w)]}{[f(w)]}) \tag{4.11}$$

4.4.4. Search procedure

We now have a method how to measure the "quality" of a given model and thus we are able two compare two different models and tell which one is better. But we still need a method for coming up with an initial model and how to adjust a given model so that it improves. Heuristics for these tasks are explained in the following.

The search procedure consists of three main steps:

1. Proposing an (exactly one) **initial split** for *every* word.
2. Creating **signatures** out of the split words.

[6] In the paper, the middle addend is written as $P(t)$ (instead of $P(t|\sigma(w))$. I assume this to be an error, as in the summary of page 18, the formula is rewritten, with middle addend depended on $\sigma(w)$.

[7] And again in order to incorporate the negation of the log, we swap numerator and denominator.

3. **Adjusting** signatures as long as the MDL improves.

Initial splits

(Goldsmith, 2001, page 19 ff.) explains two heuristics how to come up with various initial splits for every word from which we choose one. The first one considers *all* possible splits of a word into a stem and a suffix. Probabilities for every split are assigned based on a Boltzmann distribution. For every word, the split with the highest probability is taken. This may also lead to splits into prefix and stem, for instance at the word *de-composition*.

This heuristic leads to a split for *every* word, even such ones that actually only consist of one morpheme such as *stomach* and *this*.

The second heuristic operates with a suffix n-gram count. Every word gets an end-of-word marker # added and then, for all 2- to 6-letter n-grams the following likelihood is calculated:

$$\frac{[n_1 n_2 \cdots, n_k]}{\text{Total count of } k\text{-grams}} \log \frac{[n_1 n_2 \cdots n_k]}{[n_1][n_2] \cdots [n_k]} \tag{4.12}$$

This likelihood can be seen as a *weighted mutual information* and is used as an indicator of a n-gram being a candidate suffix. The approach stops at 6 letters as it is not expecting longer suffixes. The top 100 n-grams are chosen to be candidate suffixes and every word is split using these candidate suffixes.

As will be pointed out in the section 4.8, it is these (rather complicated and probably noisy) heuristics for finding initial splits that may be replaced with methods based on formal analogies.

Creating signatures

We do now have the words split into a candidate stem and suffix (sometimes into a candidate prefix and stem) – but they are not yet merged into signatures. To do this, we create a list for every stem with all its suffixes the stem appears with. For instance, for $(walk, walks, walked, walking)$ we get the signature $(NULL, s, es, ing)$. All stems with the same suffix list are merged together into one signatures.

Having done this, we discard all signatures with only one stem and with only one suffix.

What remains, are so-called *regular signatures* with at least two stems which according to (Goldsmith, 2001, page 21) are a "very good approximation and constitute a good initial analysis".

Adjusting signatures

Even while being a good first approximation, the model still contains some flaws at this stage. These and the various heuristics dealing with them are described in the following. For all tweaks of the model, "the entire description length of the corpus is recomputed under the alternative analysis; the reanalysis is adopted if and only if the description length decreases." (Goldsmith, 2001, page 25).

Collapsing of two suffixes into one In the analysis of English, we will probably find suffixes as *ings* and *ments*, which both are actually only combinations of two also existing suffixes. To find them, every suffix is checked if it can be created out of to other ones. In the mentioned examples, we will very probably find *ing* / *ment* and *s*.

Suffix contains stem-final material If a letter in a language appears very often at the end of stems – as *t* does in English – it may be wrongly analyzed as part of the suffix. *ted.ting.ts* and *ted.tion* (generated from words like *existed, existing, exists, acted, action*) are candidate suffixes in English related to this problem. To find such analyzes, the suffix list of every signature is checked if all suffixes begin with the same letter, and if they do, this very letter is moved to the stem.

Spurious signatures Looking still at English, there are many words ending with *s* and in many of them *s* is a morphological suffix, marking the third person of a verb or the plural of a noun. But this is not the case for all words, as the example of the very word *spurious* shows. On the other hand, every word ending with *ness* uses these four letters as a suffix. As apparently the length of the suffix plays an important role, the system uses this measure to find out whether a candidate suffix is also an actual suffix for this words.

Allomorphy Linguistica even contains some heuristics to find allomorphy. Alternations like *win(s)* and *winn(er)(ing)* are found by checking every pair of stems whether they be related by a simple substitution process, like adding, deleting or replacing a letter.

4.5. Morfessor

4.5.1. Introduction

The approach of *Morfessor* is described first in (Creutz and Lagus, 2002), another short description is presented in (Creutz, 2003), and a more extended description is given in (Creutz and Lagus, 2005). Like Linguistica, Morfessor uses the distributional measure of MDL to evaluate its model, however the later paper of (Creutz and Lagus, 2005) replaced it by a Maximum a posteriori (MAP) formulation. But both are considered equivalent and produce the same results. As the following description is based on the latter paper, also the MAP formulation will be used here.

Morfessor is free software released under the GNU GPL[8].

Like Linguistica, Morfessor takes a word list (with optional word frequencies) as input. It then only outputs the same word list with the words split into morphs. So the system does not try to find allomorphs of a morpheme nor to categorize the found morphs in any way, nor does it group words with the same morphological paradigm.

In contrast to Linguistica, Morfessor was designed to work especially with words containing many morphemes, i.e., with agglutinative morphology. As the authors formulated it:

> "Many algorithms proceed by segmenting (i.e., splitting) words into smaller components. Often the limiting assumption is made that words consist of only one stem followed by one (possibly empty) suffix [...]. This limitation is reduced in (Goldsmith, 2001) by allowing a recursive structure, where stems can have inner structure, so that they in turn consist of a substem and a suffix. Also prefixes are possible. However, for languages with agglutinative morphology this may not be enough. In Finnish, a word can consist of

[8]available at http://www.cis.hut.fi/projects/morpho/

lengthy sequences of alternating stems and affixes." (Creutz, 2003)

Thus, Morfessor allows an unlimited number of morphs per word. Additionally, it states to be language-independent, however it allows to set parameters that may be language-specific.

Various versions of Morfessor have been released, however, we will take a look only at the so-called *Baseline models* implemented in Morfessor 1.0.

4.5.2. Model

Unlike Linguistica, Morfessor does not differentiate morphs into stems and suffixes. All morphs are simply items in a list of morphs of length M: $\mu_1 \cdots \mu_M$.

A word may contain an infinite number of morphs, the word's character count being the natural limit.

4.5.3. Probabilities

As already stated, Morfessor uses the Maximum a posteriori formulation (MAP) instead of the Minimum Description Length (MDL). Both differ only minimally, to put it short, MAP does not take the log of the probabilities, thus we calculate directly on the probability values, thus we are not adding but multiplying, thus we are looking for the model generating the maximum probability, not the minimum length.

$$\arg\max_{model} P(model|data) = \arg\max_{model} P(data|model) \cdot P(model) \tag{4.13}$$

Data given the model

We now have got two factors whose product should be as big as possible. Let us begin with the first one, $P(data|model)$. This probability is computed straightforwardly as

relative counts

$$P(\mu_i) = \frac{f_{\mu_i}}{N} \tag{4.14}$$

$$P(data|model) = \prod_{j=1}^{W} \prod_{k=1}^{n_j} P(\mu_{jk}) \tag{4.15}$$

where $\mu_{1\cdots M}$ are morphs of M types and N tokens in a corpus of W words, every word containing n morph tokens.

Model

The second factor, $P(model)$ is calculated differently in various Morfessor versions. In the later, non-Baseline versions, the model is a joint probability of a lexicon and a grammar, the latter modelling simple morphotactics, i.e., word inner syntax. The Baseline models we are now talking about only use the lexicon which is a joint probability of the properties of every morph μ in the set of M morphs.

$$P(model) = P(lexicon) = M! \cdot P(properties(\mu_1), \cdots, properties(\mu_M)) \tag{4.16}$$

$M!$ accounts for the fact that there may be that many different orderings of a set of M items – which share the same probability. As properties, only the morph frequency $P(f_\mu)$ and its string $P(s_\mu)$ are considered, which are assumed to be independent of each other.

$$P(properties(\mu_1), \cdots, properties(\mu_M)) = P(f_{\mu_1}, \cdots, f_{\mu_M})) \cdot P(s_{\mu_1}, \cdots, s_{\mu_M}) \tag{4.17}$$

In the simplest Baseline model, the frequency property is modeled implicitly, thus assigning all morphs the same probability [9]

$$P(f_{\mu_1}, \cdots f_{\mu_M}) = 1/\binom{N-1}{M-1} \tag{4.18}$$

The string property is about the length and the letters of the morph. In the simplest, i.e., implicit modeling, the probabilities of the different morph strings are assumed to be independent of each other. Furthermore, the string probability derives from the

[9] Note the difference between f_μ used in the previous equation and $P(f_\mu)$ used here.

probabilities of its letters which are assumed to be independent of each other:

$$P(s_{\mu_1}, \cdots s_{\mu_M}) = \prod_{i=1}^{M} P(s_{\mu_i}) \tag{4.19}$$

$$P(s_{\mu_i}) = \prod_{j=1}^{l_{\mu_i}} P(c_{ij}) \tag{4.20}$$

l_{μ_i} contains the length of a given morph and $P(c_{ij})$ are the individual probabilities of every letter computed from the data by their relative frequencies observed there.

Implicit modeling is done with a special *end-of-morph* character # at the end of each morph string. The probability of seeing a morph with length l is thus:

$$P(l) = [1 - P(\#)]^l \cdot P(\#) \tag{4.21}$$

which will be also part of the string property probability. As we see, this probability decreases with an increasing l, we are thus having an exponential distribution where the longer the morph, the smaller its length probability.

4.5.4. Search procedure

After initializing the model with every word type being a morph, a greedy search algorithm is run over the morph lexicon in a different random order on every iteration. Every morph is split recursively into two parts, the system calculates the overall probability for every split and keeps the split with the highest overall probability. The iterations are repeated until the increase of the overall probability is lower than a given threshold.

4.6. Rali-Ana and Rali-Cof

4.6.1. Introduction

The last previous work described here is the recently published approach of Lavallée and Langlais (2009a). There exists another, slightly shorter paper (Lavallée and Langlais, 2009b). The approach uses *formal analogies* as described in chapter 3. Unlike the two

other previous works described here, it does not use MDL nor any other measure based on information theory for evaluation.

There is no implementation available online.

The system works as follows: Like the other presented ones, it takes an unannotated list of words as input and segments the words into their morphemes. Depending on the subsystem, it may do this based only on formal analogies or based on so-called *cofactor rules* (short c-rules), created out of the analogy factorizations.

The system does not categorize morphs and allows an arbitrary number of morphs per word.

4.6.2. Basic Idea

Given a word list, the algorithm searches for quadruples of words that form an analogy. For every found analogy, the factorization with the lowest degree is calculated. For instance, checking the quadruple $(say, says, lay, lays)$ will find that it is an analogy and possibly also tell us the factorization with the lowest degree of 2: $(say, lay)(\epsilon, s)$.

Based on this factorization that looks so promising for our task of finding morphemes, we would continue to create rules out of the factors or simply use the factors for segmenting the words into morphs. Both subsystems I will describe in the following.

As we see, the basic assumption of the approach is that the factors of formal analogies are good candidates for linguistic morphemes.

However, we should already note a main flaw at this point: There may be more than one factorization with the lowest degree. For the example above, there are two other ones: $(sa, la)(y, ys)$ and $(s, l)(ay, ays)$. The paper does not state how the system deals with such ambiguities, in a personal email conversation with me, the author Phlilippe Langlais states that one of them is picked randomly. Looking ahead, in my approach I tried to solve this problem by creating a factorizer that outputs all factorizations with the minimum degree.

4.6.3. Computational Problem

As will be discussed further in chapter 5, the search for analogies suffers from a computational problem: Analogies are quadruples of words, so in order to find analogies, one has to find appropriate quadruples. Potentially, there are $\binom{n}{4}$ quadruples in a list of n words, given for example an average lexicon of $n = 100,000$ words, we would need to check around $4 \cdot 10^{18}$ quadruples. Performing that many checks is infeasible for today computers – and probably also for tomorrow ones.

Therefore, the authors use an algorithm for filtering quadruples described in (Langlais and Yvon, 2008). However, this is an algorithm for searching suitable triples for a given word in order to find analogies involving this word. So it solves a problem slightly different to the present one: Here we have one big list of words and want to find analogies within this list.

Unfortunately, the particular adaptation of this algorithm on the current problem is not explained in the paper. In a personal email conversation, Philippe Langlais stated that the algorithm is applied sequentially on every word in the input list, with a certain timeout. All analogies that were found for this word within this time are collected.

Because of this time limit, a lot, possibly valid quadruples are not checked (Lavallée and Langlais, 2009a, page 6), thus many analogies are not found, thus many words are left as is (i.e., unsegmented) in the final analysis. This impacts recall in the pure analogical systems, as we will see in the results section 4.7. To cope with that fact, so-called cofactor based systems were developed that try to extract morphological rules out of the found analogies, in order to apply them on words not covered by analogies.

4.6.4. Pure analogical subsystems

There are two pure analogical subsystems called *(Rali-)Ana-Seg* and *(Rali-)Ana-Pair*.

Ana-Seg

Ana-Seg simply uses the analogy factors as morphs and segments the words according to them. However, there may be different factorizations of a word if it participates in more than one analogy (which is virtually nearly always the case). For instance *abolishing*

may be part of the two different analogies, leading to two different factorizations, thus to two different segmentations:

$$abolishing : polishing :: abolish : polish \Rightarrow (ab, p)(olishing, olish) \quad \Rightarrow ab + olishing \tag{4.22}$$

$$abolishing : polishing :: doing : do \Rightarrow (abolish, do)(ing, \epsilon) \qquad \Rightarrow abolish + ing \tag{4.23}$$

Hence, we need a way to pick a segmentation from the proposed ones. *Ana-Seg* simply uses frequency: The most frequent segmentation is taken as the segmentation for the word.

Ana-Pair

The second pure analogical system Ana-Pair does actually not solve the segmentation task. Instead, for every word it creates a list of other words that share a morph with the given word. For instance, for the word *disabled*, the list could contain *enable* (sharing *able*), *disprove* (sharing *dis*) and *walked* (sharing *ed*).

Such a list suits the evaluation method of Morpho Challenge: Since the task does not require the contributors to assign a certain set of morph categories, the contribution systems are evaluated if they recognize words sharing the same morphemes.

As stated, this approach does not solve the task of this thesis as its output are not segmented words. Therefore, we will omit it in the evaluative discussion.

4.6.5. Cofactor-based subsystems

Generating c-rules

The other two subsystems described in (Lavallée and Langlais, 2009a) are *(Rali-)Cof-First* and *(Rali-)Cof-Graph*. Both calculate rules out of the cofactors, so-called *c-rules*. For instance, based on an analogy like *cordial : cordially :: appreciative : appreciatively*, we get the cofactors $(cordial, appreciative), (\epsilon, ly)$. The latter one we may use to construct a rule like $[*ly \rightarrow *\epsilon]$ with $*$ marking possible context. As there is a $*$ at the

beginning but not end, the rule may be only applied at the end of the words [10]. The left hand side cannot be shorter than the right hand side, hence $|\alpha| \geq |\beta|$.

Based on the latter restriction, we know that successive applications of a c-rule will tend towards a shorter word and we assume that shorter words are also morphologically simpler.

Filtering c-rules

The number of c-rules generated as described is usually enormous. Some of them like $[anti - * \rightarrow \epsilon]$ capture a morphological relationship but there are also many spurious ones found like $[ka* \rightarrow \epsilon]$. To filter out the latter ones, simple frequency does not help. Instead, a measure called *productivity* is used and calculated as follows:

$$productivity(\text{c-rule}) = \frac{valid\,results}{applicable} \tag{4.24}$$

applicable is the number of cases where a rule can be applied, i.e., where the left hand side matches a word in the data. *valid results* then is the number of cases where application of a rule also leads to a word that is in the data. For instance, we may apply $[ka* \rightarrow \epsilon]$. to a word like *karl* but we would not find the result *rl* in our data, hence it would be not valid. On the other hand, we may apply $[anti - * \rightarrow \epsilon]$ to *anti-alcoholic* and would probably also find *alcoholic* in the data.[11].

With productivity, the $anti - *$ rule outweighs the $ka*$ with score of 0.94 against only 0.24. Note that a rule with a low weight is not dropped. It can still be part of a path with several rules within the Cof-Graph system, as explained in the following.

Building a Word-relation tree / graph

Using the c-rules, we are constructing a Word-relation tree (see Figure 4.3, respectively a graph, where the words are connected based on the c-rules. This is the point where the two subsystems Cof-First and Cof-Graph differ: In Cof-First, words can only be connected if there exists a c-rule that leads from one word in the data to another word in the data. In Cof-Graph, a connection is also possible if two words are connected via

[10] Consider $[appreciative* \rightarrow cordial*]$ as the rule created out of the other cofactor.
[11] Examples taken from (Lavallée and Langlais, 2009a, page 4)

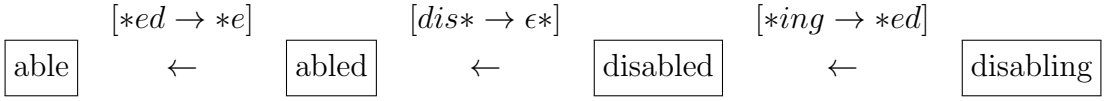

Figure 4.3.: Word-relation tree connecting the words *able, abled, disabled* and *disabling* with c-rules generated out of co-factors. Taken from (Lavallée and Langlais, 2009a, page 5)

more than one rules with intermediate words that are not part of the data. See for instance the connection from *disabled* to *able*: In Cof-First, there must exist *abled* in the data in order to make the connection.

The score of a connection between two words with several c-rules in Cof-Graph is calculated similar to (Hidden) Markov Models or Weighted Finiste-state machines over the probability semiring: along one path, the productivity scores are multiplied, the overall score is the sum of all path scores.

Obtaining morphs from the tree / graph

Based on the word-relation tree or graph, we now can obtain morphs. In a tree, the word of the root node is a morpheme for itself. All words of other nodes get their morphs from the c-rule pointing to their father merged with the morphs of their father. For instance, the morphs of *disabled* are $[dis, able, ed]$, since the word points to *abled* with the rule $[dis* \rightarrow *\epsilon$ and *abled* already is constructed of the morph set $[able, ed]$ (based on the rule $[*ed \rightarrow *e]$ and the root node *able*).

Note that using this method we are also able to capture a bit of allomorphy: Since c-rules are not only deleting but also replacing, *abled* is analyzed with the morphs $[able, ed]$.

4.7. Comparison of Performance

The challenge of this section is now to compare three different approaches. Truly valid results on that can only be found by carrying out experiments on the same dataset. However, the only results I have found related to this question are

- a comparison of *Morfessor* and *Linguistica* in the first publication about Morfessor (Creutz and Lagus, 2002, page 7 ff.)

Method	Correct	Incomplete	Incorrect
Morfessor	49.6%	29.7%	20.6%
Linguistica	43.1%	24.1%	32.8%

Table 4.1.: Morfessor vs. Linguistica: Estimate of accuracy of morpheme boundary detection based on visual inspection of a sample of 2500 Finnish word tokens, simplified from (Creutz and Lagus, 2002, page 9).

- a comparison of *Morfessor* and *Rali-Ana/Rali-Cof* in the Proceedings of Morpho Challenge 2009 (Kurimo et al., 2009).

I will present these numbers in the next subsections.

4.7.1. Morfessor vs. Linguistica

As the motivation for developing Morfessor was to have a segmentation algorithm for languages with agglutinative morphology – for which Linguistica is not suitable for – the publication paper (Creutz and Lagus, 2002) did especially a comparison with Linguistica on the same data. It was done based on three categories:

1. "correct and complete segmentation (i.e., all relevant morpheme boundaries were identified),
2. correct but incomplete segmentation (i.e., not all relevant morpheme boundaries were identified, but no proposed boundary was incorrect),
3. incorrect segmentation (i.e., some proposed boundary did not correspond to an actual morpheme boundary)." (Creutz and Lagus, 2002)

These categories can be seen as variants of the Accuracy measure. While the experiments were carried out on a Finnish and an English corpus, unfortunately only the numbers of Finnish are presented in detail, as shown in Table 4.1[12]. While we can see significant improvements in Finnish, the authors report that the two systems applied to English "were rather equal in performance." Generally, Morfessor "leaves very common word forms unsplit, and often produces excessive splitting for rare words. [...] Linguistica,

[12]In the original paper, Morfessor's search procedure was implemented in two ways, recursive and sequential. As recursive achieved better results and as it is also the implementation of the published software, only its values are presented here.

	Morfessor			Rali-Ana			Rali-Cof		
Language	Pr.	Rc.	F	Pr.	Rc.	F	Pr.	Rc.	F
Vow. Arabic	86.87	4.90	9.28	91.30	2.83	5.49	95.09	1.50	2.95
English	74.93	49.81	59.84	64.61	33.48	44.10	68.32	46.45	55.30
German	81.70	22.98	35.87	61.39	15.34	24.55	67.53	34.38	45.57
Finnish	89.41	15.73	26.75	60.06	10.33	17.63	74.76	26.20	38.81
Turkish	89.68	17.78	29.67	69.52	12.85	21.69	48.43	44.54	46.40

Table 4.2.: Morfessor vs. Rali-Ana/Rali-Cof: Collected performance numbers of Morfessor and Rali-Ana/Rali-Cof at Morpho Challenge 2009 in Precision, Recall and F-measure. All values in percent.

on the other hand, employs a more conservative splitting strategy, but makes incorrect segmentations for many common word forms." (Creutz and Lagus, 2002, page 8-9)

4.7.2. Morfessor vs. Rali-Ana/Rali-Cof

A comparison of the performance of Morfessor and the Rali-Ana/Rali-Cof systems is presented in (Kurimo et al., 2009, page 11 ff.), its numbers are presented in Table 4.2. The results were discussed in detail in (Lavallée and Langlais, 2009a).

Starting with a comparison between the two analogical systems, we can see that Rali-Cof outperforms Rali-Ana in almost all languages. Explaining this is trivial: Rali-Ana is directly tied to the actual analogies found by the system – because of the high computational complexity discussed above, only a small subset of the analogies has been identified which impacts recall. As the authors put it

> "Since analogical learning somehow relies on the pattern frequency to identify morphemes, several valid morphemes might be overlooked due to their low frequency in the training set. The high precision supports this hypothesis as it shows that what the systems manage to learn from the lexicon is valid but that only a few morphological phenomenon could be identified." (Lavallée and Langlais, 2009a, page 8).

Overcoming this problem by the generalization based on c-rules is apparently a good way since the recall is literally jumping up from Rali-Ana to Rali-Cof.

But compared to Morfessor, the analogical systems have a lower precision. This is a bit

surprising, since the "search procedure" of the analogical systems restricted to found analogies seems a lot more conservative as Morfessor's free search among all possibe splits.

On the other hand, the Recall of Rali-Cof is much higher in three languages and so also its F-measure compared to Morfessor's. Seemingly, the analogical approach manages to find morphs that Morfessor misses because they are not "seeded" – as explained further in subsection 4.8.2.

4.8. Discussion

4.8.1. Linguistica

The very first question I had when reading about Linguistica was if it uses formal analogies, since the last paper (Goldsmith, 2007) had such a promising title, "Morphological Analogy: Only a Beginning". I was quite surprised that it does not and it does not even have a reference on the work of them, although several works on formalizing formal analogies like (Yvon et al., 2004) were published before.

In fact, the heuristic-based analogies found by Goldsmith fulfill the formal analogy definitions from section 3.2 if we assume a signature with two stems and two suffixes. Signatures with more stems or affixes can be seen as formal analogies by considering all possible quadruple instances in the same way. For example, the signature 4.25 can be considered as the three analogies from 4.26 to 4.28.

$$\begin{Bmatrix} \text{jump} \\ \text{walk} \end{Bmatrix} \begin{Bmatrix} \text{NULL} \\ \text{s} \\ \text{ing} \end{Bmatrix} \tag{4.25}$$

$$jump : jumps :: walk : walks \tag{4.26}$$

$$jump : jumping :: walk : walking \tag{4.27}$$

$$jumps : jumping :: walks : walking \tag{4.28}$$

However, as we see these captured analogies are restricted to two factors at most. In my opinion, the system could benefit a lot from replacing its heuristic for finding initial

splits with methods based on formal analogies, for instance on analogical factorizations. Additionally, it is probably unnecessary to divide morphemes into stems and affixes and thus increasing the complexity of the model. Such a division is also not useful even when applying to only European languages as Linguistica priorly wanted to: Even in non-agglutinative German, there are many words containing two or more stems, as for instance *Wohnungsschlüssel (apartment key)* or *alkoholhaltig (alcohol-containing)*.

It seems that Linguistica's model structure with dividing morphs into stems and suffixes is only caused by its suffix-oriented heuristic for finding initial splits. Furthermore, it is this suffix-orientation that will probably not allow Linguistica to capture morphologically rich languages as Finnish or (vowelized) Arabic which uses a lot of infixation.

4.8.2. Morfessor

My main reason for taking a look on Morfessor was its popularity, as it is even used as a benchmark algorithm in every Morpho Challenge competition. Its model has a convincingly simple structure, imposing as few as possible constraints on the morphology of the data it is analyzing. Basically, "a morph is just a morph", without any further label. To "survive" during the search process, it has to prove to appear well distributed within the data. One could name this approach as *purely distribution-based*.

However, the main flaw of Morfessor is its search algorithm. It works sequentially, analyzing and updating the model always based on only one word at a time. Because of this, it may miss well distributed morphs, as the example in Figure 4.4 shows.

In the left run, Morfessor fails to find to morphs *xyz* and *xyz*, although they appear in every word. The reason for this is: When splitting *aaxyz*, Morfessor does only look at the word itself and the current model. As the initial model only contains the unsplit input words, the system does not see an advantage in splitting into the morph-*xyz* at any point, as it temporarily would always lead to an *increase* of the description length.

This is different in the right data which contains *xyz* explicitly. Thus, this morph is also in the initial model, leading to a split in all -*xyz* words, and then also in all -*zyx* words.

One can describe this behavior as if Morfessor needs morphs to be "seeded" in the input data: in order to start the "chain of discovering morphs", at least one morph must appear somewhere in the data as a single word.

Input:	Output:
aaxyz	aaxyz
bbxyz	bbxyz
ccxyz	ccxyz
aazyx	aazyx
bbzyx	bbzyx
cczyx	cczyx

Input:	Output:
aaxyz	aa + xyz
bbxyz	bb + xyz
ccxyz	cc + xyz
aazyx	aa + zyx
bbzyx	bb + zyx
cczyx	cc + zyx
xyz	xyz

Figure 4.4.: Two Morfessor inputs and outputs: One fails in finding `xyz` and `zyx` (left), the other succeeds because `xyz` is also part of the input.

Summarizing the two MDL approaches, Linguistica and Morfessor, both use a very good evaluation and development measure – MDL – but have flaws within the search procedure. Looking ahead, the opposite is true for Rali-Ana/Rali-Cof.

4.8.3. Rali-Ana/Rali-Cof

Finally, the analogy based systems present an interesting search algorithm. As morphology is the alternation of strings (e.g., affix morphs) around same string contexts (e.g. stem morphs), analogy seems the to be the natural way of finding them.

However, it is the part of *finding them* that is computationally very expensive and thus very time consuming. This a really huge drawback of the approach.

In order to find analogies for every word in the data anyway, the authors applied a filtering algorithm that apparently filtered too much:

> "It is important to note that because we computed only a small portion of all the analogies, there are many words that these two systems do not treat adequately. In particular, the words for which no analogy is identified are left as is in the final solution, which clearly impacts recall." (Lavallée and Langlais, 2009a, page 6)

So the problem of low recall in the Rali-Ana system is apparently not caused directly by its basic concept. It is maybe solvable, or at least improvable, by applying faster machines or by improving the filtering process.

Another question that could be raised is if sparsity of data might be a problem: It could be possible that even when checking for all possible quadruples for analogies, still many words are left uncovered. However, the experiments that were carried out with the filter and segmenter of this thesis showed that in all languages, around 90% of the words could be analyzed with a pure analogical approach (see Section 6.2.2). This finding contradicts the assumption of analogical data-sparsity. So applying an exhaustive analogy search on much more data (using better computers or a better filter algorithm) could probably lead to a better recall.

Eventually, it is interesting that to cope with the problems of the pure analogical system, the authors chose generating co-factor rules for abstraction, instead of already used approaches based on distribution, like MDL. One could use the analogical factors as morph candidates for an MDL-based evaluation – and thus merge Morfessor and Rali-Ana together. The authors themselves suggest further research based in this direction:

> "Our approach on the pure analogical system was quite simple and could gain from using some information theory metric such as perplexity to calculate the probability of the different segmentations." (Lavallée and Langlais, 2009a, page 9)

4.9. Summary

Summarizing the two MDL approaches, Morfessor and Linguistica, both use a very good evaluation and development measure – MDL – but have flaws within the search procedure. Quite the opposite is true for the analogical systems (Rali-)Ana-Seg and partially for (Rali-)Ana-Cof.

In this thesis, I want to improve the analogical methods of Lavallée and Langlais (2009a) by introducing a novel exhaustive filtering and analogical factorization approach. Detailling and evaluating this approach will be the topic of the remaining chapters. The interesting further step of combining formal analogies and MDL, as suggested by Lavallée and Langlais (2009a) above, will have to remain the subject for future work.

5. Proposed Segmenter

5.1. Introduction

The approach I will propose and describe in the following is heavily based on (Lavallée and Langlais, 2009a), and there on the subsystem *Ana-Seg*. As a key change, I have constructed a Finite-state transducer calculating not only one but *all* factorizations of an analogy with a minimal degree. Additionally, I have created a filtering algorithm that generates *all* quadruple candidates from the given word list. We thus will get an exhaustive recall of all analogies and all factors – thus all morph candidates (based on formal analogies) – in the input data.

In the current chapter, I will describe this approach in detail. These steps of the new segmenter will concern us:

1. Pre-Filtering all quadruples extracted from the data if they can form an analogy.
2. Checking if the quadruples are in fact analogies, if yes: outputting all factorizations with the minimal degree.
3. Using the factors as morph candidates for segmentation, evaluating different approaches.

5.2. Filtering Quadruples

5.2.1. Motivation

Before generating segmentations from analogies, we first need to find the analogies. In order to find all analogies in a given word list of n words, we would need to check all possible quadruples in the list which would mean about $\binom{n}{4}$ checks. For only $n = 10,000$

Input	Output
lay	$(lay, say, lays, says)$
say	$(lay, say, laid, said)$
laid	...
said	
says	
lays	
foo	
bar	
...	

Figure 5.1.: Input and Output of quadruple filter.

we thus would need to test 416,416,712,497,500 (i.e., around 416 trillion quadruples) which would not be computationally feasible.

In order to narrow down the input space of the actual analogy checker, I will present a filtering algorithm that exploits properties of formal analogies, mainly the character count property (Section 3.3.2). This will bring down the complexity to $\binom{n}{2}$, leading to around 50 million steps for $n = 10,000$.

Using these properties, the filtering algorithm is similar to the one used by (Lavallée and Langlais, 2009a), described in (Langlais and Yvon, 2008). The main differences will be pointed out in the Summary section of this chapter – looking ahead, its main distinction is that it generates all candidates available, and not only a subset.

To sum up, the module explained in the following will take a list of words as input and output quadruples that probably form an analogy, as depicted in Figure 5.1.

5.2.2. Recalling the Character count property

As stated, the filter relies mainly on the character count property of formal analogies, as presented in Subsection 3.3.2. Let us recall it:

$$\forall c \in \Sigma : |x|_c + |t|_c = |y|_c + |z|_c \tag{5.1}$$

The property says that in an analogy, the count of every character within the words must fulfill the above equation. Given the example $[say : said :: lay : laid]$, for instance

the count of y in *say* and *laid* must be equal to the one in *said* and *lay* – and in fact it is 1 on both sides. The property even applies for non-existent characters like m in this analogy: In this case, the count is 0 on both sides.

5.2.3. Character count property as Analogy predictor

To make use of the property, we represent all words in the data as vectors of character counts: For every character, we note its count within the word.

$$lay : [a : 1, l : 1, y : 1] \tag{5.2}$$
$$say : [a : 1, s : 1, y : 1] \tag{5.3}$$
$$laid : [a : 1, d : 1, i : 1, l : 1] \tag{5.4}$$
$$said : [a : 1, d : 1, i : 1, s : 1] \tag{5.5}$$

Now we create pairs of all words and their vectors by adding the vectors:

$$(lay, say) : [a : 2, l : 1, s : 1, y : 2] \tag{5.6}$$
$$(lay, laid) : [a : 2, d : 1, i : 1, l : 2, y : 1] \tag{5.7}$$
$$(lay, said) : [a : 2, d : 1, i : 1, l : 1, s : 1, y : 1] \tag{5.8}$$
$$(say, laid) : [a : 2, d : 1, i : 1, l : 1, s : 1, y : 1] \tag{5.9}$$
$$(say, said) : [a : 2, d : 1, i : 1, s : 2, y : 1] \tag{5.10}$$
$$(laid, said) : [a : 2, d : 1, i : 2, l : 1, s : 1] \tag{5.11}$$

We may note now that there are only two equal vectors within all pairs: The vector of $(lay, said)$ and the one of $(say, laid)$. Each of the two vectors can be seen as one side of the character count property equation. As it happens, these four words form an analogy.

We should although not forget that the character count property is just a necessary but not sufficient condition, since it does not take the order of the characters into account. We may imagine a word like *ayl*, forming a character count vector with *said* that would also look exactly the same like the above vector for $(say, laid)$. However, the quadruple $(ayl, said, say, laid)$ does not form an analogy.

Nevertheless, a passed character count filter is apparently a strong predictor for a quadruple being an analogy. As can be seen in detail in section 6.2.1, roughly 25% to 50% of the

quadruples in both English and German fulfilling this condition also form an analogy.

5.2.4. Creating and grouping character count vector pairs

Based on this insight, we want to group all pairs according to their vectors: Pairs with the same vector will be summarized in the same group.

$$[a:2, d:1, i:1, l:1, s:1, y:1] : (lay, said), (say, laid) \tag{5.12}$$

$$[a:2, d:1, i:1, l:2, y:1] : (lay, laid) \tag{5.13}$$

$$[a:2, l:1, s:2, y:2] : (lay, says), (say, lays) \tag{5.14}$$

We then need to check for analogies only within a group – and every pair of pairs only once, with no matter what order, since the equivalences explained in Section 3.3.1 apply.

We will note that the vast majority of groups contains only one pair, thus no further check will be required there, since an analogy needs 4 words, i.e., 2 pairs. In the groups created with the evaluation datasets of this thesis, around 87-90% of the vector pair groups contain only one pair.

So the creation of pairs remains the computationally crucial point. Since we need to create all possible pairs, we come up to a complexity of $\binom{n}{2} \approx \frac{n^2}{2}$ steps, with n being the number of words in the data.

5.2.5. Additional filtering with boundary character property

After having found candidates via the just described method, we additionally test if the candidates fulfill the boundary character property, as described in 3.3.3. From the already generated candidate quadruples, this test deletes on average 40-50% quadruples.

Input		Output
$(lay, say, lays, says)$	$\rightarrow$	$[(lay, say)(\epsilon, s);$
		$(la, sa)(y, ys);$
		$(l, s)(ay, ays)]$
$(lay, say, laid, said)$	$\rightarrow$	$[(la, sa)(y, id);$
		$(l, s)(ay, aid)]$
$(ayl, say, laid, said)$	$\rightarrow$	$\emptyset$
$\ldots$	$\ldots$	

Figure 5.2.: Input and Output of FSM analogy factorizer.

5.3. FSM for finding and factorizing formal analogies

5.3.1. Introduction

The next step is the heart of the system: given the just before calculated quadruples as input, we want to calculate *all* factorizations of the minimal degree, as depicted in Figure 5.2. This will be done by a weighted finite-state machine (FSM), being more specific, by a transducer, constructed using on the work of (Yvon et al., 2004). Such a transducer may be implemented in any FSM framework, always producing the same results.

This FSM will be based on a novel representation of formal analogies, which might be also called *encoding*. It will allow us to represent analogical quadruples in a short form and might be also useful in another applications concerning formal analogies.

5.3.2. Idea

Recalling the analogy definition

The idea shall be described using the example of the input quadruple $(lays, laid, says, said)$. In order to check if this quadruple forms an analogy, we may recall Section 3.2: "It must be possible to factorize every of the four given words in a way that every factor quadruple is of the form (a, a, b, b) or (a, b, a, b)." Such factorizations with the minimal number of factors for the given example can be seen in Figure 5.3.

Words	Patterns			Min. Factorizations				Min. Atomic Factorization			
lays	a	a	a	la	ys	l	ays	l	a	y	s
laid	b	a	a	la	id	l	aid	l	a	i	d
says	a	b	a	sa	ys	s	ays	s	a	y	s
said	b	b	a	sa	id	s	aid	s	a	i	d
pattern type:	1	2	4	2	1	2	1	2	4	1	1

Figure 5.3.: Factorizations for *lays : laid :: says : said*.

Pattern types

We may require the factors to contain only single characters (with allowing the empty character ϵ). Such atomic factor quadruples must be still of the form (a, a, b, b) or (a, b, a, b). We call these forms *patterns* and give them a number according to the number of a counting from the start:

- 1 for (a, b, a, b),
- 2 for (a, a, b, b) and
- 4 for (a, a, a, a).

The last pattern is a special case of the two previous ones and can be called a "subset" of them, since factors of pattern 4 match also pattern 1 *and* 2. In Figure 5.3, an example of such a pattern is the middle-a of the four words. It is these factors that cause different minimal factorizations – since, as seen here, we may attach this atomic factor to the previous type-2-pattern or the following type-1-pattern. Thus, the existence of ambiguous minimal factorizations is directly related to the existence of type-4 patterns.

At this point, we can also see another way of calculating the degree: It is the minimum number of times where we need to switch from one pattern type to another plus 1 – and pattern type 4 appearing as type 1 or 2.

In our example, we need to start with 2, can then take the 4 still as 2 but must then switch to 1, hence one switch. Another option is to start again with 2, take the 4 as 1, i.e., make the switch already here and continue with 1. Again, we made only one switch, adding 1 makes a degree of 2. Again, we can see that if there are no type-4 patterns, no ambiguity can occur.

(a, b, a, b)	$=$	`1ab`	*lays : laid :: says : said* $=$
(a, a, b, b)	$=$	`2ab`	$(l, l, s, s)(a, a, a, a)(y, i, y, i)(s, d, s, d) =$
(a, a, a, a)	$=$	`4aa`	`2ls4aa1yi1sd`

Figure 5.4.: Analogy encoding, definition (left) and example (right).

Encoding analogies

Based on the definition of pattern types, we may define an encoding for analogies. We may have noted that although we are dealing with four words and thus with *quad*-ruples of characters, there are always at most two different characters per quadruple at play (assuming the four words form an analogy). Hence, we may encode analogy quadruples according to the mapping in Figure 5.4.

4-way alignment

The given example of *lays : laid :: says said* is kind of trivial, as the four words have the same length and fulfill the required patterns on their "natural" alignment by just writing them one below the other. This is different with *lay : laid :: say : said* for instance, where we first need to find the right alignment.

This alignment must not be confused with the popular alignment based on edit-distance (Levenshtein, 1966). We can draw the difference based on the example in Figure 5.5.

An edit-distance alignment is usually an alignment between two words. The aligned tuples of characters may be of the form (a, a) for same characters, $(-, a)$ for insertions, $(a, -)$ for deletions and (a, b) for substitutions. All tuples except the first usually have

Alignments	Edit-distance						Analogical								
							w	o	r	d	x	-	-	-	-
	-	w	o	r	d	x	-	-	-	-	x	w	o	r	d
	x	w	o	r	d	-	w	o	r	d	y	-	-	-	-
							-	-	-	-	y	w	o	r	d
pattern type:							1	1	1	1	2	1	1	1	1

Figure 5.5.: (2-way) shortest alignment based on edit-distance (left), 4-way shortest alignment based on analogical patterns (right).

a cost assigned. The shortest alignment is thus the alignment with the smallest cost.

In contrast, an analogical alignment is a 4-way alignment, allowing quadruples of the above described patterns. Hence, there may also be no analogical alignment – if the four words do not form an analogy.

We will use a finite-state machine to check if such an analogical alignment exists and, if there are more than one, to calculate the shortest alignment.

5.3.3. FSM Definition

Previous work

As stated, the Weighted FSM we want to construct, will be based on (Yvon et al., 2004). In contrast to our task, this paper concerns the solution of analogical equations, i.e., given for instance *lays : laid :: says : X*, calculating the solution $X = said$. But since such a task also involves finding the *optimal* solution, thus the solution with the lowest number of factors, we may use many of their "ingredients" for our FSM. One of them is the way of calculating the 4-way alignment of four words in (Yvon et al., 2004, page 14 ff.).

Notational preliminaries

In the following description, the same concepts of finite-state machines will be used as in the previous work. Basically, acceptors represent languages (i.e., sets of words) and

transducers represent relations of languages. The following notations apply:

$\{a, b\}$: an acceptor representing the language with the words a and b

$A, \{\cdots\}$: capital letters or curly brackets denote acceptors

$A \times B$: a crossproduct of two acceptors A and B forms a transducer, mapping from A to B

$A \cup B$: union of the acceptors A and B

$A \cap B$: intersection of the acceptors A and B

$A \circ B$: composition of transducers or acceptors A and B

A^* : star closure of the acceptor A

$\{\cdots\}\langle w \rangle$: an acceptor weighted with weight w

$Bestpath(A)$: a function returning the path with the minimal weight

$P_2(A)$: a function extracting the output language of a transducer

Overview

We assume to be given the four input words as simple finite-state acceptors. We want to construct a FSM that

1. prepares the input acceptors,
2. intersects the prepared acceptors, thus creates an acceptor representing all analogical alignments,
3. filters out unnecessary paths of the intersected acceptor leaving a path representing the unique minimal factorization based on the presented encoding above, and finally
4. generates an acceptor containing all minimal factorizations.

Preparing input words

We want to format the input acceptors in a way that they just need to be intersected and the resulting acceptor then contains all analogies as its paths. We need two things to consider.

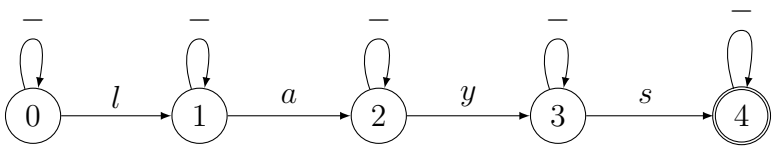

Figure 5.6.: FSM accepting *lays* with an arbitrary number of spaces at any point of the word.

The first concerns the alignment in general: If two or more words shall be aligned, and especially if they are of different length, we must be able to insert spaces, otherwise an alignment can be impossible. So we will use $-$ as the symbol representing a space and allow an infinite number of spaces to be inserted at any point of the words. The resulting automaton for the word *lays* is depicted in Figure 5.6.

Secondly, we want the alignment to be analogical – where every quadruple has the form of an analogical pattern. We can make this sure by trying to *encode* the words as an analogy, as it has been defined in subsubsection 5.3.2. If it succeeds, the given four words must be an analogy.

By looking at the encoding definition again, we note that, for instance, characters of the first word *lays* only appear as the left character in the encoded version. This is similar with the characters of last word *said*, only appearing as the right character in the encoding.

In Figure 5.7, we see an overview of possible encodings of a character. Hence, we will substitute every transition of the acceptor in Figure 5.6 with their corresponding acceptors from Figure 5.7. We define $\overline{\Sigma}$ as the set containing Σ and $-$.

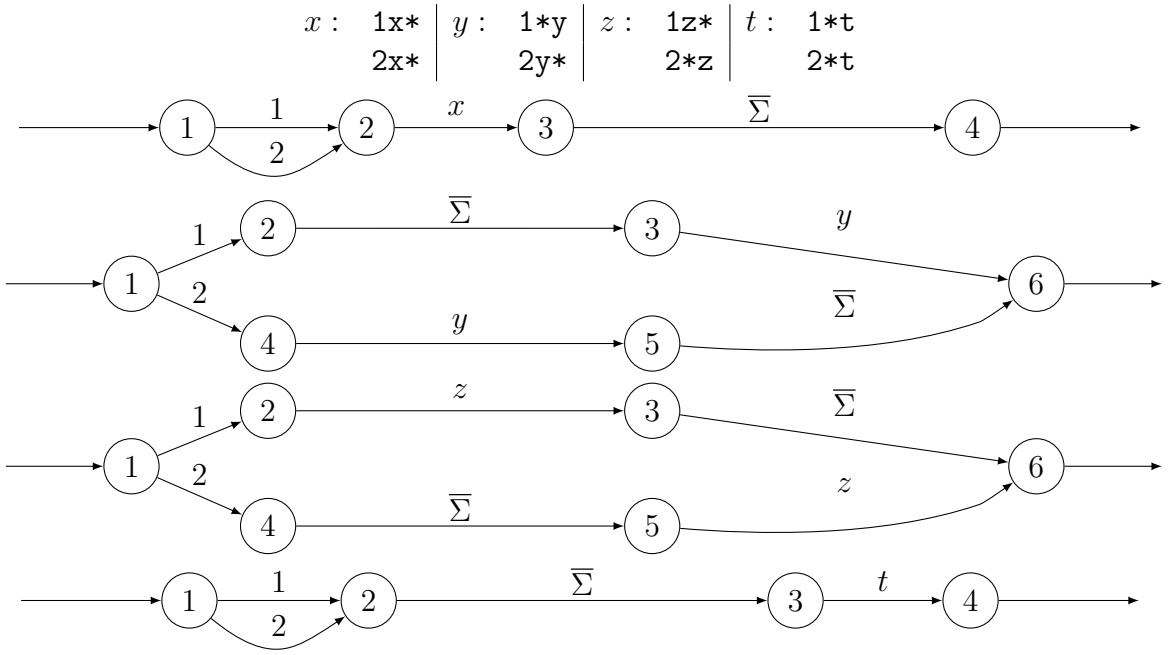

Figure 5.7.: Possible encodings of a character given the analogy $X : Y :: Z : T$ with x, y, z, t as corresponding characters ($*$ indicating any character plus space) – and their corresponding acceptors ($\overline{\Sigma}$ indicating any character plus space).

Finally, we come up with the following operations for preparing the input words:

$$\alpha_- = \{\epsilon\} \times \{-\} \tag{5.15}$$

$$\alpha_1 = \{\epsilon\} \times \{1\} \tag{5.16}$$

$$\alpha_2 = \{\epsilon\} \times \{2\} \tag{5.17}$$

$$\alpha_{\overline{\Sigma}} = \{\epsilon\} \times \overline{\Sigma} \tag{5.18}$$

$$X_{prepared} = P_2\left(X \circ (\Sigma \cup \alpha_-)^* \circ \left((\alpha_1 \cup \alpha_2) \cdot \overline{\Sigma} \cdot \alpha_{\overline{\Sigma}}\right)^*\right) \tag{5.19}$$

$$Y_{prepared} = P_2\left(Y \circ (\Sigma \cup \alpha_-)^* \circ \left(\left(\alpha_1 \cdot \alpha_{\overline{\Sigma}} \cdot \overline{\Sigma}\right) \cup \left(\alpha_2 \cdot \overline{\Sigma} \cdot \alpha_{\overline{\Sigma}}\right)\right)^*\right) \tag{5.20}$$

$$Z_{prepared} = P_2\left(Z \circ (\Sigma \cup \alpha_-)^* \circ \left(\left(\alpha_1 \cdot \overline{\Sigma} \cdot \alpha_{\overline{\Sigma}}\right) \cup \left(\alpha_2 \cdot \alpha_{\overline{\Sigma}} \cdot \overline{\Sigma}\right)\right)^*\right) \tag{5.21}$$

$$T_{prepared} = P_2\left(T \circ (\Sigma \cup \alpha_-)^* \circ \left((\alpha_1 \cup \alpha_2) \cdot \alpha_{\overline{\Sigma}} \cdot \overline{\Sigma}\right)^*\right) \tag{5.22}$$

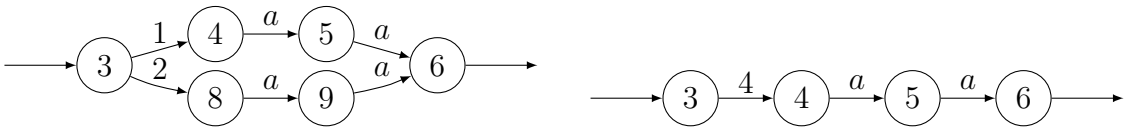

Figure 5.8.: Sequence of intersected acceptor before and after merging factors with ambiguous pattern type.

Intersecting and filtering

Given the four prepared input word acceptors, we just need to intersect them:

$$A = X_{prepared} \cap Y_{prepared} \cap Z_{prepared} \cap T_{prepared} \tag{5.23}$$

We thus obtain an automaton A

- containing all analogical alignments of the four words
- or being empty if the words do not form an analogy. In this case, the work is done at this point.

In the first case, we now need to filter all these alignments to get the best one, that is, the one with the lowest number of factorizations. Solving this task is quite straightforward: We simply put a weight on every factor, that is: on every occurrence of the markers 1 and 2 – and then search for the path with the lowest weight. Thus, the shortest path will win. This means that non-minimal alignments, as shown for example Figure in 5.9 will be filtered out, making the automaton remarkably smaller and so speeding up the following steps.

However, there may be more than one path with the same minimal weight. In this case, we want all of these alignments, thus, all of the minimal factorizations.

The following part of the algorithm may be omitted if we assume to have a $Bestpaths(A)$ function in our FSM framework, leaving *all* all paths in the acceptor a with a minimum weight. Even the introduction of pattern type 4 could be left out, we would just calculate switches between 1 and 2. However, this function is not available in the framework I am using (Hanneforth, 2009), nor in the popular OpenFST (Allauzen et al., 2007).

We therefore apply a transducer preserving all factors that are marked with both 1 *and* 2 and mark them as 4, according to our definition for encoding analogies in subsubsec-

	-	l	a	y	s
	-	l	a	i	d
	s	-	a	y	s
	s	-	a	i	d
pattern type:	2	2	1/2	1	1

Figure 5.9.: Non-minimal analogical alignment in A, filtered out in $A_{filtered}$.

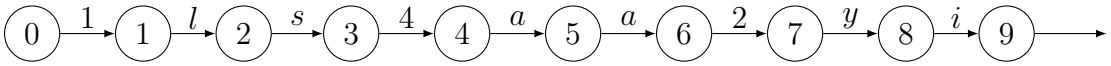

Figure 5.10.: Beginning sequence of filtered acceptor $A_{filtered}$ for *lays : laid :: says : said*, containing the single shortest path.

tion 5.3.2 As we remember, it is these ambiguous factors that create multiple minimal factorizations. By merging into one factor we "disambiguate" them. See Figure 5.8 for an acceptor sequence before and after applying this transducer.

Note that factors marked with 4 do not have a weight attached (in contrast to factors marked with 1 or 2), thus they will always be the shorter to paths that use 1 or 2 for the very same position.

So finally, we come up with these operations and obtain an acceptor containing one path representing all minimal factorizations of *lays : laid :: says : said* depicted in Figure 5.10:

$$\beta_4 = \bigcup_{a \in \Sigma} (\{1,2\} \times \{4\}) \cdot a \cdot a \tag{5.24}$$

$$\beta_{count} = \{1,2\}\langle 1 \rangle \tag{5.25}$$

$$A_{filtered} = Bestpath\left(P_2\left(A \circ \left(\beta_4 \cup \beta_{count} \cup \overline{\Sigma}\right)^*\right)\right) \tag{5.26}$$

Generating all minimal factorizations

The current one-path acceptor already represents all minimal factorizations but it is using the pattern type 4. Thus, the current factorization contains sequences of this pattern as a factor of its own: In our running example, the middle a would be such a factor of its own, leading to a factorization $(l, s)(a, a)(ys, id)$.

As factors of type 4 are only "subsets" of type-1 and type-2 factors, the straightforward solution might be in just replacing all 4's with 1 and 2. However, this would lead to an overgeneration. To clarify this, let us look at the fictional example of *laays* : *laaid* :: *saays* : *saaid*. In encoded form, this analogy looks like:

```
2ls 4aa 4aa 1yi 1sd
```

Converting all 4's to 1 and 2 would generate the following factorizations:

1) `2ls 1aa 1aa 1yi 1sd`
2) `2ls 1aa 2aa 1yi 1sd`
3) `2ls 2aa 1aa 1yi 1sd`
4) `2ls 2aa 2aa 1yi 1sd`

While the factorizations 1), 3) and 4) do a pattern-type switch only once, the factorization 2) switches three times and counts every *a* as a factor of its own. Hence, 2) has more pattern type switches as other factorizations and is not minimal anymore.

Therefore, we need a way to convert back the 4's without introducing non-minimal factorizations. We have to deal with three cases:

		Current sequence		Desired sequence
1.	4 enclosed by 1 / by 2	`...1441...`	→	`...1111...`
		`...2442...`	→	`...2222...`
2.	4 between 1 and 2 / between 2 and 1	`...1442...`	→	`...1112...`
				`...1122...`
				`...1222...`
		`...2441...`	→	`...2111...`
				`...2211...`
				`...2221...`
3.	4 at the beginning / at the end	`41...`	→	`11...`
		`42...`	→	`22...`
		`...14`	→	`...11`
		`...24`	→	`...22`

These conversions are done with the following transducers applied sequentially to $A_{filtered}$:

$$\gamma_1 = \{1\} \cdot \overline{\Sigma} \cdot \overline{\Sigma} \tag{5.27}$$

$$\gamma_2 = \{2\} \cdot \overline{\Sigma} \cdot \overline{\Sigma} \tag{5.28}$$

$$\gamma_{4\times1} = (\{4\} \times \{1\}) \cdot \overline{\Sigma} \cdot \overline{\Sigma} \tag{5.29}$$

$$\gamma_{4\times2} = (\{4\} \times \{2\}) \cdot \overline{\Sigma} \cdot \overline{\Sigma} \tag{5.30}$$

$$\gamma_{enclosed1} = (\gamma_1 \cdot (\gamma_{4\times1}^* \cdot \gamma_1)^* \cup \{1,2,4\} \cup \overline{\Sigma})^* \tag{5.31}$$

$$\gamma_{enclosed2} = (\gamma_2 \cdot (\gamma_{4\times2}^* \cdot \gamma_2)^* \cup \{1,2,4\} \cup \overline{\Sigma})^* \tag{5.32}$$

$$\gamma_{between12} = (\gamma_1 \cdot \gamma_{4\times1}^* \gamma_{4\times2}^* \cdot \gamma_2 \cup \{1,2,4\} \cup \overline{\Sigma})^* \tag{5.33}$$

$$\gamma_{between21} = (\gamma_2 \cdot \gamma_{4\times2}^* \gamma_{4\times1}^* \cdot \gamma_1 \cup \{1,2,4\} \cup \overline{\Sigma})^* \tag{5.34}$$

$$\gamma_{beginning} = (\gamma_{4\times1}^* \cdot \gamma_1 \cup \gamma_{4\times2}^* \cdot \gamma_2 \cup \{1,2,4\} \cup \overline{\Sigma})^* \tag{5.35}$$

$$\gamma_{end} = (\gamma_1 \cdot \gamma_{4\times1}^* \cup \gamma_2 \gamma_{4\times2}^* \cdot \cup \{1,2,4\} \cup \overline{\Sigma})^* \tag{5.36}$$

$$A_{factorizations} = P_2(A_{filtered} \circ \gamma_{enclosed1} \circ \gamma_{enclosed2} \circ \gamma_{between12} \circ \gamma_{between21} \circ \gamma_{beginning} \circ \gamma_{end}) \tag{5.37}$$

With these auxilliary transducers, we have created a straightforward and easy implementable way for bypassing the often missing $Bestpaths()$ function.

And here we are done. $A_{factorizations}$ now contains all minimal factorizations of the given analogy – and nothing more.

Summary

The described finite-state machine is summarized in the following. As already stated, the input words represented as acceptors X, Y, Z, T and the final acceptor containing all

factorizations (or being empty in case of no analogy) is $A_{factorizations}$.

$$X_{prepared} = P_2\left(X \circ (\Sigma \cup \alpha_-)^* \circ \left((\alpha_1 \cup \alpha_2) \cdot \overline{\Sigma} \cdot \alpha_{\overline{\Sigma}}\right)^*\right) \tag{5.38}$$

$$Y_{prepared} = P_2\left(Y \circ (\Sigma \cup \alpha_-)^* \circ \left(\left(\alpha_1 \cdot \alpha_{\overline{\Sigma}} \cdot \overline{\Sigma}\right) \cup \left(\alpha_2 \cdot \overline{\Sigma} \cdot \alpha_{\overline{\Sigma}}\right)\right)^*\right) \tag{5.39}$$

$$Z_{prepared} = P_2\left(Z \circ (\Sigma \cup \alpha_-)^* \circ \left(\left(\alpha_1 \cdot \overline{\Sigma} \cdot \alpha_{\overline{\Sigma}}\right) \cup \left(\alpha_2 \cdot \alpha_{\overline{\Sigma}} \cdot \overline{\Sigma}\right)\right)^*\right) \tag{5.40}$$

$$T_{prepared} = P_2\left(T \circ (\Sigma \cup \alpha_-)^* \circ \left((\alpha_1 \cup \alpha_2) \cdot \alpha_{\overline{\Sigma}} \cdot \overline{\Sigma}\right)^*\right) \tag{5.41}$$

$$A_{factorizations} = P_2(Bestpath((X_{prepared} \cap Y_{prepared} \cap Z_{prepared} \cap T_{prepared}) \circ \tag{5.42}$$

$$\left(\beta_4 \cup \{1,2\} \cup \overline{\Sigma}\right)^* \circ \left(\beta_{count} \cup \{4\} \cup \overline{\Sigma}\right)^*) \circ \tag{5.43}$$

$$\gamma_{enclosed1} \circ \gamma_{enclosed2} \circ \gamma_{between12} \circ \gamma_{between21} \circ \gamma_{beginning} \circ \gamma_{end}) \tag{5.44}$$

We may note that $A_{factorizations}$ does not contain any brackets or morpheme borders yet. However, these can be obtained easily be just reading sequentially the encoded analogies and inserting a split after every change of 1 to 2 and back. We leave that to the following step producing the actual segmentations of the words.

5.4. Segmenting words using analogical factorizations

Given the exhaustive analogical factorizations of the previously described FSM, the next step generates segmentations for the words involved in the analogies, as depicted in Figure 5.11.

The work is done in two modes: For every analogy found

Input		**Output**
$[(lay, say)(\epsilon, s);$	$\rightarrow$	lay = lay
$(la, sa)(y, ys);$		lays = lay + s
$(l, s)(ay, ays)]$		say = say
$[(lay, hit)(\epsilon, s)]$		says = say + s
$[(say, hit)(\epsilon, s)]$		hit = hit
		hits = hit + s

Figure 5.11.: Input and output of the segmenter.

1. all factorizations are picked. This represents the new approach presented here

2. only one factorization is picked randomly. This is the approach of (Lavallée and Langlais, 2009a), and is also done by this segmenter in order to compare the two approaches.

The following part of the algorithm is the same as Ana-Seg in (Lavallée and Langlais, 2009a): For all picked factorizations of an analogy, the words involved are segmented according to their picked factorizations, and the four segmented words of an analogy become candidates for the final segmentations.

For every word, a global list of all its segmentation candidates is collected. Finally, after having processed all analogies, for every word the most frequent segmentation is taken as its final segmentation.[1]

For instance, given the two factorizations $[(la, sa)(ys, id); (l, s)(ays, aid)]$, every word involved gets two segmentation candidates:

$$lays = (la + ys, l + ays) \tag{5.45}$$

$$says = (sa + ys, s + ays) \tag{5.46}$$

$$laid = (la + id, l + aid) \tag{5.47}$$

$$said = (sa + id, s + aid) \tag{5.48}$$

Now *lays* has globally two candidates with the frequency 1: $la + ys$ and $l + ays$. Other factorizations may increase the frequency of one of theses candidate or add new candidates. As stated, in the end the most frequent candidate will be the final segmentation of *lays*.

5.5. Summary

The presented segmenter is composed by three modules: a filter, a factorizer and a segmenter. The first and the latter steps were written as a Python script[2], the FSM factorizer was implemented in the FSM2 toolkit (Hanneforth, 2009).

[1] If there is more than one most frequent segmentation, one is picked randomly as done by (Lavallée and Langlais, 2009a), stated by Philippe Langlais in a personal e-mail conversation.

[2] http://www.python.org/

While the segmenter is a simple adaption of the algorithm of (Lavallée and Langlais, 2009a), the first two modules contain important differences compared to previous approaches.

5.5.1. Filter

To cope with the huge search space when searching for analogies in a non-toy word list, Lavallée and Langlais (2009a) use an algorithm presented in (Langlais and Yvon, 2008). As this algorithm is looking for triples given one input word, the words in the input list are processed sequentially. For every word, a time limit is set, thus not for all words analogies are found.

This is different in the filter presented here: It processes the whole word list at once and gives an exhaustive list of all possible analogies. As in (Lavallée and Langlais, 2009a), the filter makes use of the character count property (called there T-Trick) and the boundary character property (called there S-Trick).

However, the computational complexity of the current approach is still not low, as it is quadratic in relation to the input list size. Therefore, the input need to be limited, as will be pointed out in the Evaluation chapter 6.

5.5.2. Factorizer

To my knowledge, there did not exist a description of an analogical factorizer yet in the literature. Also (Lavallée and Langlais, 2009a) misses a hint or reference to the factorization technique they applied.

The factorization algorithm applied here is based on the analogical solver presented in (Yvon et al., 2004, page 12 ff.). Again, this algorithm was not a direct solution for the given task and had to be adapted. Its actual purpose is solving analogical equations, as already explained in Section 3.1.3.

The basic construction of the current factorizer is similar to the existing solver: Both are based on finite-state machines and try to create a n-way alignment of the input words by preparation and intersection. However, in (Yvon et al., 2004), this alignment is not yet restricted, allowing priorly all possible alignments, which may lead to a huge

automaton. Only the next step, called the "atomic solver", applies analogical pattern to the aligned tuples and filters out non-analogical alignments.

In contrast, the factorizer presented here restricts the alignments to analogical ones. Thus, after intersection, no further atomic solver is required to check for an analogy. Additionally, the intersected automaton is probably much smaller, as it contains only analogical alignments.

Moreover, it does not output only one word (as the analogical solver), but all words of the processed analogy in a new way of encoding analogies. This by-product, a space-saving formalism for encoding analogies, might be also useful in other analogical applications, as explained in Conclusion chapter 7.

6. Evaluation

6.1. Used datasets

6.1.1. Sources

The segmenter proposed in the previous chapter was then implemented and evaluated, focussing on the aspect of finding all factorizations of an analogy. For choosing the dataset, the following restrictions applied:

- The evaluation should be comparable to the one in (Lavallée and Langlais, 2009a), as it was this approach that has been developed further here.
- Since the filter and factorizer have a quadratic complexity regarding to the size of the input word list and the computational resources and time were limited, the input data had to be limited. Thus, a reasonable way for choosing an excerpt of the data had to be found.

(Lavallée and Langlais, 2009a) developed their method for Morpho Challenge 2009. Its "Competition 1" offers a large dataset of words in 5 different languages – English, German, Finnish, Turkish and Arabic, together with their frequencies, still downloadable online[1]. However, the Gold standard available online is very small, as it contains only 400-700 words per language.

Therefore, to obtain a much more significant evaluation, Lavallée and Langlais (2009a) used also data from *Celex* (Baayen et al., 1993), offering around 70,000 morphologically annotated word forms in English and more than 300,000 in German[2]. As Lavallée and Langlais (2009a, page 7) did already an extraction of morpheme analyses from the Celex

[1] http://research.ics.tkk.fi/events/morphochallenge2009/datasets.shtml

[2] Celex contains also Data in Dutch, which however was omitted as Dutch was not part of the Morpho Challenge 2009 competition.

dataset:	Celex	Celex additional	Morpho Challenge (MC)
segmenter	top 1000	freq. ranks 5000-10000	top 10000 +
input words:	top 2000	freq. ranks 10000-15000	words from gold standard
	top 5000	freq. ranks 15000-20000	(400-700 per language)
	top 10000		≈ 10400 to 10700 words
evaluation words:	all input	all input	only gold standard
languages:	eng, ger	eng, ger	eng, ger, fin, tur, vowara

Figure 6.1.: Datasets used for evaluation. Thus, "Celex" contained 4 different word lists for each of its languages, "Celex additional" 3 and "MC" 1.

data, I asked and was given access to their dataset. This dataset however did not contain word frequencies.

So by using the same data sources, the evaluation could be comparable. However, due to the complexity and calculation time, the data still had to be limited, as will be described in the following section.

6.1.2. Compilation

The most straightforward way for limiting the data was ranking the words by frequency and then picking the topmost n words. By this, we probably would also come close to a "real world" situation where for the analysis of a not well explored language, only some data would be available and we may assume that such data usually would consist of high-frequent words of this language.

Since the Morpho Challenge training data (MC) contained frequencies, ranking the words was easy there. This was not the case for Celex, so in order to rank its words, the Celex word list was intersected with the MC list and then ranked according to the frequencies from MC.

Finally, the following word lists were compiled and used, an overview is also depicted in Figure 6.1.

- *Celex* with the languages English (eng) and German (ger), using the most-frequent 1000, 2000, 5000 and 10000 words.[3]

[3]The limit of 10000 was chosen since a run with that many words takes around 11 hours for one

- *MC* with the languages English (eng), German (ger), Finnish (fin), Turkish (tur) and Vowelized Arabic (vowara) using the most-frequent 10000 words. As mentioned, MC data was not labeled, only an additional gold standard with 400-700 contained labels. Most of the words in the gold standard were not part of the most frequent words, so in order to be able to measure the performance of the segmenter, the gold standard words were added to the MC data.[4]
- In order to measure any bias caused by the selection by frequency, a third *Celex additional* dataset was created, containing chunks of each 5000 words of lower ranks, e.g., words from frequency rank 5000 to 10000, from rank 10000 to 15000, and so on. The limit 5000 was picked as it seemed to be a good trade-off between computation time and segmenter performance.

6.1.3. Processing

In every dataset, all characters that were not Latin letters, numbers or part of $\{.-'\}$, were replaced by a $\sim$. This affected mainly Finnish, as the original data contained Germanic umlauts which were now converted to hyphens. For German this was not an issue as the data both from Celex and MC contained already transcribed umlauts.

While this might have corrupted some results in Finnish, note that the effect can be only on precision not recall since the applied preprocessing mapped characters from a larger set to a smaller. Thus, for instance the following analogy can be still found when replacing the *ä* with $\sim$:

$$pyydettiin : pyydetty :: nähtiin : nähty \tag{6.1}$$

$$pyydettiin : pyydetty :: n \sim htiin : n \sim hty \tag{6.2}$$

language on the computer I had access to – which I considered a reasonable time for applying and possible refining the system. Remeber that this time grows quadrically in relation to the input size.

[4]It may be confusing to use gold standard data for creating a system and evaluating it at the same time. However, note that only the *words* from the gold standard were used, not their morphological analyzes. Remember also, because of its structure, the system can be run only once on a "training=test" dataset already producing analyzes, not a model. A later query with an unseen word is not possible. Therefore, we do not have the typical distinction between training and test data like in other unsupervised systems. These usually first create a model using training data which then can be applied on (unseen) test data.

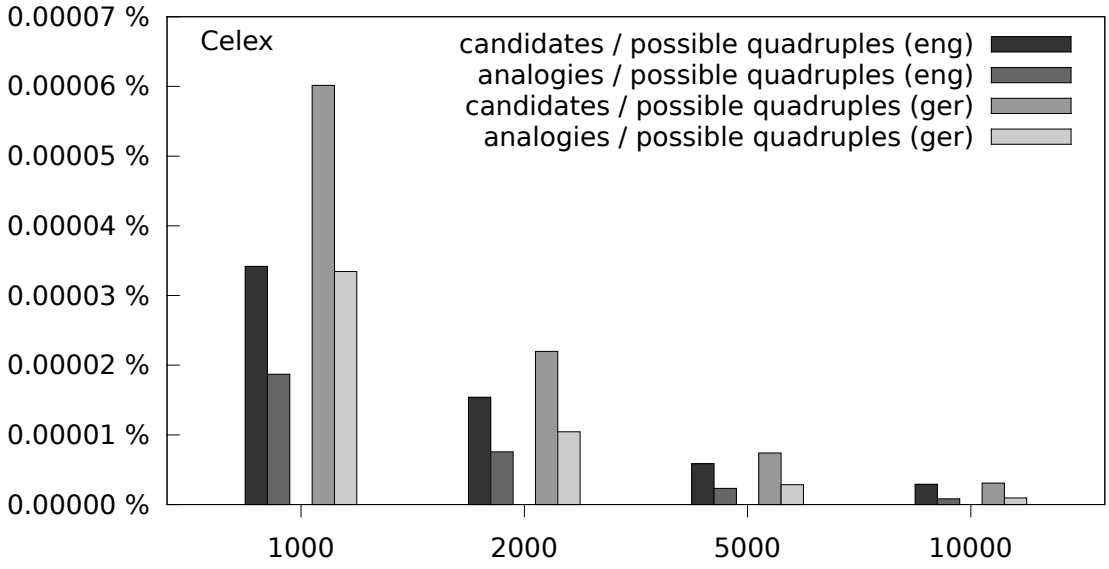

Figure 6.2.: Rates of analogy candidates and analogies in relation to possible quadruples. Based on Celex dataset.

6.2. Experiments on analogies

6.2.1. Filter performance

The raw numbers of all experiments are presented in the Appendix chapter A.

The first step of the segmenter pipeline is the filter which generates candidate quadruples that might be an analogy. Its results, depicted in Figure 6.2, show the rate of generated candidate quadruples and actual analogies in relation to possible quadruples. Two things can be seen:

- The filter saves us *a lot* of analogy checks. From all possible quadruples, i.e., $\binom{n}{4}$, on average only $0,00001\%$ need to be checked with the FSM factorizer.
- Roughly half to one fourth of analogy candidates turn out to be analogies, declining with growing input size.
- The rate of candidates and analogies in relation to possible quadruples is falling rapidly. As the number of analogies is growing with more input, the pure number of analogies is still growing with more input but not as fast as the number of possible quadruples.

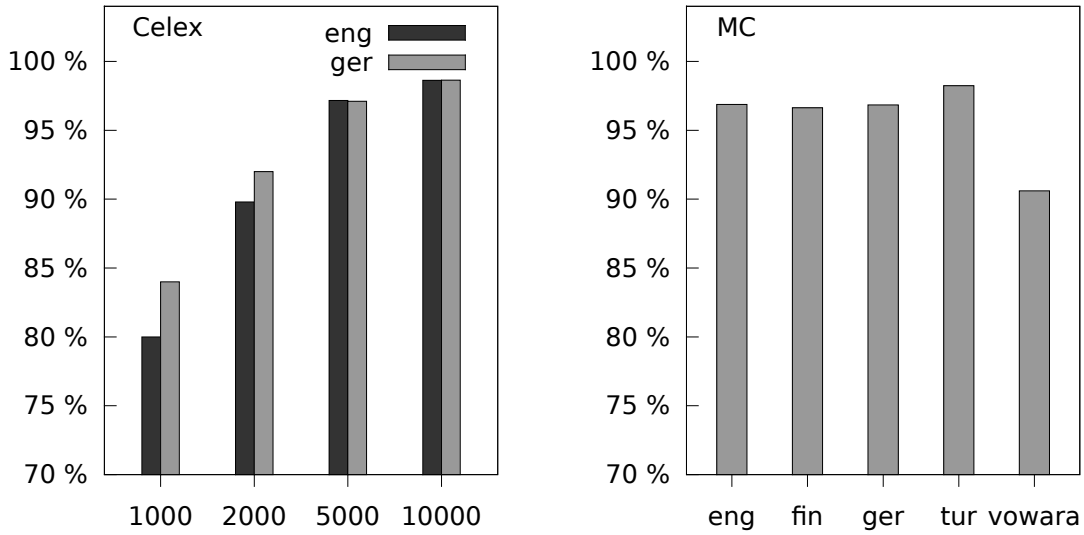

Figure 6.3.: Rate of words covered by at least one analogy, in Celex dataset (left) and MC (right).

6.2.2. Analogy coverage

In following, we examine how many of the input words were found as parts of an analogy, i.e., were "covered by an analogy", since only these words had a chance to be analyzed by the proposed segmenter. The results are portrayed in Figure 6.3.

We note that with more input, the rate of covered words is growing, reaching a very high level of more than 95%. This is interesting, since (Lavallée and Langlais, 2009a, page 6) observed "many words that these two systems do not treat adequately" because many analogies were not found.

These on first sight contradicting findings could be explained, if the coverage would be that high only among high frequent words and would drop again when taking more non-frequent words into account. An experiment carried out on the *Celex additional* dataset using only lower frequent words seems to confirm this, as seen in Figure 6.4. However, we still don't know how the coverage looks for high- *and* low-frequent words together. Our current assumption is probably misleading: there is a drop in coverage between 1-5000 and 5000-10000 (6.4) but a raise between top-5000 and top-10000 (Figure 6.3).

So maybe the assumption of analogical data-sparsity (i.e., that within lists of low-frequent words, the words are too "special" or rare to find other words which could

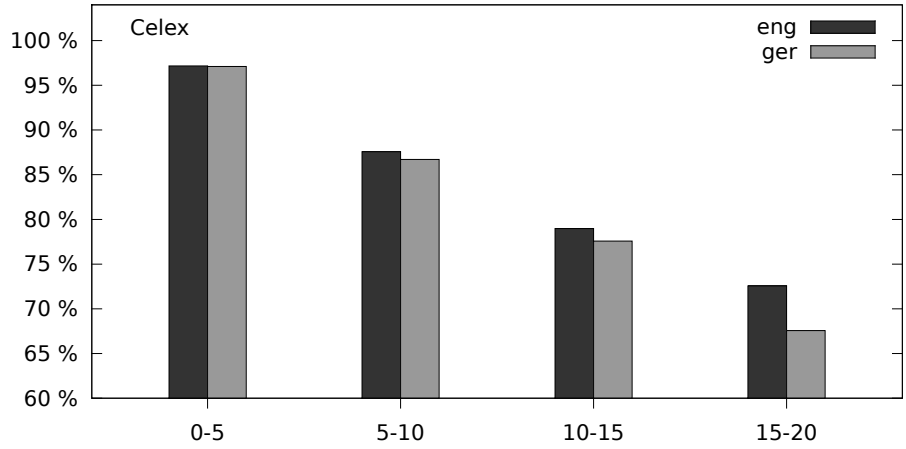

Figure 6.4.: Rate of words covered by at least one analogy, The x values denote word frequency ranks in thousands.

form an analogy with them) is not true: Maybe we can cover nearly all words with more input. We just need an exhaustive analogy finder (and some patience when using only contemporary computers).

Looking at the MC data in Figure 6.3, we can see a similar high coverage in other languages, only Arabic falls off a bit, possibly due to its very complex infix inflection, causing sparsity.

6.2.3. Different factorizations

In the next section, we shall check the segmenter performance. We will compare considering all factorizations of an analogy vs. considering only one. Before we do this, it would be good to know how many analogies actually have more than one factorization. Only on these analogies the new approach can have an impact. The numbers are depicted in Figure 6.5.

Firstly, there is no noteworthy change by scaling the input. But secondly, there are big differences among the languages. While in English only slightly more than 10% of the analogies have more than one factorization, the rate in German is nearly 30%.

A reason for this could be tied character sequences (like “sch”), that occur probably more often in German than in English. If then all four words of an analogy contain the same

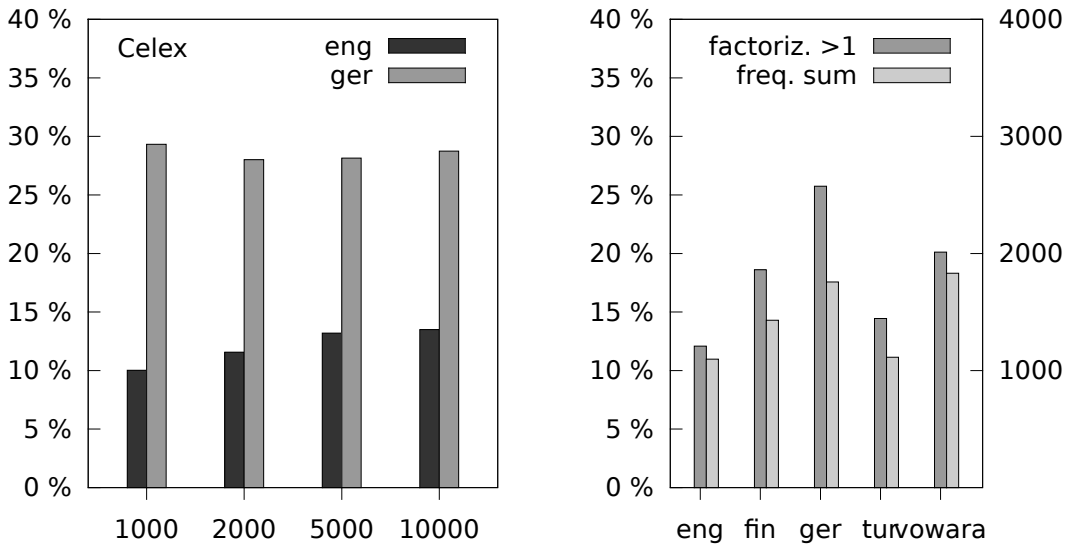

Figure 6.5.: Rate of Analogies with more than one factorization among all analogies, in Celex dataset (left) and MC (right). The right diagramm for MC also contains summed frequencies for the top-10 character bigrams in each language.

sequence, multiple factorizations are generated. For instance, the analogy *politischen : politisch :: deutschen : deutsch* has four minimal factorizations:

$$(politi, deut)(schen, sch)$$
$$(politis, deuts)(chen, ch)$$
$$(politisc, deutsc)(hen, h)$$
$$(politisch, deutschen)(en, \epsilon)$$

all caused by the common tied character sequence "sch" which is frequent in German adjective endings.

To confirm this assumption, we take a look on character bigrams. To measure the proportion of common bigrams in a language, the sum of the frequencies of the top-10 character bigrams for every MC language in the used dataset was calculated. The higher this number, the more common tied char sequences and thus more analogies with more than one factorization are expected.

For every MC language, these sums were plotted alongside the rate of analogies with

more than one factorization. As we see in Figure 6.5, both numbers seem to correlate more or less.

So if the new system with considering all factorization has any effect, it is expected to be rather in German or Arabic than in English. However, in every language the vast majority of analogies – more than $\frac{2}{3}$ – contains only one factorization, thus the new system will have no effect there.

6.3. Experiments on segmentation

6.3.1. Scaling and Languages

In the next experiments, we compare the performance of the actual morpheme segmenter, the main task of this thesis. As a start, we checked how the performance differs between

- considering all factorizations (my approach),
- considering one factorization (Ana-Seg by (Lavallée and Langlais, 2009a)) and
- Morfessor (as a general benchmark).

To begin with, I used the Celex dataset with English and German to see how the numbers change with more input. Both diagrams are depicted in Figure 6.6.

First of all, the results of Ana-Seg (one factorization) are coherent with the ones reported in (Lavallée and Langlais, 2009a, page 8), thus the present implementation can be considered as faithful.

The key result is quite disappointing: There is barely a noticeable difference between considering all factorizations (i.e., the main improvement of the proposed segmenter) and only one factorization, as already done by (Lavallée and Langlais, 2009a). In nearly all measured categories, the performance is only slightly better, not more than one percentage point.

We see the same (non-)effect on the other languages, as depicted in the last row of Figure 6.6: No big difference between considering all and one factorization.

The reason for this is probably in general the low rate of analogies with more than one factorization – as noticed in the previous section, it is only 20% on average. But it is

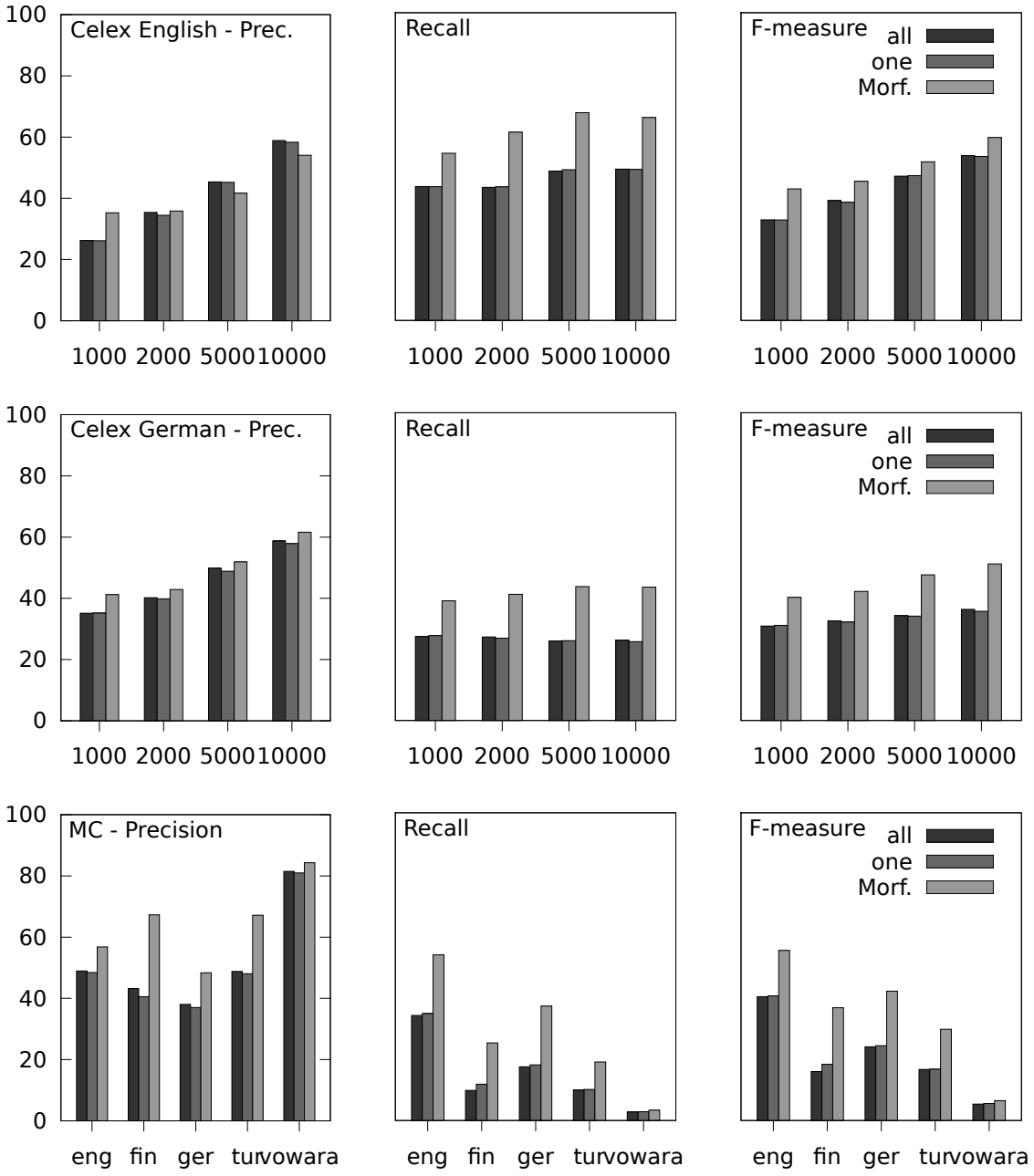

Figure 6.6.: Performance of the segmenter on Celex English and German (first two rows) and MC (last row). Every x-point shows three bars: Analogical segmenter using all minimal factorizations (approach presented in this thesis), one factorization (Lavallée and Langlais, 2009a), Morfessor

still surprising, that even in German having nearly 30% analogies with more than one factorization, no impact can be seen.

Compared to Morfessor, the two analogical systems underachieve in nearly all measured categories, but especially in recall. It is also interesting that more even data does not improve it – in contrast to precision. This might be due to the effect that an analogy usually captures only one morphological change at a time and a generalization is missing, causing a problem for multi-morphemic words. For instance, the word *underachieves* may appear in the two analogies, generating these segmentation candidates:

$$underachieves : underachieve :: says : say \rightarrow underachieve + s \tag{6.3}$$

$$underachieves : achieves :: undertake : take \rightarrow under + achieves \tag{6.4}$$

As we see, both morphemes are captured – but by different analogies, and the generated segmentation candidates are not merged together. Thus, probably only one candidate will win – instead of the correct merged candidate $under + achieve + s$.

This is a main drawback of the analogical approach that (Lavallée and Langlais, 2009a) tried to fix with c-rules, implemented in the more sophisticated Ana-Cof system. Another way of coping with this drawback might be using informational measures like MDL, as already indicated. We will come back to this in Conclusion chapter 7.

6.3.2. Frequent and less frequent words

Finally, as the data had to be limited for computational reasons, I wanted to check if the choice of high frequent words makes a difference. Therefore, in comparison to the already calculated top-5000-words chunk of Celex, the following 5000-words chunks were given to the segmenter. If the results would remain similar, one could maybe divide all the data into such chunks and receive results for *all* of the data, avoiding the problem of computational complexity. Unfortunately this is not the case, as depicted in Figure 6.7.

As we can see, the numbers are falling slightly in all categories. This is a bit surprising on first sight, as one could assume that less frequent words are more morphologically regular, thus more (morphological) analogies should be found. However, most certainly the reason for this is the lower analogy coverage, as shown in subsection 6.2.2. As

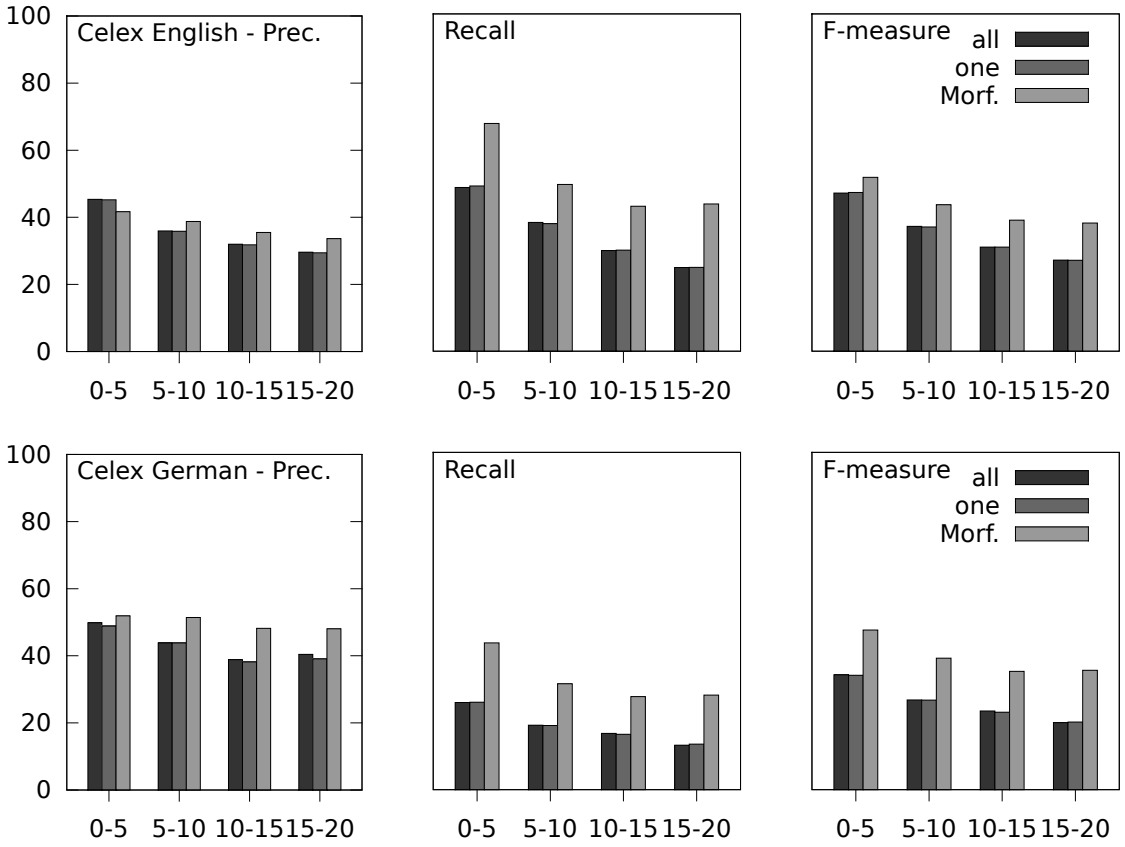

Figure 6.7.: Performance of the segmenter on Celex English and German using words of different frequency ranks. The x values denote word frequency ranks in thousands. Every x-point shows three bars: Analogical segmenter using all minimal factorizations (approach presented in this thesis), one factorization (Lavallée and Langlais, 2009a), Morfessor

explained there, the reason for this lower coverage is probably not data-sparsity but the missing high-frequency words within the lower-frequency parts. We can expect better results when analyzing high- *and* low-frequency words together – when better machines or algorithms for calculating analogies within them are available.

6.4. Summary

First of all, the filter is apparently a good predictor: at least a quarter of its generated candidates turn out to be analogies. Comparing to the potential number of analogies to be checked of $\binom{n}{4}$, the number of filterer candidates are only a small fraction.

Secondly, different factorizations do not play an important role on analogies. More than one factorization occurs among different languages at average only on $\frac{1}{4}$ of their analogies. Thus, the possible impact of considering more than one factor is already limited.

Finally, the actual impact turns out to be nearly non-existing. Among all experiments carried out – on different input sizes, languages and high or less frequent words – no noticeable difference can be seen between considering all factorizations and considering only one at random using the current method of picking a final segmentation candidate. This could be different when applying the exhaustive factorization system to Ana-Cof or to a novel system combining formal analogies with measures from information theory like MDL.

7. Conclusion

7.1. Summary of this thesis

In the present diploma thesis, I have given an outline on the problem of morpheme segmentation and of formal analogies. I have then described two existing well-known approaches for the problem and an already existing system using formal analogies.

The latter one, I have tried to develop further, with a focus on multiple minimal factorizations – a problem that was apparently not recognized and considered until now. I have described the conditions for occurring such factorizations and developed a factorizer based on finite-state machines returning an exhaustive set of multiple minimal factorizations. This factorizer was also presented in detail.

Furthermore, I have created and presented a filter for analogical quadruples, reducing the computational complexity for finding analogies from $\binom{n}{4}$ to quadratic.

The developed factorizer was then implemented and evaluated with experiments carried out on different input sizes, languages, and high vs. less frequent words. It turned out that considering all minimal factorizations did not lead to an improvement using the present method of calculating final segmentation candidates.

Nevertheless, considering all minimal factorizations the analogical system is cleaner than picking one at random. While this may have no impact on the actual segmenter performance with the current further steps of creating segmentation candidates, this may be different using another approach, as for instance Ana-Cof from (Lavallée and Langlais, 2009a) or MDL.

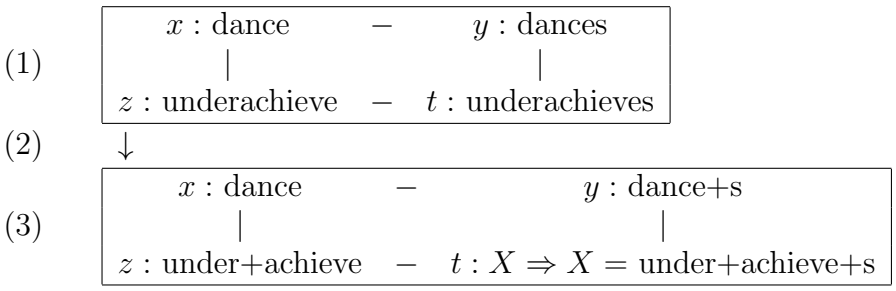

Figure 7.1.: Segmenting the word *underachieves* using a list of segmented words that only contains *dance, dances, underachieve* and their segmentations. In step (1), we search for appropriate triples and find *(dance, dances, underachieve)*. In (2), we map the items from the input to output domain, i.e., we simply look up their segmentations. Eventually, we generate our solution, i.e. our translation in step (3). This Figure is a simple adaption of Figure 3.3.

7.2. Future work

7.2.1. Improving the Filter

The main current limitation of approaches related to formal analogies is the relative rather small data size they can handle, caused by the computational complexity. As we have seen in subsection 6.3.1, the recall improves with more data. Thus, improving the filter and making it able to handle more input data would very likely also improve recall and so the performance of the segmenter.

7.2.2. Supervised approach using Analogical learning

Another idea of using formal analogies for morpheme segmenation could be realised within the framework of analogical learning, as presented in section 3.4. However, this approach would not be unsupervised anymore, as it would need segmented data to learn from. As input domain, we still could use a word list, the output domain would be the same words in segmented form. See Figure 7.1 for an outline of the approach.

The realisation of such an approach would also be much easier if there would already exist an implemented analogical learning framework. Having such an implementation one could examine various systems (linguistic) “mapping” problems, be it mapping words

to their segmentations, words to their Part-of-Speech tags but also sentences to their syntax tree.

7.2.3. Formal Analogies with MDL

As already indicated, it might be very promising to investigate an approach combining formal analogies with measures from information theory like MDL. Recalling Goldsmith from section 4.3,

> "[...] the purpose of linguistic theory is to serve as a set of heuristics to help the linguistic scientist come up with a tight, snug grammar, given a set of data. MDL can determine which of a set of grammars is the best one, given the data; no feasible process can search all possible grammars, so there is no guarantee that another linguist will not come along tomorrow with a better grammar for the data." (Goldsmith, 2007, page 8)

The currently existing approaches seem to be restricted to one part of the combination: Morfessor and Linguistica use MDL for evaluating, but only heuristics for generating the candidates – Ana-Seg and Ana-Cof use Formal Analogies, but no MDL for evaluation.

By combining formal analogies with MDL, we would have on the one hand a clean, non-heuristic theoretical model for generating morph candidates and a morphology grammar, and on the other hand a well-explored model of examining the given candidates and grammar. With the recognition and extraction of multiple minimal factorizations and also the exhaustive filtering of analogical candidates presented in this thesis, we now have the basics for a good exploitation of formal analogies.

A. Evaluation Results in Detail

A.1. Experiments on analogies

A.1.1. Filter perfomance

input size	possible quadruples	eng candidates	eng analogies	ger candidates	ger analogies
1000	41417124750	14153	7747	24913	13849
2000	664668499500	102332	50344	146080	69369
5000	26010428123750	1526320	607169	1925181	738127
10000	416416712497500	12175882	3437902	12825949	3940156

A.1.2. Analogy coverage

input size	Celex eng covered words	Celex ger covered words	MC Language	input size	covered words
1000	800	840	eng	10466	10139
2000	1796	1840	fin	10634	10276
5000	4858	4855	ger	10525	10192
10000	9863	9864	tur	10581	10394
			vowara	10690	9685

A.1.3. Different factorizations

input size	Celex eng analogies w. > 1 fctrz.	Celex ger analogies w. > 1 fctrz	MC Language	analogies	analogies with > 1 factorizations
1000	777	4061	eng	2967623	358658
2000	5827	19428	fin	2041579	379905
5000	80140	207717	ger	3328257	856805
10000	464164	1132453	tur	2483118	358861
			vowara	1166654	234669

A.2. Experiments on segmentation

A.2.1. Scaling and Languages

Celex eng

	all			one			Morfessor		
input	Pr	Rc	F	Pr	Rc	F	Pr	Rc	F
1000	26.18	43.53	32.70	26.15	43.47	32.66	35.26	54.33	42.76
2000	35.39	43.23	38.92	34.44	43.43	38.42	35.84	61.18	45.20
5000	45.35	48.56	46.90	45.23	48.99	47.04	41.67	67.52	51.54
10000	58.87	49.15	53.57	58.31	49.09	53.30	54.08	65.96	59.43

Celex ger

	all			one			Morfessor		
input	Pr	Rc	F	Pr	Rc	F	Pr	Rc	F
1000	35.14	27.22	30.68	35.16	27.59	30.92	41.22	38.91	40.03
2000	40.21	27.14	32.41	39.85	26.72	31.99	42.88	41.02	41.93
5000	49.86	25.84	34.04	48.85	25.95	33.90	51.92	43.50	47.34
10000	58.73	26.06	36.11	57.87	25.60	35.49	61.58	43.36	50.89

MC

	all			one			Morfessor		
input	Pr	Rc	F	Pr	Rc	F	Pr	Rc	F
eng	48.95	34.16	40.24	48.46	34.78	40.49	56.79	53.83	55.27
fin	43.21	9.77	15.94	40.55	11.79	18.27	67.28	25.21	36.67
ger	38.04	17.46	23.94	37.01	18.10	24.31	48.37	37.19	42.05
tur	48.82	9.98	16.58	48.07	10.08	16.67	67.14	19.04	29.66
vowara	81.48	2.76	5.33	80.99	2.87	5.53	84.32	3.36	6.46

A.2.2. Frequent and less frequent words

Celex additional eng

	all			one			Morfessor		
input	Pr	Rc	F	Pr	Rc	F	Pr	Rc	F
1-5000	45.35	48.56	46.90	45.23	48.99	47.04	41.67	67.52	51.54
5001-10000	35.93	38.19	37.03	35.85	37.84	36.82	38.74	49.45	43.44
10001-15000	32.00	29.84	30.88	31.82	29.97	30.86	35.47	42.96	38.86
15001-20000	29.59	24.83	27.00	29.44	24.85	26.95	33.63	43.66	38.00

Celex additional ger

	all			one			Morfessor		
input	Pr	Rc	F	Pr	Rc	F	Pr	Rc	F
1-5000	49.86	25.84	34.04	48.85	25.95	33.90	51.92	43.50	47.34
5001-10000	43.81	19.09	26.59	43.83	19.04	26.55	51.41	31.40	38.99
10001-15000	38.84	16.70	23.35	38.18	16.43	22.97	48.19	27.60	35.10
15001-20000	40.41	13.17	19.87	39.09	13.49	20.06	48.04	28.03	35.40

B. Selbständigkeitserklärung

Hiermit versichere ich, Georg Jähnig, geboren am 20. März 1980 in Berlin, dass ich die vorliegende Arbeit selbständig verfasst habe und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt habe.

Berlin, den 31. Juli 2011

C. Zusammenfassung in deutscher Sprache

In der vorliegenden Arbeit habe ich mich mit *Unüberwachter Morphem-Segmentation* beschäftigt. Dieses Problemfeld der Computeringuistik widmet sich der Entwicklung von Systemen, die Wörter in ihre kleinsten bedeutungstragenden Einheiten (Morpheme) trennen, und dabei nur natürlichsprachliche, nicht-annotierte Daten zu nutzen.

Solche Systeme haben zum einen praktische Relevanz: Morphem-Segmentation ist bei Verarbeitung natürlicher Sprache oft die Voraussetzung für weitere Schritte wie z.B. lexikalische und syntaktische Analyse. Zum anderen haben sie auch theoretische Relevanz: Eine Morphologie, die automatisch und direkt aus natürlichsprachlichen Daten erstellt wurde, kann zur Inspiration und Weiterentwicklung manuell erstellter Morphologien dienen bzw. diese bestätigen oder widerlegen.

Arbeiten rund um unüberwachte Morphem-Segmentation erscheinen regelmäßig zur *Morpho Challenge*, einer jährlichen Konferenz an der Universität Aalto (Finnland). Eine der dort eingereichten Arbeit von Lavallée and Langlais (2009a) fußt maßgeblich auf dem Konzept *formaler Analogien*: Vier Wörter wie z.B. $fragen : fragst :: sagen : sagst$ bilden eine solche Analogie. Aus dieser werden *minimale Faktorisierungen* generiert, wie z.B. $(frag, sag)(en, st)$. Diese Faktoren werden dann als Morphem-Kandidaten genutzt. Die dort eingereichte Arbeit vernachlässigte jedoch das Problem *multipler minimaler Faktorisierungen*: Nicht nur die gegebene, sondern auch $(fra, sa)(gen, gst)$ und $(fr, s)(agen, agst)$ sind valide minimale Faktorisierungen. In dieser Arbeit stelle ich ein System vor, das all diese Faktorisierungen in Betracht zieht.

Dazu habe ich in Kapitel 2 zunächst das Problem dieser Arbeit disktutiert und abgegrenzt, dabei verschiedene Quellen aus der Morphologie herangezogen. In Kapitel 3 habe ich in die theoretischen Grundlagen formaler Analogien eingeführt. Kapitel 4 widmete ich einer genaueren Beschreibung von (Lavallée and Langlais, 2009a) sowie den

zwei weiteren, bekannten Systemen *Morfessor* (Creutz and Lagus, 2005) und *Linguistica* (Goldsmith, 2006). In Kapitel 5 habe ich mein System vorgestellt und dieses in Kapitel 6 evaluiert.

Bei der Evaluation stellte sich heraus, dass die eingeführte Veränderung zu keiner sichtbaren Verbesserung der Ergebnisse bei der Morphem-Segmentierung führt (siehe Abschnitt 6.3.1). Immerhin konnten bei der Evaluation aber auch allgemeine Erkenntnisse zum Vorkommen formaler Analogien in natürlichsprachlichen Daten gewonnen werden. So ist die Anzahl von Wörtern, die eine Analogie bilden können, mit durchschnittlich 95% unerwartet hoch (siehe Abschnitt 6.2.2). Die für das System entwicklte Kodierung formaler Analogien 5.3.2 könnte zudem nützlich für andere Anwendungen sein.

Bibliography

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. OpenFst: A general and efficient weighted finite-state transducer library. In *Proceedings of the Ninth International Conference on Implementation and Application of Automata, (CIAA 2007)*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23. Springer, 2007. `http://www.openfst.org`.

R. H. Baayen, R. Piepenbrock, and van H. Rijn. *The CELEX lexical data base on CD-ROM*. Linguistic Data Consortium, Philadelphia, PA, 1993.

Hadumod Bussmann, Gregory Trauth, and Kerstin Kazzazi. *Routledge dictionary of language and linguistics / Hadumod Bussmann ; translated and edited by Gregory Trauth and Kerstin Kazzazi*. Routledge, London ; New York :, 1996. ISBN 0415022258 0415203198 0415203198 0415022258.

Stanley F. Chen. *Building Probabilistic Models For Natural Language*. PhD thesis, Harvard University, 1996.

Mathias Creutz. Unsupervised segmentation of words using prior distributions of morph length and frequency. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 280–287, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1075096.1075132. URL `http://dx.doi.org/10.3115/1075096.1075132`.

Mathias Creutz and Krista Lagus. Unsupervised discovery of morphemes. *Computing Research Repository*, 2002.

Mathias Creutz and Krista Lagus. Unsupervised morpheme segmentation and morphology induction from text corpora using morfessor 1.0. In *Helsinki University of Technology*, 2005.

Ferdinand de Saussure. *Course in General Linguistics*. McGraw-Hill, New York, 1959. Translated by Wade Baskin.

John Goldsmith. Unsupervised learning of the morphology of a natural language. *Comput. Linguist.*, 27:153–198, June 2001. ISSN 0891-2017. doi: http://dx.doi.org/10.1162/089120101750300490. URL http://dx.doi.org/10.1162/089120101750300490.

John Goldsmith. An algorithm for the unsupervised learning of morphology. *Nat. Lang. Eng.*, 12:353–371, December 2006. ISSN 1351-3249. doi: 10.1017/S1351324905004055. URL http://portal.acm.org/citation.cfm?id=1181159.1181162.

John Goldsmith. Morphological analogy: Only a beginning. 2007.

Margaret A. Hafer and Stephen F. Weiss. Word segmentation by letter successor varieties. *Information Storage and Retrieval*, 10(11-12):371 – 385, 1974. ISSN 0020-0271. doi: DOI:10.1016/0020-0271(74)90044-8. URL http://www.sciencedirect.com/science/article/B6X2J-465CX4W-7W/2/d6a624819389ad477cffb68edbd0a72b.

Rogers P. Hall. Computational approaches to analogical reasoning : A comparative analysis. *Artificial Intelligence*, 39(1):39 – 120, 1989. ISSN 0004-3702. doi: DOI:10.1016/0004-3702(89)90003-9. URL http://www.sciencedirect.com/science/article/pii/0004370289900039.

Harald Hammarström. *Unsupervised Learning of Morphology and the Languages of the World*. PhD thesis, Chalmers University of Technology and University of Gothenburg, 2009.

Thomas Hanneforth. *fsm2* - A Scripting Language Interpreter for Manipulating Weighted Finite-state Automata. In *Anssi Yli-Jyrä et al. (eds): Finite-State Methods and Natural Language Processing, 8th International Workshop*, pages 13–30, 2009.

Zellig S. Harris. From Phoneme to Morpheme. *Language*, 31(2):190–222, 1955. ISSN 00978507. doi: 10.2307/411036. URL http://dx.doi.org/10.2307/411036.

Robert R. Hoffman. Monster analogies. *AI Magazine*, 16(3):11–35, 1995. URL http://dblp.uni-trier.de/db/journals/aim/aim16.html#Hoffman95.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*

(Prentice Hall Series in Artificial Intelligence). Prentice Hall, 1 edition, 2000. ISBN 0130950696.

Paul Kroeger. *Analyzing Grammar: An Introduction*. Cambridge University Press, Cambridge, 2005.

Mikko Kurimo, Sami Virpioja, Ville T. Turunen, Graeme W. Blackwood, and William Byrne. Overview and results of Morpho Challenge 2009. In *Working Notes for the CLEF 2009 Workshop*, September 2009. URL `http://eprints.pascal-network.org/archive/00006052/`.

Mikko Kurimo, Sami Virpioja, Ville Turunen, and Krista Lagus. Morpho challenge competition 2005–2010: evaluations and results. In *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*, SIGMORPHON '10, pages 87–95, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. ISBN 978-1-932432-76-3. URL `http://portal.acm.org/citation.cfm?id=1870478.1870489`.

Philippe Langlais and Alexandre Patry. Translating unknown words by analogical learning. In *EMNLP-CoNLL*, pages 877–886. ACL, 2007. URL `http://dblp.uni-trier.de/db/conf/emnlp/emnlp2007.html#LanglaisP07`.

Philippe Langlais and François Yvon. Scaling up analogical learning. 2008.

Jean-François Lavallée and Philippe Langlais. Morphological acquisition by formal analogy. In *Morpho Challenge 2009*, Corfu, Greece, 2009a.

Jean-François Lavallée and Philippe Langlais. Unsupervised morphological analysis by formal analogy. In Carol Peters, Giorgio Maria Di Nunzio, Mikko Kurimo, Djamel Mostefa, Anselmo Pe nas, and Giovanna Roda, editors, *CLEF (1)*, volume 6241 of *Lecture Notes in Computer Science*, pages 617–624. Springer, 2009b. ISBN 978-3-642-15753-0. URL `http://dblp.uni-trier.de/db/conf/clef/clef2009.html#LavalleeL09`.

Yves Lepage. Solving analogies on words: An algorithm. In *COLING-ACL*, pages 728–735, 1998. URL `http://dblp.uni-trier.de/db/conf/acl/acl98.html#Lepage98`.

Yves Lepage and Etienne Denoual. Purest ever example-based machine translation: Detailed presentation and assessment. *Machine Translation*, 19(3):251–282, 2005.

VI Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.

Vito Pirelli and Stefano Federici. "derivational" paradigms in morphonology. 1994.

Vito Pirrelli and François Yvon. The hidden dimension: a paradigmatic view of data-driven nlp. *J. Exp. Theor. Artif. Intell.*, 11(3):391–408, 1999.

Jorma Rissanen. *Stochastic Complexity in Statistical Inquiry Theory*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1989. ISBN 981020311X.

Harold Somers, Sandipan Dandapat, and Sudip Kumar Naskar. A review of ebmt using proportional analogies. In *Proceedings of the 3rd Workshop on Example-Based Machine Translation*, 2009.

Nicolas Stroppa and François Yvon. An analogical learner for morphological analysis. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 120–127, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W05/W05-0616`.

François Yvon. Finite-state transducers solving analogies on words. Technical report, 2003.

François Yvon, Nicolas Stroppa, Arnaud Delhay, and Laurent Miclet. Solving analogical equations on words. Technical report, Paris, France, 2004.