A Logical Approach to Sentiment Analysis
DRAFT

Niklas Christoffer Petersen

Kongens Lyngby 2012
IMM-MSc-2012-????

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-MSc-2012-????

Summary (English)

The goal of the thesis is to ...

Summary (Danish)

Målet for denne afhandling er at ...

Preface

This thesis was prepared at the department of Informatics and Mathematical Modelling at the Technical University of Denmark in partial fulfilment of the requirements for acquiring an MSc in Computer Science and Engineering.

The thesis deals with ...

The thesis consists of ...

Furthermore three project-specific learning objectives for the project concerned should be presented. The following is simply suggestions for those:

- Understand and extend modern techniques for processing of natural language texts using formal logical systems.
- Demonstrate methods for formal reasoning with respect to natural language understanding.
- Present a *proof of concept* system, that is a fully functional implementation of essential theoretical presented methods.

Kgs. Lyngby, September 30, 2012

Niklas Christoffer Petersen

Acknowledgements

I would like to thank my....

Contents

Chapter 1

Introduction

The study of opinion is a one of the oldest fields, with roots in philosophy going back to the Ancient Greek philosophers. The wide adoption of the Internet has made it possible for individuals to express their subjective opinions to a extent much more far-reaching then previous was possible. This has recently been intensified even more due to the explosive popularity of social networks and microblogging services.

The amount of opinions are often huge compared to what traditional opinion analyses, e.g. questionnaire surveys, requires to yield significant results. Furthermore the opinions cover nearly every thinkable topic. This gives an incentive, given that the potential value of such opinions can be great, if information can be extracted effectively and precisely. Given enough opinions on some topic of interest, they can yield significant indication of *collective opinion shifts*, e.g. shifts in market trends, political sympathies, etc. The interest in such shifts is far from recent, and is a well established subfield of the *psychometrics* and has strong scientific grounds in both psychology and statistics.

However, since these opinions are often stated in an informal setting using natural language, usual methods developed for traditional opinion analyses, e.g. questionnaire surveys, cannot by directly applied on the data. The burst of computational power available has meanwhile made it possible to automatically analyze and classify these huge amounts of opinion data. The application of computational methods to extract such opinions are more commonly known as *sentiment analysis*.

This thesis presents a *formal logical approach* to extract the *sentiment* of natural language text reviews. In this chapter traditional methods for data collection of sentiments are briefly considered and thereafter the overall challenges involved in collecting reviews stated in natural language are presented. The opinions considered in this thesis are in form of product and service reviews, however most of the techniques presented can be generalized to other types of topics.

1.1 Classical data collection

One of the most used approaches to collect data for opinion analyses is through questionnaire surveys. Most of us are familiar with such surveys, where the subject is forced to answer questions with a fixed scale. For instance, given the statement "The rooms at the Swissôtel Hotel are of high quality.", a subject must answer by selecting one of a predefined set of answers, e.g. as shown in Figure 1.1.

1. Strongly disagree
2. Disagree
3. Neither agree nor disagree
4. Agree
5. Strongly agree

Figure 1.1: Likert scale.

Such scales, where the subject indicates the *level of agreement*, are know as *Likert scales*, originally presented by Likert [1932], and has been one of the favourite methods of collection data for opinion analyses cf. [?]. Other scales are also widely used, for instance the *Guttman scale* [?], where the questions are binary (yes/no) and ordered such that answering yes to a questions implies the answer yes to all questions ordered below this. Thus the answer on a Guttman scale can likewise be captured by a single index. An example of an Guttman scale is shown in Figure 1.2.

Given a set of answers, the result of such surveys are fairly easy to compute. At its simplest it can be an per question average of the subject's answers, however mostly it is also interesting to connect the questions – for instance how does subjects' answer to the above statement influent their answer to the statement "The food at the Swissôtel Restaurant are of high quality.", etc.

1. I like eating out
2. I like going to restaurants
3. I like going to themed restaurants
4. I like going to Chinese restaurants
5. I like going to Beijing-style Chinese restaurants

Figure 1.2: Guttman scale.

One advantage of using fixed frameworks as the Likert and Guttman scales is that the result of the data collection is highly well-structured, and multiple answers are known to be provided by the same subject. This makes further analysis as the example just mentioned possible, something that will be much harder to achieve when harvesting reviews from the Internet, where the author of the review is presumably unknown, or at least not connected to any other reviews. Furthermore, since most questionnaire surveys are conducted in relatively controlled settings, where the subjects in many cases has been preselected to constitute a representative sample of some population, the results intuitively have relative high certainty.

However these properties also contributes to some of the disadvantages of classical data collection, namely the difficulty of getting people to answer them. Another issue is that people only can answer on the questions that are provided, which mean that significant aspects of the subjects opinion might not be uncovered if it is not captured by a question.

1.2 Natural language data collection

In this thesis it is argued that a far more natural way for subjects to express their opinions is through their most natural communication form, i.e. their language. The strongest incentive for consider natural language texts as a data source is simply the amount of data available through the Internet. This especially includes posts on social networking services and microblogging services, e.g. Facebook[1] and Twitter[2], where people often express the opinion on products and services.

This though introduces the need for efficient candidate filtering as the posts in general

[1] Facebook, `http://www.facebook.com/`

[2] Twitter, `http://www.twitter.com/`

of cause are not constrained to a specific entity or topic of interest. This can be fairly easy achieved as most of the services provides APIs that allows keyword filtering. The approach also raises ethical issues, since the author of the post might never realize that it is being used for the purpose of opinion analysis. Larger texts, such as blog posts, could indeed also be considered, however the contextual aspects of large, contiguous texts often makes interpretation extremely complex, thus making it a difficult task to extract opinions on a specific entity. In this thesis only relatively short reviews are thus considered.

One concern is whether social networking users can constitute a representative sample of the population in question. The actual population of course rely on the target of the analysis. This is a non-trivial study itself, but just to demonstrate the sample bias that often are present consider Figure 1.3. The figure shows the age distribution of respectively Twitter Users and the population of Denmark cf. [Pingdom, 2010] and [Eurostat, 2010]. If the target group was Danes in general, harvesting opinions from Twitter without any correction would presumably cause some age groups to be vastly overrepresented, i.e. the mid-aged Danes, while others would be underrepresented, i.e. young and old Danes.

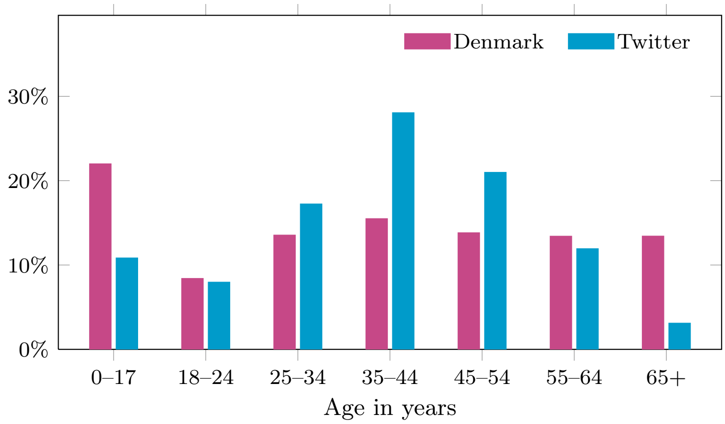


Figure 1.3: Age of Twitter Users and population of Denmark.

Further details on this issue will not be concerned, but it is indeed necessary to correct collected data for sampling bias in order to draw any significant conclusions, such that the distribution of collected opinions indeed follows the target of the analysis.

Another more progressive approach for natural language data collection could be *opinion seeking queries* as the one shown in (1.1). Such queries are intended to ensure succinct reviews that clearly relate to the *entity* in question (e.g. product or service) with respect to a specific *topic of interest*.

What do you think about pricing at the Holiday Inn, London? (1.1)

This method might not seem that different from that of the previously mentioned Likert scales, but it still allows the reviewer to answer with a much broader sentiment and lets the reviewer argue for his/hers answer as shown in the examples (1.2, 1.3).

The price is moderate for the service and the location. (1.2)

Overall an above average hotel based on location and price but not one for a romantic getaway! (1.3)

1.3 Sentiment of a text

This section gives a succinct presentation of sentiment analysis, and introduce it as a research field. The research in sentiment analysis has only recently enjoyed high research activity cf. [Liu, 2007], [Pang and Lee, 2008], which probably is due to a combination of the progress in machine learning research, the availability of huge data sets through the Internet, and finally the commercial applications that the field offers. Liu [2007, chap. 11] identify three *kinds* of sentiment analysis:

- *Sentiment classification* builds on text classification principles, to assign the text a *sentiment polarity*, e.g. to classify the entire text as either positive or negative. This kind of analysis works on *document level*, and thus no details are discovered about the entity of the opinions that may by expressed by the text. The result is somewhat coarse, e.g. it seems to be hard to classify (1.4) as *either* positive or negative, since it contains multiple opinions.

  The buffet was expensive, but the view is amazing. (1.4)

- *Feature-based sentiment analysis* works on *sentence level* to discover opinions about entities present in the text. The analysis still assigns *sentiment polarities*, but on a entity level, e.g. the text (1.4) may be analyzed to express a negative opinion about the *buffet*, and a positive opinion about the *view*.
- *Comparative sentence and relation analysis* focus on opinions that describes similarities or differences of more than one entity, e.g. (1.5).

  The rooms at Holiday Inn are cleaner than those at Swissôtel. (1.5)

The kind of analysis presented by this thesis is closest to the *feature-based sentiment analysis*, however Liu [2007, chap. 11] solely describes methods that uses *mashine learning approaches*, whereas this thesis will present a *formal logical approach*. The difference between these approaches, and arguments for basing the solution on formal logic will be disclosed in the next section, and further details on the overall analytic approach is presented in Chapter 2.

Finally Liu [2007, chap. 11] identify two *ways* of expression opinion in texts, respectively *explicit* and *implicit* sentiments. An explicit sentiment is present when the sentence directly expresses an opinion about a subject, e.g. (1.6), whereas an implicit sentiment is present when the sentence implies an opinion, e.g. (1.7). Clearly sentences can contain a mix of explicit and implicit sentiments.

The food for our event was delicious. (1.6)

When the food arrived it was the wrong order. (1.7)

Most research focus on the explicit case, since identifying and evaluating implicit sentiment is an extremely difficult task which requires a high level of domain specific knowledge, e.g. in (1.7) where most people would regard it as negative if a restaurant served another dish then what they ordered. To emphasize this high domain dependency Pang and Lee [2008] considers the sentence (1.8), which in the domain of *book reviews* imply be positive sentiment, but the exact same sentence implies a negative sentiment in the domain of *movie reviews*.

Go read the book! (1.8)

The thesis will thus focus on the explicit case, since the implicit case was considered to simply require too much domain specific knowledge. This is due to two reasons, firstly the presented solution should be adaptable to *any* domain, and thus tying is too closely to one type of domain knowledge was not an option, secondly the amount of domain knowledge required is in the vast number of cases simply not available, and thus needs to be constructed or collected. With that said the explicit case is neither domain independent, which is a problematic briefly touched in the next section, and detailed in Section 2.4.

1.4 The logical approach

A coarse classification of the different approaches to sentiment analysis is to divide it into two classes: *formal approaches* and *machine learning approaches*. To avoid any confusion this thesis will present a method that belong to the formal class.

- *Formal approaches* models the texts to analyze as a formal language, i.e. using a formal grammar. This allows a syntactical analysis of the texts, yielding the structures of the texts, e.g. sentences, phrases and words for *phrase structure grammars*, and binary relations for *dependency grammars*. Semantic information is then extractable by augmenting and inspecting these structures. The result of the semantic analysis is then subject to the actual sentiment analysis, by identifying positive and negative concepts, and how these modifies the subjects and objects in the sentences.

- *Machine learning approaches* uses feature extraction to train probabilistic models from a set of labeled train data, e.g. a set of texts where each text is labeled as either positive or negative for the *sentiment classification*-kind analysis. The model is then applied to the actual data set of which an analysis is desired. If the feature extracting *really* do captures the features that are significant with respect to a text either being negative or positive, and the texts to analyze has the *same* probability distribution as the training data, then the text will be classified correctly.

Notice that the presented classification only should be interpreted for the process of the actual sentiment analysis, not any preprocessing steps needed in order apply the approach. Concretely the presented formal approach indeed do rely on machine learning techniques in order to efficiently identify lexical properties of the text to analyze as will be covered in Chapter 4.

The motivation for focusing on the formal approach is two-folded: Firstly, different domains can have very different ways of expressing sentiment. What is considered as positive in one domain can be negative in another, and vice-verse. Likewise what is weighted as significant (i.e. either positive or negative) in one domain maybe completely nonsense in another, and again vice-verse. Scientific findings for this are also presented in Chapter 4, but also really follows from basic intuition. Labeled train data are sparse, and since machine learning mostly assumes at least some portion of labeled target data are available this constitutes an issue with the pure machine learning approach. The end result is that the models follows different probability distributions, which complicates the approach, since such biases needs to be corrected, which is not a trivial task.

Secondly, machine learning will usually classify sentiment on document, sentence or simply on word level, but not on a entity level. This can have unintended results when trying to analyze sentences with coordination of sentiments for multiple entities, e.g. (1.4). The machine learning approaches that does try to analyze on entity level, e.g. *feature-based sentiment analysis* by Liu [2007, chap. 11], relies on some fixed window for feature extraction, e.g. Liu [2007, chap. 11] uses n-grams. As a result such methods fails to detect long distance dependencies between an entity and opinion stated about that entity. An illustration of this is shown by the potentially unbound number of *relative clauses* allowed in English, e.g. (1.9), where *breakfast* is described as *best*, however one would need to use a window size of at least 9 to detect this relation, which is much larger then normally considered (Liu only considers up to trigrams).

The breakfast that was served Friday morning was the best I ever had! (1.9)

Formal logical systems are opposed to machine learning extremely precise in results. A conclusion (e.g. the sentiment value for a specific subject in a given text) is only possible if there exists a formal proof for this conclusion.

Thesis 1.1 *It is the thesis that a logical approach will be able to capture these complex relationships between entities and sentiments, thus achieving a more fine-grained sentiment analysis.*

■

With that said, a logical approach indeed also suffers from obvious issues, most notable robustness, e.g. if there are missing, or incorrect axioms a formal logical system will not be able to conclude anything, whereas a machine learning approach will always be able to give an estimate, which might be a very uncertain estimate, but at least a result. This issue of robustness is crucial in the context of review texts, since such may not always be grammatical correct, or even be constituted by sentences. In Section 1.6 this issue will be addressed further, and throughout this thesis it will be a returning challenge. Details on the logical approach is presented in Chapter 3

1.5 Related work

In the following notable related work on sentiment analysis are briefly presented. As mentioned there are two main flavors of sentiment analysis, namely implicit and explicit. Most of the work fund focus solely on the explicit kind of sentiment, just like this work does.

Furthermore it seems that there is a strong imbalance between the *formal approaches* and *machine learning approaches*, with respect to amount of research, i.e. there exists a lot of research on sentiment analysis using machine leaning compared to research embracing formal methods.

Notably related work using formal approaches include Tan *et al.* [2011], who presents a method of extracting sentiment from dependency structures, and also focus on capturing long distance dependencies. As dependency structures simply can be seen as binary relations on words, it is indeed a formal approach. However what seems rather surprising is that in the end they only classify on sentence-level, and thus in this process loose entity of the dependency.

The most similar work on sentiment analysis found using a formal approach is the work by Simančík and Lee [2009]. The paper presents a method to detect sentiment of newspaper headlines, also using a computational logic approach (in fact the same grammar formalism that later will be presented and used in this work). The paper focus on some specific problems arising with analyzing newspaper headlines, e.g. such as headline texts often do not constitute a complete sentence, etc. However the paper also present more general methods, including a method for building a highly covering map from words to polarities based on a small set of positive and negative seed words. This method has been adopted by this thesis, as it solves the assignment of polarity values on the lexical level quite elegantly, and is very loosely coupled to the domain. However their actual semantic analysis, which unfortunately is described somewhat shallow in the paper, seem to suffers from severe problems with respect to curtain phrase structures, e.g. relative clauses.

1.6 Using real data sets

For the presented solution to be truly convincing it is desired to present a fully functional *proof of concept* implementation that shows at least the most essential capabilities. However, for such product to be demonstrated properly, real data is required. Testing in on some tiny pseudo data set constructed for the sole purpose of this demonstration would not be convincing. Chapter 5 presents essential aspects of this *proof of concept* implementation.

An immediate concern that raises when dealing with real data sets is the possibility of incorrect grammar and spelling. A solution that would only work on *perfect texts* (i.e. text with perfectly correct grammar and spelling) would not be adequate. Reasons for this could be that word is simply unpresent from the system's vocabulary (e.g. misspelled), or on a grammatical incorrect form (e.g. wrong person, gender, tense, case, etc.).

Dealing with major grammatical errors, such as wrong word order is a much harder problem, since even small changes in for instance the relative order of subject, object, verb etc. may result in an major change in interpretation. Thus it is proposed, only to focus on minor grammatical errors such as incorrect form. Chapter 6 presents a evaluation of the implementation on actual review data.

Chapter 2

Sentiment analysis

An continuous analog to the sentiment polarity model presented in the introduction is to weight the classification. Thus the polarity is essential a value in some predefined interval, $[-\omega; \omega]$, as illustrated by Figure 2.1. An opinion with value close to $-\omega$ is considered highly negative, whereas a value close to ω is considered highly positive. Opinions with values close to zero are considered almost neutral. This model allows the overall proccess of the sentiment analysis presented by this thesis to be given by Definition 2.1.

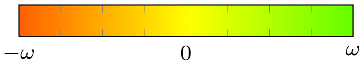


Figure 2.1: Continuous sentiment polarity model.

Definition 2.1 A sentiment analysis $\mathcal{A}$ is a computation on a review text $T \in \Sigma^\star$ with respect to a subject $s \in S$, where $\Sigma^\star$ denotes the set of all permissible texts in the language. The result is an normalized score as shown in (2.1). The yielded score should reflect the *polarity* of the subject in the text, i.e. whether the overall opinion is positive, negative, or neutral.

$$\mathcal{A} : \Sigma^\star \to S \to [-\omega; \omega] \tag{2.1}$$

■

It should be evident that this computation is far from trivial, and constitutes the cornerstone of this thesis. There are several steps needed, if such computation should yield any reasonable result. As mentioned in the introduction the goal is a logical approach for achieving this. The following outlines the overall steps to be completed, their associated problematics in this process, and succinctly presents different approaches to solve each step. The chosen approach for each step will be presented in much more details in later chapters. Finally ... Something about data acquisition of test-data

2.1 Tokenization

In order the even start processing natural language texts, it is essential to be able to identify the elementary parts, i.e. *lexical units* and *punctuation marks*, that constitutes a text. Decent tokenization is essential for all subsequent steps. However even identifying the different sentences in a text can yield a difficult task. Consider for intance the text (2.2) which is taken from the Wall Street Journal (WSJ) corpus [Paul and Baker, 1992]. There are six periods in it, but only two of them indicates sentence bounderies, and delimits the text into its two sentences.

> *Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29. Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group.* (2.2)

The domain of small review texts allows some restrictions and assumptions, that at least will ease this issue. For instance it is argued that the review texts will be fairly succinct, and thus it seems like a valid assumption that they will consists of only a few sentences. Its is argued that this indeed is achievable by sufficient instructing and constraining the reviewers doing data collection, e.g. only allowing up to a curtain number of characters. This allows sentences in such phrases to be proccesed independently (i.e. as two seperate review texts).

Even with this assumption, the process of identifying the sentences, and the lexical units and punctuation marks within them, is not a trivial task. Webster and Kit [1992] criticizes the neglection of this process, as most NLP studies focus purely on analysis, and assumes this process has already been performed. Such common assumptions might derive from English being a relatively easy language to tokenize. This is due to its space marks as explicit delimiters between words, as opposed to other languages, e.g. Chinese which has no delimiters at all. This might hint that tokenization is very language dependent. And even though English is considered simple to tokenize, a naive approach like segmenting by the occurrence of spaces fails for the text (2.3), which is also from the WSJ corpus, as it would yield lexical units such as “(or”, “perceived,” and “rate),”. Simply consider all groups of non-alphanumerics as punctuation marks does not work either, since this would fail for i.a. ordinal numbers, currency symbols, and abbreviations, e.g. “Nov.” and “Elsevier N.V.” in text (2.2). Both of these methods also fail to recognize “Pierre Vinken” and “Elsevier N.V.” as single proper noun units, which is arguably the most sane choice for such.

> *One of the fastest growing segments of the wine market is the category of superpremiums – wines limited in production, of exceptional quality (or so perceived, at any rate), and with exceedingly high prices.* (2.3)

Padró *et al.* [2010] presents a framework of analytic tools, developed in the recent years, for various NLP tasks. Specially interesting is the morphological analyzer, which applies a cascade of specialized (i.e. language dependent) processors to solve exactly the tokenization. The most simple use pattern matching algorithms to recognize numbers, dates, quantity expressions (e.g. ratios, percentages and monetary amounts), etc. More advanced processing are needed for proper nouns, which relies on a two-level solution: first it applies a fast pattern matching, utilizing that proper nouns are mostly capitalized; and secondly a more advanced ... TODO Cite: (Named entity extraction using adaboost). These recognize proper nouns with accuracy of respectively 90% and over 92%. The analyzer also tries to identify lexical units that are composed of multiple words, e.g. proper nouns and idioms.

It is thus possible, by the use of this framework, to preprocess the raw review texts collected from users, and ensure that they will be tokenized into segments that are suitable for the lexical-syntactic analysis. Thus more details on the tokenizer as regards design will not be presented.

2.2 Lexical-syntactic analysis

The syntactic analysis determines the grammatical structure of the input texts with respect to the rules of the English language. It is expected that the reader is familiar with English grammar rules and syntactic categories, including phrasal categories and lexical categories (also called parts of speech). As mentioned erlier it is essential that the chosen solution is able to cope with *real* data, collected from actual review scenarios. This implies a robust syntactic analysis accepting a large vocabulary and a wide range of sentence structures. In order to calculate the actual polarity it is essential to have semantic annotations on the lexical units. It is argued that a feasible and suitable solution is to use a grammar that is *lexicalized*, i.e. where the rules are essentially langauge independent, and the syntactic properties are derived from a lexicon. Thus the development of a lexicalized grammar is mainly a task of acquiring a sutable lexicon for the desired language.

Even though the task of syntactic analysis now is largely reduced to a task of lexicon acquisition, which will be addressed in chapter 3, there are still general concerns that are worth acknowledging. Hockenmaier *et al.* [2004, p. 108-110] identifies several issues in being able to efficiently handle natural language texts solely with lexicalized grammars, mainly due to the need for entries for various combinations of proper nouns, abbreviated terms, dates, numbers, etc. Instead they suggest to use pattern matching and statistical techniques as a preprocessing step, for which efficient components exists, which translate into reduced complexity for the actual syntactic analysis.

Even though it can be argued that the use of proper nouns and specific dates is fairly limited in review text, in that a context have already been established for the reviewer cf. section 1.2, it is still

However the domain of small review texts also introduce concerns that are absent from other domains, including the possebility of incorrect grammar and spelling, since the texts comes unedited from humans with varying English skills. A solution that would only work on *perfect texts* (i.e. texts of sentences with completely correct grammar and spelling) would not be adequate. In order to at least try to handle minor misspellings it is intended to use algoritms that can select alternatives from the lexicon. Reasons for this could be that word is simply unpresent from the system's vocabulary (e.g. misspelled), or on a grammatical incorrect form (e.g. wrong person, gender, tense, case, etc.).

2.3 Mildly context-sensitive grammars

There exists formal proofs that some natural language structures requires formal power beyond *context-free grammars* (CFG), i.e. [Shieber, 1985] and [Bresnan *et al.*, 1982]. Thus the search for grammars with more expressive power has long been a major study within the field of computational linguistics. The goal is a grammar that is so restrive as possible, allowing efficient syntactic analysis, but still capable of capturing these structures. The class of *mildly context-sensitive grammars* are conjectured to be powerful enough to model natural languages while remaining efficient with respect to syntactic analysis cf. [Joshi *et al.*, 1990].

Different grammar formalisms from this class has been considered, including *Tree Adjunct Grammar* (TAG) [Joshi *et al.*, 1975], in its lexicalized form (LTAG), *Head Grammar* (HG) [Pollard, 1984] and *Combinatory Categorial Grammar* (CCG) [Steedman, 1998]. It has been shown that these are all equal in expressive power by Vijay-Shanker and Weir [1994]. The grammar formilism chosen for the purpose of this thesis is *Combinatory Categorial Grammar* (CCG), pioneered largely by Steedman [2000]. CCG adds a layer of combinatory logic onto pure Categorial Grammar, which allows an elegant and succinct formation of *higher-order* semantic expressions directly from the syntactical analysis. Since the goal of this thesis is a logical approach to sentiment anaylsis, CCG's native use of combinatory logic seemed like the most reasonable choice. Chapter 3 will formally introduce the CCG in much more detail.

2.4 Semantic analysis

The overall process of semantic analysis in the context of sentiment analysis is to identify the polarity of the entities appearing in the text, and to relate these entities to the subject of the sentiment analysis. The approach is to *annotate* the lexical units of adjectives adverbs with suitable polarities, and then fold these onto the phrasal structures, yielded by the syntactical analysis, in order to identify the bindings of these polarities, i.e. which subjects/objects they modify directly or indirectly.

There exists datasets that tries to bind a general polarity to each word in a lexicon, e.g. [Esuli and Sebastiani, 2006] and [Baccianella *et al.*, 2010]. While such might be fine for general sentiment analyses, or analysis where the domain is not known, it is argued that better results can be achieved by using a domain specific annotation. For instance the adjective "huge" might be considered positive for a review describing rooms at a hotel, while negative for a review describing sizes of cell phones.

As already mentioned, the use of a lexicalized syntactic analysis allows the annotation to appear directly on the entries in the lexicon. A manual annotation of a large lexicon is evidently not a feasible approach. Furthermore the model must also be generic enough so it can be adapted to ideally any domain contexts with minimum efforts, i.e. it is not desired to tie the model to any specific domain, or type of domains. To achieve such a model that is loosely coupled to the domain the concept of *semantics networks* was chosen cf. Russell and Norvig [2009, p. 454–456].

A semantic network is in its simplest form just a collection of different semantic concepts, and relations between them. The idea is to dynamically construct such semantic networks from a small set of domain specific knowledge, namely a set of positive and negative seed concepts in the domain – a technique presented by Simančík and Lee [2009]. Furthermore such semantic networks can also be used to relate entities to the subject of interest. Section 4.3 in Chapter 4 will presents details on the approach of calculating the polarities of adjectives and adverbs and additionally present some handling of negations.

The final result of the sentiment analysis is simply the aggregation of the results yielded for each of the results of the semantic analysis.

Chapter 3

Combinatory categorial grammar

In this chapter the formalishm of Combinatory Categorial Grammar (CCG) is introduced, and based on this applied to the proposed sentiment analysis introduced in the previous chapter. For the purpose of explaining and demonstrating CCG a small fragment of English is used. This allow the usage of a "handwritten" lexicon initially. In chapter 4 the issues related to acquiring, and analyzing with, a wide coverage lexicon are addressed. For the syntactic analysis a CCG lexicon is defined as follows:

Definition 3.1 A CCG lexicon, $\mathcal{L}_{\text{CCG}}$, is mapping from a lexical unit, $w \in \Sigma^\star$, to a set of 2-tuples, each containing a lexical category and semantic expression that the unit can entail cf. (3.1), where Γ denotes the set of lexical and phrasal categories, and Λ denotes the set of semantic expressions.

$$\mathcal{L}_{\text{CCG}} : \Sigma^\star \to \mathcal{P}(\Gamma \times \Lambda) \tag{3.1}$$

■

A *tagging* of a lexical unit $w \in \Sigma^\star$ is simply the selection of one of the pairs yielded by $\mathcal{L}_{\text{CCG}}(w)$. Thus given some ordered set of lexical units, which constitutes the text $T \in \Sigma^\star$ to analyse, there might exists many different taggings. This is simply due to the fact that a lexical unit can entail different lexical categories (e.g. "service" is both a noun and a verb), and different semantic expressions (e.g. the noun "service" can both refer to assistance and tableware). The number of taggings can thus be large, but is always finite.

The set of lexical and phrasal categories, Γ, are of a somewhat advanced structure in the CCG presented, since it follows recent work by Baldridge and Kruijff [2003] to incorporate *modalities*. A category is either *primitive* or *compound*. The set of primitive categories, $\Gamma_{\text{prim}} \subset \Gamma$, is language dependent and, for the English desired to be covered by this thesis, it consists of S (sentence), NP (noun phrase), N (noun) and PP (prepositional phrase). Compound categories are recursively defined by the infix operators $/_\iota$ (forward slash) and $\backslash_\iota$ (backward slash), i.e. if α and β are members of Γ, then so are $\alpha/_\iota\beta$ and $\alpha\backslash_\iota\beta$. This allows the formation of all other lexical and phrasal categories needed. The operators are left associative, but to avoid confusion inner compound categories are always encapsulated in parentheses througout this thesis.

The basic intuitive interpretation of $\alpha/_\iota\beta$ and $\alpha\backslash_\iota\beta$ is as a function that takes a category β as argument and yields a result of category α. Thus the argument is always stated on the right side of the operators, and the result on the left. The operator determines the dictionality of the application, i.e. *where* the argument should appear relative to the function: the forward operator ($/_\iota$) denotes that the argument must appear on the right of the function, whereas the backward operator ($\backslash_\iota$) denotes that the argument must appear on the left. The subscript, ι, denotes the *modality* of the operator, which is a member of a finite set of modalities $\mathcal{M}$ and will be utilized to restrict acceptence in the next section.

The syntactic categories constitutes a type system for the semantic expressions, with a set of primitive types, $\mathcal{T}_{\text{prim}} = \{\tau_x \mid x \in \Gamma_{\text{prim}}\}$. Thus, if a lexicon entry has category $(N\backslash_\iota N)/_\iota(S/_\iota NP)$ then the associated semantic expression must honor this, and have type $(\tau_{\text{NP}} \rightarrow \tau_{\text{S}}) \rightarrow \tau_{\text{N}} \rightarrow \tau_{\text{N}}$ ($\rightarrow$ is right assosiative). This is a result of the *Principle of Categorial Type Transparency* [?, Montague, 1974], and the set of all types are denoted $\mathcal{T}$. For now it is sufficient to describe the set of semantic expressions, Λ, as the set of *simply-typed* λ-expressions, Λ', cf. Definition 3.2. In section 3.4 this is extended to support the desired sentiment analysis.

Definition 3.2 The set of simply typed λ-expressions, Λ', is defined recursively as the set of expressions e, where e is either a variable x from an infinite set of typed variables $\mathcal{V} = \{v_1 : \tau_\alpha, v_2 : \tau_\beta, \ldots\}$, a function abstraction, or a functional application.

$$\begin{array}{rcll} x : \tau \in \mathcal{V} & \Rightarrow & x : \tau \in \Lambda' & \text{(Variable)} \\ x : \tau_\alpha \in \mathcal{V},\ E : \tau_\beta \in \Lambda' & \Rightarrow & \lambda x.E : \tau_\alpha \rightarrow \tau_\beta \in \Lambda' & \text{(Abstraction)} \\ E_1 : \tau_\alpha \rightarrow \tau_\beta \in \Lambda',\ E_2 : \tau_\beta \in \Lambda' & \Rightarrow & (E_1 E_2) : \tau_\alpha \in \Lambda' & \text{(Application)} \end{array}$$

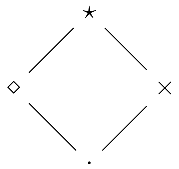
■

3.1 Combinatory rules

CCGs can be seen as a deductive proof system where the axioms are members of $\Gamma \times \Lambda$. A text $T \in \Sigma^\star$ is accepted as a sentence in the language, if there exists a deductive proof for S, for some tagging of T.

The inference rules of the proof system are known as *combinators*, since they take one or more function pairs, in the form of instances of $\Gamma \times \Lambda$, and produces new instances from the same set. The combinators determines the expressive power of the grammar. A deep presentation of which rules are *needed*, and thus the linguistic motivation behind this, is out of the scope of this thesis. In the following essential combinators covered by Steedman [2011, chap. 6] are succinctly described, which constitutes a *midely context-sensitive* class grammar. These are the development of the combinatory rules Steedman presented in [2000, chap. 3], however with significant changes with respect to coordinating conjucntions, due to the introduction of modalities on the infix operators.

The set of modalities used, $\mathcal{M}$, follows [Baldridge and Kruijff, 2003] and [Steedman, 2011], where $\mathcal{M} = \{\star, \diamond, \times, \cdot\}$. The set is partially ordered cf. the lattice (3.2).



(3.2)

The basic concept of annotating the infix operators with $\iota \in \mathcal{M}$, is to restrict the application of inferrence rules during deduction in order ensure the soundness of the system. The $\star$ is the most restrictive, allowing only basic rules, $\diamond$ allows rules which perserves the word order, $\times$ allows rules which permutate the word order, and finally $\cdot$ allows any rule without restrictions. The partial ordering allows the most restrictive categories to also be included in the less restrictive, e.g. any rule that assumes $\alpha/_\diamond\beta$ will also be valid for $\alpha/_\star\beta$. Since $\cdot$ permits any rule it is convenient to simply write $/$ and $\backslash$ instead of respectively $/.$ and $\backslash.$, i.e. the dot is omitted from these operators.

The simplest combinator is the *functional application*, which simply allows the instances to be used as functions and arguments, as already described. The forward and backward functional application combinator can be formulated as respectivly ($>$) and ($<$), where X and Y are variables ranging over lexical and phrasal categories, and f and a are variables ranging over semantic expressions. Since the operators are annotated with $\star$, the rules can apply to even the most restrictive categories. For readability instances (α, e) of $\Gamma \times \Lambda$ is written $\alpha : e$. Notice that since the semantic expressions are typed, the application of a to f is sound.

$$X/_{\star}Y : f \qquad Y : a \quad \Rightarrow \quad X : f\,a \tag{>}$$
$$Y : a \qquad X\backslash_{\star}Y : f \quad \Rightarrow \quad X : f\,a \tag{<}$$

With only these two simple combinatory rules, (>) and (<), the system is capable of capturing any context-free langauge cf. Steedman [2000, p. 34]. For the fragment of English, used to demonstrate CCG, the lexicon is considered to be finite, and it is thus possible, and also convinient, to simply write the mapping of entailment as a subset of $\Sigma^{\star} \times \Gamma \times \Lambda$. Figure 3.1 shows a fragment of this demonstration lexicon. For readability, instances (w, α, e) of $\Sigma^{\star} \times \Gamma \times \Lambda$ is written $w \models \alpha : e$. Notice that the semantic expressions are not yet specified, since it for now is sufficient that just the type of the expressions is correct, and this follows implicitly from the category of the entry.

$$\begin{aligned}
\textbf{the} &\models NP/_{\diamond}N : (\ldots) && \text{(Determiners)}\\
\textbf{an} &\models NP/_{\diamond}N : (\ldots) &&\\
\textbf{hotel} &\models N : (\ldots) && \text{(Nouns)}\\
\textbf{service} &\models N : (\ldots) &&\\
\textbf{had} &\models (S\backslash NP)/NP : (\ldots) && \text{(Transative verbs)}\\
\textbf{exceptional} &\models N/N : (\ldots) && \text{(Adjectives)}
\end{aligned}$$

Figure 3.1: A fragment of a tiny handwritten lexicon.

The lexicon for instance shows how determiners can be modeled by the category which takes a noun on the right and yields a noun phrase. Likewise a transitive verb is modeled by a category which first takes a noun phrase on the right (the object), then a noun phrase on the left (the subject) and lastly yields a sentence. Figure 3.2 shows the deduction of S from the simple declarative sentence “the hotel had an exceptional service” (semantics are omitted).

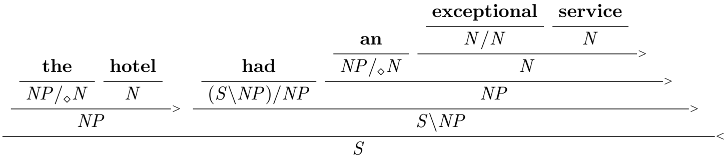


Figure 3.2: Deduction of simple declerative sentence.

Besides functional application, CCG also has a set of more restrictive rules, including *functional composition*, defined by the forward and backward functional composition combinators, respectively ($>_{\mathbf{B}}$) and ($<_{\mathbf{B}}$), where Z likewise is a variable ranging over Γ, and g over Λ.

$$\begin{aligned} X/_{\diamond}Y : f \quad Y/_{\diamond}Z : g \quad &\Rightarrow \quad X/_{\diamond}Z : \lambda a.f(g\ a) && (>_{\mathbf{B}}) \\ Y\backslash_{\diamond}Z : g \quad X\backslash_{\diamond}Y : f \quad &\Rightarrow \quad X\backslash_{\diamond}Z : \lambda a.f(g\ a) && (<_{\mathbf{B}}) \end{aligned}$$

Notice that the semantic expression yielded by ($>_{\mathbf{B}}$) and ($<_{\mathbf{B}}$) is equivalent to regular functional composition ($\circ$) of f and g, but since $f \circ g \notin \Lambda$ they need to be written as λ-expressions.

Functional composition is often used in connection with another rule, namely *type-raising*, defined by the forward and backward type-raising combinators, respectively ($>_{\mathbf{T}}$) and ($<_{\mathbf{T}}$), where T is a variable ranging over categories.

$$\begin{aligned} X : a \quad &\Rightarrow \quad T/_{\iota}(T\backslash_{\iota}X) : \lambda f.fa && (>_{\mathbf{T}}) \\ X : a \quad &\Rightarrow \quad T\backslash_{\iota}(T/_{\iota}X) : \lambda f.fa && (<_{\mathbf{T}}) \end{aligned}$$

Type-rasing allows a often primitive category, X, to raise into a category that instead captures a compound category, which is a function over X. The modality of the result is not controllable and is thus often suppressed, however any constrains of the applicability of X of cause continue cf. [Baldridge and Kruijff, 2003].

Notice that the introduction of these rules, i.e. function composition and type-raising, allows deductional ambiguity, i.e. a proof for some sentence may be achievable by multiple deductions as shown in Figure 3.3. However such ambiguities are immaterial, since they do not correspond to semantic ambiguity.

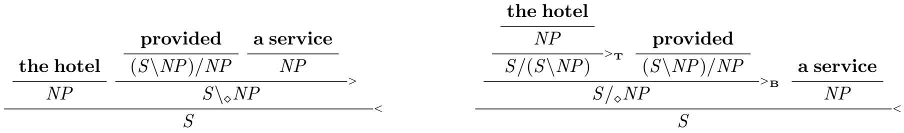


Figure 3.3: Multiple deductions of the same sentence.

A system with these rules demonstrates what is arguably CCG's most unique advantage, namely the ability to handle *unbounded dependencies* without any additional lexicon entries. For instance a transitive verb, with the *same* category as shown in Figure 3.1, can participate in relative clauses as shown in Example 3.1, given the presence of a small set of entries for relative pronouns, e.g. Figure 3.4.

$$\textbf{that} \models (N\backslash_{\diamond}N)/(S/_{\diamond}NP) : (\ldots) \qquad \text{(Relative pronouns)}$$
$$\textbf{that} \models (N\backslash_{\diamond}N)/(S\backslash_{\diamond}NP) : (\ldots)$$

Figure 3.4: Fragment of lexicon for the relative pronoun “that”.

Example 3.1 *Figure 3.5 shows an example of both type-rasing and functional composition. The transitive verb (provided) is requiring an object in the form of a noun phrase to its right. However, since it participate in a relative clause, its object is given by the noun that the clause modifies. Type raising allows the subject of the relative clause to raise into a category that can compose with the verb, and thus allows the relative pronoun (that) to bind the relative clause to the noun.*

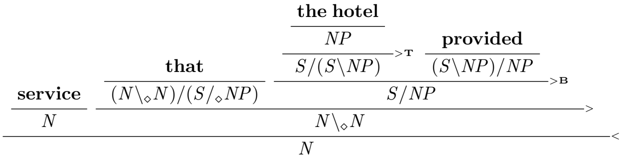


Figure 3.5: Deduction of noun phrase with relative clause.

■

The last set of rules presented here is the *crossed functional composition*, defined by the forward and backward crossed functional composition combinators, respectively ($>_{\mathbf{B}_\times}$) and ($<_{\mathbf{B}_\times}$).

$$X/_{\times}Y : f \quad Y\backslash_{\times}Z : g \quad \Rightarrow \quad X\backslash_{\times}Z : \lambda a.f(g\ a) \qquad (>_{\mathbf{B}_\times})$$
$$Y/_{\times}Z : g \quad X\backslash_{\times}Y : f \quad \Rightarrow \quad X/_{\times}Z : \lambda a.f(g\ a) \qquad (<_{\mathbf{B}_\times})$$

Crossed functional composition allows *permutation* of the word order. This is usefull to allow adverbs in sentences with shifting of heavy noun phrases as shown in Example 3.2.

Example 3.2 *Normally an adverb is put after the object of the verb it modifies in English, e.g. "the hotel served breakfast daily". However if the object of the verb becomes "heavy" it may sometimes be moved to the end of the sentence, e.g. "the hotel served daily a large breakfast with fresh juice".*

In such cases the adverb needs to compose with the verb, before the verb combines with its object. The crossed functional composition allows exatly such structures as shown in Figure 3.6.

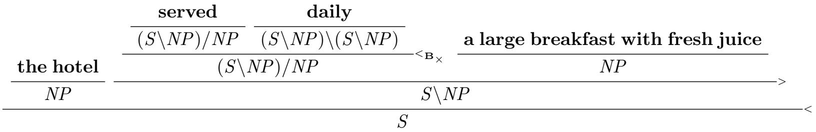


Figure 3.6: Deduction of "heavy" noun phrase shifting.

■

Steedman [2000; 2011] introduces a few additional combinators to capture even more "exotic" linguistic phenomena. Recollect that the rules are language independent, and indeed some of the additional phenomena covered by Steedman is either considered infrequent (e.g. *parasitic gaps*), or even absent (e.g. *cross-serial dependencies*), from the English language desired to cover by this sentiment analysis. It will later be shown (Chapter 4) that the rules already presented indeed cover a substantial part of English.

3.2 Coordination

As mentioned in the introduction, one of the goals is to correctly capture the sentiment of entities in sentences with coordination of multiple opinions.

Coordination by appearance of a coordinating conjunction, such as *and*, *or*, *but*, punctuation and comma, etc., can be modeled simply by the intuition that such should bind two constituents of same syntactic category, but with different semantic expressions, and yield a result also of that category. Some examples of the *and* coordinating conjunction is shown in Figure 3.7.

$$\begin{aligned}
\textbf{and} &\models (S\backslash_{\star}S)/_{\star}S : (\ldots) && \text{(Conjunctions)}\\
\textbf{and} &\models (N\backslash_{\star}N)/_{\star}N : (\ldots)\\
\textbf{and} &\models (NP\backslash_{\star}NP)/_{\star}NP : (\ldots)\\
&\ldots
\end{aligned}$$

Figure 3.7: Fragment of lexicon for the coordinating conjunction "and".

It now becomes evident, why the modalities are needed, since application of the crossed composition combinators without any restrictions could allow scrambled sentences to be deducted falsely, e.g. Figure 3.8.

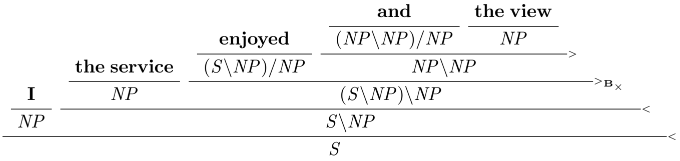


Figure 3.8: Unsound deduction of sentence given absence of modalities.

Similar pit-falls are possible if unresticted application of ($>_{\mathbf{B}}$) and ($<_{\mathbf{B}}$) was allowed, as shown by Baldridge [2002, chap. 4] for the Turkish language. This justifies the requirement for the modalities Baldridge originally proposed in [2002, chap. 5] and Baldridge and Kruijff presented in a refined version in [2003].

3.3 Features and agreement

The syntactic analysis until now has concerned the acceptable order of lexical units based on their categories. However, to guarantee that the accepted phrases indeed follows correct grammar, the *features* of the lexical units must also *agree*. The set of features that might apply is language dependent, for instance most indo-european languages state features for person (e.g. 1st, 2nd or 3rd), number (e.g. singular or plural), gender (e.g. male or female), etc. To incorporate this the primitive categories, Γ_{prim}, cannot be seen as atomic entities, but instead as structures that carries features, e.g. S_{dcl} and $NP_{\text{sg,3rd}}$ denotes respectively a *declarative* sentence, and a *singular, 3rd-person* noun phrase. A set of features *agrees* with another if they do not contain different elements of the same *kind*. For instance $NP_{\text{sg,3rd}}$ agree with NP_{sg}, but not with $NP_{\text{pl,3rd}}$, etc.

However, as mentioned in Section 2.2, a strict enforcement is not intended for the purpose of sentiment analysis, e.g. reviews containing small grammatical errors, such as wrong number as shown in (3.3), should not be discarded simply for this reason.

$$\textit{The hotel have great service} \tag{3.3}$$

However completely ignoring the features is neither an option. A evident demonstration of this is the usage of *predicative adjectives*, e.g. adjectives that modify the subject in a sentence with a *linking verb* as shown in Figure 3.9. Without the correct features, having such entries in the lexicon would allow sentences as “the hotel great”, which of cause is not desired. The linguistic background for the which features are considered necessary for English is not within the scope of this thesis, but one is given by Hockenmaier in [2003], and that feature-set will be used.

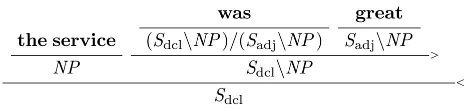

was
great
the service
$(S_{\mathrm{dcl}}\backslash NP)/(S_{\mathrm{adj}}\backslash NP)$
$S_{\mathrm{adj}}\backslash NP$
>
NP
$S_{\mathrm{dcl}}\backslash NP$
<
S_{dcl}

Figure 3.9: Sentence with predicative adjective.

3.4 Extending the semantics

The CCG presented in the previous sections has been based on established literature, but in order to apply the grammar formalism to the area of sentiment analysis the expressive power of the semantics need to be adapted to this task. Until now the semantics has not been of major concern, recall that it just was defined as simply typed λ-expressions cf. Definition 3.2. Furthermore the actual *semantics* of these semantic expressions has not been disclosed, other than the initial use of λ-expressions might hint that ordinary conventions of such presumably apply. The semantic expressions are given by Definition 3.3.

Definition 3.3 The set of semantic expressions, Λ, are defined as a superset of Λ' (see Definition 3.2). Besides variables, function abstraction and functional application, the following structures are available: a n-ary *functor* ($n \geq 0$) with name f from an infinite set of functor names, polarity $j \in [-\omega; \omega]$, and *impact argument* k ($0 \leq k \leq n$); a *sequence* of n semantic expressions of the *same* type; the *change of impact*, the *change* of an expression's polarity; and the *scale* of an expression's polarity. The magnitude of which a expression's polarity may scale is given by $[-\psi; \psi]$.

$$
\begin{array}{rcll}
E_1, \ldots, E_n \in \Lambda, 0 \leq k \leq n,\ j \in [-\omega; \omega] & \Rightarrow & f_j^k(E_1, \ldots, E_n) \in \Lambda & \text{(Functor)} \\
E_1 : \tau, \ldots, E_n : \tau \in \Lambda & \Rightarrow & \langle E_1, \ldots, E_n \rangle : \tau \in \Lambda & \text{(Sequence)} \\
E : \tau \in \Lambda, 0 \leq k' & \Rightarrow & E^{\rightsquigarrow k'} : \tau & \text{(Impact change)} \\
E : \tau \in \Lambda,\ j \in [-\omega; \omega] & \Rightarrow & E_{\circ j} : \tau \in \Lambda & \text{(Change)} \\
E : \tau \in \Lambda,\ j \in [-\psi; \psi] & \Rightarrow & E_{\bullet j} : \tau \in \Lambda & \text{(Scale)}
\end{array}
$$

The semantics includes normal α-conversion and β-, η-reduction as shown below, where $E_1[x \mapsto E_2]$ denotes the *safe* substitution of x with E_2 in E_1, and $FV(E)$ denotes the set of free variables in E. For details see for instance [?].

$$
\begin{array}{rcllr}
\lambda x.E & \Rightarrow & \lambda y.E[x \mapsto y] & y \notin FV(E) & (\alpha\text{-conversion}) \\
\lambda x.E_1 E_2 & \Rightarrow & E_1[x \mapsto E_2] & & (\beta\text{-reduction}) \\
\lambda x.Ex & \Rightarrow & E & x \notin FV(E) & (\eta\text{-reduction})
\end{array}
$$

More interesting are the rules that actually allow the binding of polarities to the phrase structures. The *change of a functor* itself is given by the rule (FC1), which apply to functors with, impact argument, $k = 0$. For any other value of k the functor acts like a non-capturing enclosure that passes on any change to its k'th argument as follows from (FC2). The *change of a sequence* of expressions is simply the change of each element in the sequence cf. (SC). Finally it is allowed to *push change* inside an abstraction as shown in (PC), simply to ensure the applicability of the β-reduction rule.

$$
\begin{array}{rcll}
f_j^0(E_1, \ldots, E_n)_{\circ j'} & \Rightarrow & f^0_{j \widehat{+} j'}(E_1, \ldots, E_n) & \text{(FC1)} \\
f_j^k(E_1, \ldots E_n)_{\circ j'} & \Rightarrow & f_j^k(E_1, \ldots, E_{k \circ j'}, \ldots E_n) & \text{(FC2)} \\
\langle E_1, \ldots, E_n \rangle_{\circ j'} & \Rightarrow & \langle E_{1 \circ j'}, \ldots, E_{n \circ j'} \rangle & \text{(SC)} \\
(\lambda x.E)_{\circ j'} & \Rightarrow & \lambda x.(E_{\circ j'}) & \text{(PC)}
\end{array}
$$

Completely analogue rules are provided for the scaling as shown in respectively (FS1), (FS2), (SS) and (PS). Notice that these *change* and *scale* rules are type preserving.

$$
\begin{array}{rcll}
f_j^0(E_1, \ldots, E_n)_{\bullet j'} & \Rightarrow & f^0_{j \widehat{\cdot}\, j'}(E_1, \ldots, E_n) & \text{(FS1)} \\
f_j^k(E_1, \ldots E_n)_{\bullet j'} & \Rightarrow & f_j^k(E_1, \ldots, E_{k \bullet j'}, \ldots E_n) & \text{(FS2)} \\
\langle E_1, \ldots, E_n \rangle_{\bullet j'} & \Rightarrow & \langle E_{1 \bullet j'}, \ldots, E_{n \bullet j'} \rangle & \text{(SS)} \\
(\lambda x.E)_{\bullet j'} & \Rightarrow & \lambda x.(E_{\bullet j'}) & \text{(PS)}
\end{array}
$$

It is assumed that the addition and multiplication operator, respectively $\widehat{+}$ and $\widehat{\cdot}$, always yields a result within $[-\omega;\omega]$, even if the pure addition and multiplication might not be in this range cf. (3.4) and (3.5).

$$j \widehat{+} j' = \begin{cases} -\omega & \text{if } j + j' < -\omega \\ \omega & \text{if } j + j' > \omega \\ j + j' & \text{otherwise} \end{cases} \tag{3.4}$$

$$j \widehat{\cdot} j' = \begin{cases} -\omega & \text{if } j \cdot j' < -\omega \\ \omega & \text{if } j \cdot j' > \omega \\ j \cdot j' & \text{otherwise} \end{cases} \tag{3.5}$$

Finally the *change of impact* allows change of a functors impact argument cf. (IC).

$$f_j^k(E_1, \dots E_n)^{\leadsto k'} \quad \Rightarrow \quad f_j^{k'}(E_1, \dots E_n) \tag{IC}$$

■

The presented definition of semantic expressions allow the specification ...

Lead up to Example 3.3 and 3.4 ...

Example 3.3 ...

Write example text for simple declarative sentence ... The entity "service" is modifyied by the adjective ... yada yada ... Notice that nouns, verbs, etc. are reduced to their lemma for functor naming...

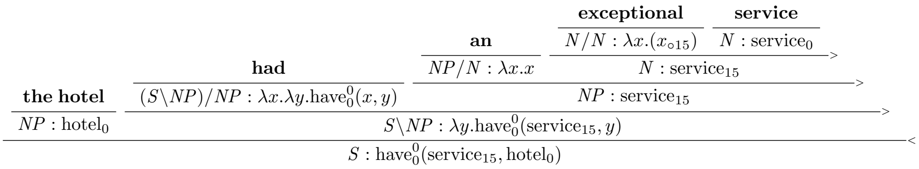

exceptional
service
an
$N/N : \lambda x.(x_{\circ 15})$
$N : \text{service}_0$
had
$NP/N : \lambda x.x$
$N : \text{service}_{15}$
the hotel
$(S\backslash NP)/NP : \lambda x.\lambda y.\text{have}_0^0(x, y)$
$NP : \text{service}_{15}$
$NP : \text{hotel}_0$
$S\backslash NP : \lambda y.\text{have}_0^0(\text{service}_{15}, y)$
$S : \text{have}_0^0(\text{service}_{15}, \text{hotel}_0)$
>
>
>
<

Figure 3.10: TODO

■

Example 3.4 …

Write example text for sentence with relative clause and "heavy" noun phrase shifting … The entity "breakfast" is modifyied by the adjective … yada yada … Notice that long distance relation between the entity and the adjective. Also explain why impact change *is needed to "close" the relative clause for further modification.*

■

End on a up-note :)

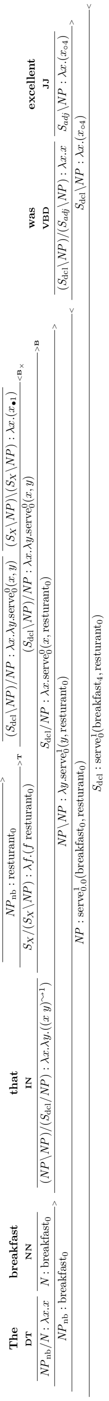

The
DT
breakfast
NN
that
IN
the
DT
resturant
NN
served
VBD
daily
RB
was
VBD
excellent
JJ
NP_nb/N : λx.x
N : breakfast_0
NP_nb : breakfast_0
(NP\NP)/(S_dcl/NP) : λx.λy.((x y)^¬1)
NP_nb/N : λx.x
N : resturant_0
NP_nb : resturant_0
S_X/(S_X\NP) : λf.(f resturant_0)
(S_dcl\NP)/NP : λx.λy.serve_0^0(x, y)
(S_X\NP)\(S_X\NP) : λx.(x_•1)
(S_dcl\NP)/NP : λx.λy.serve_0^0(x, y)
S_dcl/NP : λx.serve_0^0(x, resturant_0)
NP\NP : λy.serve_0^1(y, resturant_0)
NP : serve_{0,0}^1(breakfast_0, resturant_0)
(S_dcl\NP)/(S_adj\NP) : λx.x
S_adj\NP : λx.(x_∘4)
S_dcl\NP : λx.(x_∘4)
S_dcl : serve_0^1(breakfast_4, resturant_0)
>
>
>T
<B×
>B
>
<
>
<

Figure 3.11: Sentiment of sentence with "heavy" noun phrase shifting.

Chapter 4

Lexicon acquisition and annotation

The tiny languages captured by "handwritten" lexicons, such as the one demonstrated in the previous chapter, are obviously not a sane option when the grammar is to accept a large vocabulary and a wide range of sentence structures.

In order to use the presented model on a actual data, the acquisition of a wide covering lexicon is crucial. Initially considerable effort was made to *try* to build a CCG lexicon from a *POS-tagged corpus* (part-of-speech-tagged corpus). A POS-tagged corpus is simply a corpus where each token is tagged with a POS-tag, e.g. noun, verb, etc. There is no deep structure in such a corpus as opposed to a *treebank*. This approach turned up to have extensive issues as a result of this lack of structure, some of which are detailed in Appendix A which succinctly describes the process of the attempt.

There exists some wide covering CCG lexicons, most notable *CCGbank*, compiled by Hockenmaier and Steedman [2007] by techniques presented by [Hockenmaier, 2003]. It is essentially a translating of almost the entire Penn Treebank [Marcus *et al.*, 1993], which contains over 4.5 million tokens, and where each sentence structure has been analyzed in full and annotated. The result is a highly covering lexicon, with some entries having assigned over 100 different lexical categories. Clearly such lexicons only constitutes half of the previous defined $\mathcal{L}_{\text{CCG}}$ map, i.e. only the lexical categories, Γ. The problem of obtaining a full lexicon, that also yields semantics expressions, is addressed in the next section. It is also worth mentioning that since

Baldridge's [2002] work on modalities only slightly predates Hockenmaier's [2003] work on CCGBank, the CCGBank does not incorporate modalities[1]. However more unfortunately is that CCGBank is not free to use, mainly due to license restrictions on the Penn Treebank.

What might not be as obvious is that besides obtaining a wide-covering lexicon, $\mathcal{L}_{\text{CCG}}$, a even harder problem is for some text T to select the *right* tagging from $\mathcal{L}_{\text{CCG}}(w)$ for each token $w \in T$. Hockenmaier and Steedman [2007] calculate that the expected number of lexical categories per token is 19.2 for the CCGBank. This mean that a exhaustive search of even a short sentence (seven tokens) is expected to consider over 960 million ($19.2^7 \approx 961\,852\,772$) possible taggings. This is clearly not a feasible approach, even if the parsing can explore all possible deductions in polynomial time of the number of possible taggings. The number of lexical categories assigned to each token needs to be reduced, however simple reductions as just assigning the most frequent category observed in some training set (for instance CCGBank) for each token is not a solution. This would fail to accept a large amount of valid sentences, simply because it is missing the correct categories.

4.1 Maximum entropy tagging

Clearly a solution need to base the reduction on the setting of the token, i.e. in which *context* the token appears. Clark [2002] presents a machine learning approach based on a *maximum entropy model* that estimate the probability that a token is to be assigned a particular category, given the *features* of the local context, e.g. the POS-tag of the current and adjacent tokens, etc. This is used to select a subset of possible categories for a token, by selecting categories with a probability within a factor of the category with highest probability. Clark shows that the average number of lexical categories per token can be reduced to 3.8 while the parser still recognize 98.4% of unseen data. Clark and Curran [2007] presents a complete parser, which utilizes this tagging model, and a series of (log-linear) models to speed-up the actual deduction (i.e. the parsing) once the tagging model has assigned a set of categories to each token. What maybe even more interesting is that models trained on the full CCGBank, along with toolchain to use them (called the C&C tools), can be licensed freely for education or research purposes. For this reason it was chosen to use these models and tools.

Furthermore, even though the models neither incorporates modalities, since they are trained on the CCGBank, the deduction models solve many of these problems, since a more plausible deduction (i.e. a deduction seen more frequent in the CCGBank)

[1] There does exists another project, OpenCCG, started by Baldridge, which actually does incorporate modalities, but it has little documentation and was therefore not valued mature enough.

always will suppress other less plausible deductions. Special care are taken about coordination, so neither here seems the lack of modalities to yield significant issues.

4.2 Annotating the lexicon

The output from the C&C toolchain can be printed in various formats, including Prolog, which was considered the closest to the presented model, as it, given some set of tokens, $w_1, \ldots, w_n$, simply returns a lexicon and a deduction. An illustrative output for the tokens "the service was great" is given in Figure 4.1. In Chapter 5 more details on the actual format and the processing of it is given.

$$\begin{aligned}
&\alpha_1 \equiv \textbf{the} : \text{the}_{\text{DT}} \models NP_{\text{nb}}/N \\
&\alpha_2 \equiv \textbf{service} : \text{service}_{\text{NN}} \models N \\
&\alpha_3 \equiv \textbf{was} : \text{be}_{\text{VBD}} \models (S_{\text{dcl}}\backslash NP)/(S_{\text{adj}}\backslash NP) \\
&\alpha_4 \equiv \textbf{great} : \text{great}_{\text{JJ}} \models S_{\text{adj}}\backslash NP
\end{aligned}$$

(a) Lexicon

$$\dfrac{\dfrac{\alpha_1\ \alpha_2}{NP_{\text{nb}}}{>} \qquad \dfrac{\alpha_3\ \alpha_4}{S_{\text{dcl}}\backslash NP}{>}}{S_{\text{dcl}}}{<}$$

(b) Deduction

Figure 4.1: Illustration of output from the C&C toolchain.

Clearly, deductions in the style previously presented is trivially obtained by substituting the axioms placeholders with the lexicon entries associated. The C&C toolchain also has a build-in morphological analyzer which allow the lexicon to provide the *lemma* of the tokens and the POS-tag[2] of the token. Both of these will be proven convenient later.

There is however one essential component missing from the lexicon, namely the semantic expressions. However due to the *Principle of Categorial Type Transparency* it is known exactly *what* the types of the semantic expressions should be. There are currently a total of 429 different tags in the C&C tagging model, thus trying to handle each of these cases individually is almost as senseless choice as trying to manually construct the lexicon, and certainly not very robust for changes in the lexical categories. The solution is to handle some cases that need special treatment, and then use a generic annotation algorithm for all other cases. Both the generic and the special case algorithms will be a transformation $(\mathcal{T}, \Sigma^\star) \to \Lambda$, where the first argument is the type, $\tau \in \mathcal{T}$, to construct, and the second argument is the lemma, $\ell \in \Sigma^\star$, of the lexicon entry to annotate. Since the special case algorithms will fallback to the

[2] Since the C&C models are trained on CCGBank, which in turn are a translation of The Penn Treebank (PTB), the POS-tag-set used is equivalent to that of PTB cf. [Marcus *et al.*, 1993].

generic approach, in case preconditions for the case are not met, it is convenient to start with the generic algorithm, $\mathcal{U}_{\text{GEN}}$, which is given by Definition 4.1.

Definition 4.1 The *generic semantic annotation algorithm*, $\mathcal{U}_{\text{GEN}}$ (4.1), for a type τ and lemma ℓ is defined by the auxiliary function $\mathcal{U}'_{\text{GEN}}$, which takes two additional arguments, namely an infinite set of variables $\mathcal{V}$ cf. Definition 3.2, and an ordered set of sub-expressions, (denoted A), which initially is empty.

$$\mathcal{U}_{\text{GEN}}(\tau, \ell) = \mathcal{U}'_{\text{GEN}}(\tau, \ell, \mathcal{V}, \emptyset) \tag{4.1}$$

If τ is primitive, i.e. $\tau \in \mathcal{T}_{\text{prim}}$, then the generic algorithm simply return a functor with name ℓ, polarity and impact argument both set to 0, and the ordered set A as arguments. Otherwise there must exists unique values for $\tau_\alpha, \tau_\beta \in \mathcal{T}$, such that $\tau_\alpha \to \tau_\beta = \tau$, and in this case the algorithm return an abstraction of τ_α on variable $v \in V$, and recursively generates a expression for τ_β.

$$\mathcal{U}'_{\text{GEN}}(\tau, \ell, V, A) = \begin{cases} \ell_0^0(A) : \tau & \text{if } \tau \in \mathcal{T}_{\text{prim}} \\ \lambda v.\mathcal{U}'_{\text{GEN}}(\tau_\beta, \ell, V \setminus \{v\}, A') : \tau & \text{otherwise, where:} \end{cases}$$

$$\begin{aligned} v &\in V \\ \tau_\alpha \to \tau_\beta &= \tau \\ A' &= \begin{cases} A[e : \tau_\alpha \to \tau_\gamma \mapsto ev : \tau_\gamma] & \text{if } e' : \tau_\alpha \to \tau_\gamma \in A \\ A[e : \tau_\gamma \mapsto ve : \tau_\delta] & \text{if } \tau_\gamma \to \tau_\delta = \tau_\alpha \wedge e' : \tau_\gamma \in A \\ A \cup \{v : \tau\}) & \text{otherwise} \end{cases} \end{aligned}$$

The recursive call also removes the abstracted variable v from the set of variables, thus avoiding recursive abstractions to use it. The ordered set of sub-expressions, A, is modified cf. A', where the notation $A[e_1 : \tau_1 \mapsto e_2 : \tau_2]$ is the substitution of all elements in A of type τ_1 with $e_2 : \tau_2$. Note that e_1 and τ_1 might be used to determine the new value and type of the substituted elements. Since the two conditions on A' are not mutual exclusive, if both apply the the first case will be selected. The value of A' can be explained in a informal, but possibly easier to understand, manner:

- If there is a least one function in A, that takes an argument of type τ_α, then apply v (which is known to by of type τ_α) to all such functions in A.
- If the type of v itself is a function (i.e. $\tau_\gamma \to \tau_\delta = \tau_\alpha$), and A contains at least one element that can be used as argument, then substitute all such arguments in A by applying them to v.
- Otherwise, simply append v to A.

■

The get a little familiar with how the generic semantic annotation algorithm works, Example 4.1 shows the computation of some types and lemmas.

EXAMPLE 4.1 *Table 4.1 shows the result of applying $\mathcal{U}_{\text{GEN}}$ on some lemmas and types. The result for a noun as "room" is simply the zero-argument functor of the same name. The transitive verb "provide" captures two noun phrases, and yields a functor with them as arguments.*

More interesting is the type for the determiner "every", when used for instance to modify a performance verb, as shown in Figure 4.2. It starts by capturing a noun, then a function over noun phrases, and lastly a noun phrase. The semantic expression generated for this type is a functor, with simply the noun as first argument, and where the second argument is the captured function applied on the noun phrase.

Lemma	Type	Generic semantic expression
room	τ_{N}	room_0^0
provide	$\tau_{\text{NP}} \to \tau_{\text{NP}} \to \tau_{\text{S}}$	$\lambda x.\lambda y.\text{provide}_0^0(x, y)$
every	$\tau_{\text{N}} \to (\tau_{\text{NP}} \to \tau_{\text{S}}) \to (\tau_{\text{NP}} \to \tau_{\text{S}})$	$\lambda x.\lambda y.\lambda z.\text{every}_0^0(x, y\ z)$

Table 4.1: Some input/output of generic annotation algorithm

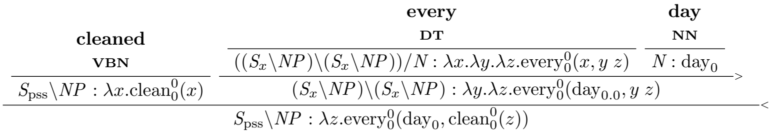

every
day
cleaned
DT
NN
VBN
$((S_x\backslash NP)\backslash(S_x\backslash NP))/N : \lambda x.\lambda y.\lambda z.\text{every}_0^0(x, y\ z)$
$N : \text{day}_0$
>
$S_{\text{pss}}\backslash NP : \lambda x.\text{clean}_0^0(x)$
$(S_x\backslash NP)\backslash(S_x\backslash NP) : \lambda y.\lambda z.\text{every}_0^0(\text{day}_{0.0}, y\ z)$
<
$S_{\text{pss}}\backslash NP : \lambda z.\text{every}_0^0(\text{day}_0, \text{clean}_0^0(z))$

Figure 4.2: Complex determiner modifying performance verb.

■

Clearly the generic algorithm does not provide much use with respect to extracting the sentiment of the text, i.e. it only provide some logic structures that are guaranteed to have the correct type. The more interesting annotation is actually handled by the special case algorithms. How this is done is determined by a combination of the POS-tag and the category of the entry. Most of these treatments are very simple, with the handling of adjectives and adverbs being the most interesting. The following briefly goes through each of the special case annotations.

- *Determiners* with simple category, i.e. NP/N, are simply mapped to the identity function, $\lambda x.x$. While determiners have high focus in other NLP tasks, such as determine if a sentence is valid, the importance does not seem significant in sentiment analysis, e.g. whether an opinion is stated about *some* or *all* of entities does not change the overall polarity of the opinion.

- *Nouns* are in general just handled by the generic algorithm, however in some cases of multi-word nouns, the sub-lexical entities may be tagged with the category N/N. In these cases the partial noun is annotated with a list structure, that eventually will capture the entire noun, i.e. $\lambda x.\langle \mathcal{U}_{\text{GEN}}(\tau_{\text{N}}, \ell), x\rangle$, where ℓ is the lemma of the entity to annotate.

- *Verbs* are just as nouns in general handled by the generic algorithm, however *linking verbs* are a special case, since they relate the subject (i.e. a entity) with one or more *predicative adjectives*. Linking verbs have the category $(S_{\text{dcl}}\backslash NP)/(S_{\text{adj}}\backslash NP)$, and since the linked adjectives directly describes the subject of the phrase such verbs are simply annotated with the identity function, $\lambda x.x$.

- *Adjectives* can have a series of different categories depending on how they participate in the sentence, however most of them have the type $\tau_\alpha \rightarrow \tau_\beta$, where $\tau_\alpha, \tau_\beta \in \mathcal{T}_{\text{prim}}$. These are annotated with the *change* of the argument, i.e. $\lambda x.x_{\circ j}$, where j is a value determined based on the lemma of the adjective. Notice that this assumes implicit type conversion of the parameter from τ_α to τ_β, however since these are both primitive, this is a sane type cast. Details on how the value j is calculated are given in Section 4.3.

- *Adverbs* are annotated in a fashion closely related to that of adjectives. However the result might either by a *change* or a *scale*, a choice determined by the lemma: normally adverbs are annotated by the change in the same manner as adjectives, however *intensifiers* and *qualifiers*, i.e. adverbs that respectively strengthened or weakened the meaning, are scaled. Section 4.3 gives further details on how this choice is made. Finally special care are taken about negating adverbs, i.e. "not", which are scaled with a value $j = -1$.

- *Prepositions* and *relative pronouns* need to change the impact argument as they capture partial sentences, i.e. *preposition phrases* and *relative clauses*, and further modification should bind to the subject entire phrase as were illustrated by Example 3.4.

Finish ...

4.3 Calculating sentiment polarities

In order to reason about the polarity of the entities present in the review text, a understanding of the domain of the review is needed. For this purpose the concept of *semantic networks* is introduced. Formally a semantic network is a structure cf. Definition 4.2.

Definition 4.2 a semantic network is a quadruple (L, S, R, M) where:

- L is the set of lexical units recognized by the network.
- S is the set of *semantic concepts* in the network.
- R is a set of binary relations on S where the relation $r \in R$ describes links between semantic concepts, i.e. $r \subset S \times S$.
- M is a mapping from lexical units to a set of semantic entities that the lexical unit can entail, i.e. $M : L \to \mathcal{P}(S)$.

■

Notice that S and R constitutes a set of graphs, i.e. for each relation $r \in R$ the graph (S, r). The graph is undirected if r is *symmetric*, and directed if r is *asymmetric*. An illustrative example of such a graph, denoting a relation in a tiny semantic network of adjective concepts is given in Figure 4.3.

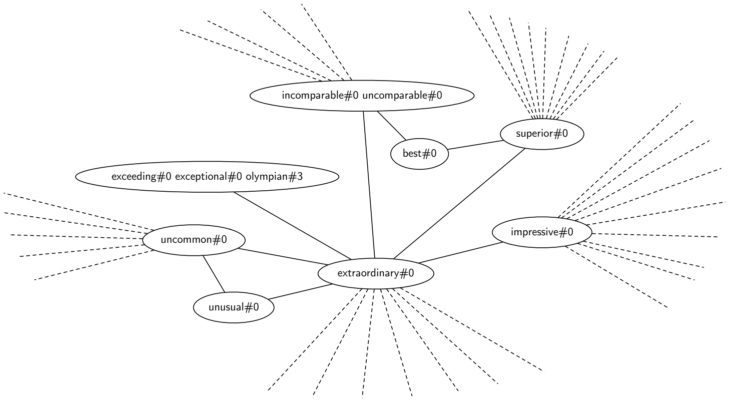

incomparable#0 uncomparable#0
best#0
superior#0
exceeding#0 exceptional#0 olympian#3
impressive#0
uncommon#0
extraordinary#0
unusual#0

Figure 4.3: Illustration of relation in a semantic network.

The concrete semantic network used is WordNet, originally presented by Miller [1995], and later presented in depth by Fellbaum [1998]. Like it was the issue when acquiring a lexicon for the syntactic analysis, the availability of free semantic networks is very sparse. It has not been possible to find other competing networks with a coverage close to that of WordNet, and thus the decision of using WordNet is really based on it being the only choice. With that said, it is argued that WordNet is a quite extensive semantic network, which in its most recent revision (3.0) contains a relatively large number of semantic concepts, namely $|S| \approx 117\,000$, while the number of lexical units recognized may be argued is not as covering as ideally, namely $|L| \approx 155\,000$.

WordNet contains a variety of relations, R, however for the purpose of calculating sentiment polarity values, only the following were considered interesting:

- The *similar*-relation links *closely similar* semantic concepts, i.e. concepts having almost the synonymies mensing in most contexts. The relation is present for most concepts entailed by adjectives.
- The *see also*-relation links *coarsely similar* semantic concepts, i.e. concepts having a clear different mensing, but may be interpreted interchangeably for some contexts. Figure 4.3 on the preceding page shows an example of exactly the *see also*-relation.

Finish ... entities are looked up using L,

Chapter 5

Implementation

In order to demonstrate the logical approach, introduced in the previous chapters, a *proof of concept* system was implemented. In the following sections key aspects of the implementation of this system will be presented. A complete walk-though will not be presented, but the complete source code for the implementation is available in Appendix **??**. Also notice that code segments presented in this chapter maybe simplified from the source code to ease understanding. For instance the C&C-toolchain uses some additional primitive categories to handle conjunctions, commas and punctuations that are not consider theoretical or implementationwise interesting, as they are translatable to the set of categories already presented. In the actual implementation of the proof of concept system this is exactly what is done, once the output from the C&C-toolchain has been parsed.

It was chosen to use the purely functional programming language *Haskell* for implementing the proof of concept system. The reason Haskell, specifically the *Glasgow Haskell Compiler*, was chosen as programming language and platform, was i.a. its ability to elegantly and effectively implement a parser for the output of the C&C-toolchain. Data structures are like in many other functional languages also possible to state in a very succinct and neat manner, which allow Haskell to model the extended semantics presented in Section 3.4, as well as any other structure presented, e.g. deduction proofs, lexical and phrasal categories, etc.

5.1 Data structures

Data structures are stated in Haskell by the means of *type constructors* and *data constructors*. To model for instance lexical and phrasal categories the two infix operators, `:/` and `:\` are declared (using `/` and `\` was not considered wise, as `/` is already used for devision by the Haskell Prelude) as shown in Figure 5.1. The *agreement* of an primitive category is simply a set of features cf. Section 3.3, which is easiest modeled using the list structure. As features are just values from some language specific finite set they are simply modeled by *nullary data constructors*. One might argue that features have *different* types, e.g. person, number, gender, etc. However it is convenient to simply regard all features as being of the *same* type, a model borrow from van Eijck and Unger [2010, chap. 9].

```haskell
infix 9 :/   -- Forward slash operator
infix 9 :\   -- Backward slash operator

type Agreement = [Feature]

data Category = S Agreement               -- Sentence
              | N Agreement               -- Noun
              | NP Agreement              -- Noun Phrase
              | PP Agreement              -- Preposision Phrase
              | Category :/ Category      -- Forward slash
              | Category :\ Category      -- Backward slash

data Feature = SDcl | SAdj | SNb | SNg | ...
```

Figure 5.1: Example of declaring the data structure for categories.

The code shown in Figure 5.1 is really all what is needed to represent the syntactic structure of categories. Another illustration of one of the data structural advantages of using a functional programming language is shown in Figure 5.2. Notice how the declaration of the syntax for the semantic expressions is completely analog to the formal syntax given in Definition 3.2 and 3.3, with the exception that the implemented syntax is untyped. The reason why types are omitted from the implemented model of semantic expressions is simply that they are always accompanied by a category, and thus the type of the expression is trivially obtainable when needed.

```haskell
data SExpr = Var String                     -- Variable
           | Abs String SExpr               -- Lambda abstraction
           | App SExpr SExpr                -- Lambda application
           | Fun String Float Int [SExpr]   -- Functor
           | Seq [SExpr]                    -- Sequence
           | ImpactChange SExpr Int         -- Impact change
           | Change SExpr Float             -- Change
           | Scale SExpr Float              -- Scale
```

Figure 5.2: Example of declaring the data structure for semantic expressions.

5.2 Reducing semantic expressions

With data structures available for representing the syntax of the semantic expressions it is time to focus on reducing the expression using the semantic rules presented in Definition 3.3. This can be easily done in a functional language by specifying a *reduction function*, i.e. a function that recursively rewrites semantic expressions based on the rules presented in the definition. By using the *pattern matching* available in Haskell, each rule can be implemented in a one-to-one manner by a function declaration that only accepts the *pattern* of that rule. For instance Figure 5.3 shows the implementation of the (FC1), (SC) and (PC) rules. A small set of additional function declarations are needed to allow reduction inside a structure that itself cannot be reduced, and finally the *identity function* matches any pattern not captured by any of the other function declarations. Notice that η-reduction was not implemented, since this rule is merely a performance enhancing rule.

```haskell
-- (FC1)
reduce (Change (Fun f j 0 ts) j') =
  Fun f (j + j') 0 $ map reduce ts

-- (SC)
reduce (Change (Seq ts) j') =
  Seq $ map (reduce . flip Change j') ts

-- (PC)
reduce (Change (Abs x t) j') =
  Abs x $ reduce $ Change t j'
```

Figure 5.3: Example of declaring the rules for semantic expressions.

5.3 Parsing output from the C&C tools

The implemented parser for the Prolog style output yielded by the C&C-toolchain, presented briefly in Section 4.2, uses the Parsec library for Haskell by Leijen [2001]. Parsec is a strong monadic parser combinator, that among other things allows fast and efficient parsing of LL[1] grammars, and can thus easily capture the subset of the Prolog language used by the C&C-toolchain. Parsec differs significantly from common Yacc approaches, since it describes the grammar *directly* in Haskell, without the need of some intermediate language or processing tools.

Figure 5.4 shows the actual raw output from the C&C-toolchain that is the basis the illustration in Figure 4.1 shown back in Section 4.2. The first section of the output represents the deduction tree, while the second represents the lexicon (obviously without semantic expressions).

```
ccg(1,
 ba('S[dcl]',
  fa('NP[nb]',
   lf(1,1,'NP[nb]/N'),
   lf(1,2,'N')),
  fa('S[dcl]\NP',
   lf(1,3,'(S[dcl]\NP)/(S[adj]\NP)'),
   lf(1,4,'S[adj]\NP')))).

w(1, 1, 'the', 'the', 'DT', 'I-NP', 'O', 'NP[nb]/N').
w(1, 2, 'service', 'service', 'NN', 'I-NP', 'O', 'N').
w(1, 3, 'was', 'be', 'VBD', 'I-VP', 'O', '(S[dcl]\NP)/(S[adj]\NP)').
w(1, 4, 'great', 'great', 'JJ', 'I-ADJP', 'O', 'S[adj]\NP').
```

Figure 5.4: Raw output from the C&C toolchain.

One of the most admirable features of Parsec is its *parser combinator library*, containing a verity bundled auxiliary functions, which allows the declaration of advanced parsers by combining smaller parsing functions. To parse for instance the categories present in both of the sections one can build an *expression parser* simply by stating the *symbol*, *precedence* and *associativity* of the operators.

Figure 5.5 shows the parser for categories. The precedence of the operators are given by the outer list in the *operator table*, while operators within the same inner list have the same precedence, which is in the case for both of the categorial infix operators. Finally a category is declared as either compound (i.e. a category expression), or as one of the four primitive categories. Notice how the parser needs to first *try* to parse *noun phrases* (NP), and then *nouns* (N), since the parser otherwise could successfully parse a noun, and then meet an unexpected “P”, which would cause a parser error.

```
pCategoryExpr :: Parser Category
pCategoryExpr = buildExpressionParser pCategoryOpTable pCategory

pCategoryOpTable :: OperatorTable Char st Category
pCategoryOpTable = [ [ op "/"  (:/) AssocLeft ,
                       op "\\" (:\) AssocLeft ] ]
                   where
                     op s f a = Infix ( string s >> return f ) a

pCategory :: Parser Category
pCategory =          pParens pCategoryExpr
           <|>       (pCategory' "S"    S)
           <|> try   (pCategory' "NP"   NP)
           <|>       (pCategory' "N"    N)
           <|>       (pCategory' "PP"   PP)
           <?> "category"
```

Figure 5.5: Example of parsing categorial expression.

The parsing of the lexicon is considered trivial, since its structure is flat with the exception of the category. ...

Finish

5.4 WordNet interface and semantic networks

To lookup semantic concepts and relations in the WordNet data files an open source interface library by Daumé III [2008] was used as base. However the interface was not complete, and missed critical features. For instance the library could only calculate the closure of two semantic concepts, which of cause only is possible when the relation forms a partial order, e.g. as is the case with the *hyponym/hypernym* relation and the *holonym/meronym* relation. Therefore the library has undergone significant rewrite and cleanup in order to use it for the presented purpose.

To model semantic networks another open source library was used, namely the *Functional Graph Library* (FGL). The library implements efficient functional graph representation and algorithms presented by Erwig [2001]. However transforming the relational representation of WordNet into an actual graph in the sense of FGL is somewhat tricky. The reason for this is that intended usage of the WordNet data files do not exposes S, and neither r in the form of a subset of $S \times S$, which makes good sense since this representation does not scale well with $|S|$. Instead it is intended to query using the lookup function, M, which is indexed and allows logarithmic time lookup of lexical units; likewise a relation, $\hat{r}$, is a function from one semantic concept to a set of related concepts, i.e. $\hat{r} : S \to \mathcal{P}(S)$. This structure makes querying WordNet efficient, but also allows some optimization with respect to calculating the sentiment polarity value of lexical units. Recall from Section 4.3 that the approach is to select a set of respectively positive and negative seed concepts, and then measure the difference of the sum of distances from a lexical unit to these. However instead of regard the entire graph (S, r) only a subgraph (S', r') is considered, namely the subgraph that constitutes the *connected component* that contains all semantic concepts that are reachable from the seed concepts using the relation function $\hat{r}$. This of cause assumes that r is symmetric, which is also the case for the relations considered cf. Section 4.3.

The construction of (S', r') for some set of positive and negative semantic concepts, respectively P_{seed} however it also makes the task of building a graph from negative bui i.e. going from a set of semantic concepts S and a relation r to the graph (S, r). The problem is that it really do not make ...

Finish

Chapter 6

Evaluation

The data set used for evaluating the presented logical approach for sentiment analysis, specifically the *proof of concept* system is the *Opinosis Dataset*, originally used by [Ganesan *et al.*, 2010]. The data set consists of texts from actual user reviews on a total of 51 different topics. The topics are ranging over different objects, from consumer electronics (e.g. GPS navigation, music players, etc.) to hotels, restaurants and cars. For most of the objects, reviews are covered by multiple topics. For instance a specific car is covered by the topics *comfort*, *interior*, *mileage*, *performance*, and *seats*.

It has been hard to find any real alternatives for the *Opinosis Dataset* for several reasons: Most collected reviews are commercial, and thus not free to use; furthermore the *Opinosis Dataset* also contains summerized texts for each of its topics, which are constructed by manual, human interpretation. The latter allow a straight approach for comparison of any results the proposed system will yield.

The hotel buffet had fabulous food. (6.1)

Very friendly servers and nice selection of food at a reasonable price. (6.2)

Room service was extortionate though, very very expensive,
so we didnt bother, as food outlets a few minutes walk away. (6.3)

The texts (6.1) to (6.3) show actual extracts from the data set for a topic on food quality on the Swissôtel Restaurant. While (6.1) is a valid declarative sentence, (6.2) is not, since it lacks a subject (i.e. the restaurant). A coarse review of the text in the dataset reveals that missing subjects are a repeating issue. This might not seem that odd, since many people would implicitly imply the subject from the topic that they are reviewing. Thus text missing subjects can in many cases still be considered as valid sentences with minimal effort. The text (6.3) is on the other hand missing a transitive verb (presumably *are*) from the subordinate clause. In cases where such savere gramatically errors occurs it is sugested to ignore the clause, and try only to analyse the main clause. Furthermore the text (6.3) use repeated adverbs (e.g. *very very*) to express intensification, however it should not be any major concern that a verb or adjective are modified multiple times by the *same* adverb, but the intended intensification will probably not be included in the semantic analysis. Thus formalizing such a grammer is mostly a tak od designing such lexicon.

As evendent from these examples far from all texts in the dataset are valid sentences.

Chapter 7

Discussion

Since adjectives and adverbs are always reduced to only contribute to the polarity, they cannot be used to identify subjects. E.g. what do you think about the white iPhone vs. the black? (need better ex.!)

7.1 Fureture work

We expect such an algorithm to calculate a match score, that is a weighted average over several metrics. Given below are methods for calculating scores for some evident metrics.

- Symbolic similarity – at its most basic form we can consider a sample string (i.e. a word from an input text) against the system's vocabulary using approximate string matching algorithms such as the *Levenshtein distance* as described by [Wagner and Fischer, 1974].
- Pronunciation similarity – it is an valid assumption that many misspellings still share a majority of the pronunciation with the intended word, i.e. they are approximately homophone. Thus comparing the phonetic properties of an sample string with possible matches can in cases correct misspellings. The

> *Soundex algorithm* by Robert C. Russell and Margaret K. Odell, as described by [Knuth, 1998, p. 391–92], is a simple, yet power full approach for this purpose.

—

Histogram of test/train distribution - plot word length - semi-supervised self-training with re-ranking ?

—

“lift” restrictions about the texts, e.g. no of sentences ... context-sensitive, e.g. resolution of relative pronpoun ... The room was luxurious, it had ...

Chapter 8

Conclusion

Appendix A

A naive attempt for lexicon acquisition

This appendix describes the efforts that was initially made in order to acquire a CCG lexicon by *transforming* a tagged corpus, namely the *Brown Corpus.*

Since English is

> *As with English around the world, the English language as used in the United Kingdom and the Republic of Ireland is governed by convention rather than formal code: there is no equivalent body to the Académie française or the Real Academia Española, and the authoritative dictionaries (for example, Oxford English Dictionary, Longman Dictionary of Contemporary English, Chambers Dictionary, Collins Dictionary) record usage rather than prescribe it. In addition, vocabulary and usage change with time; words are freely borrowed from other languages and other strains of English, and neologisms are frequent.*
>
> *http://en.wikipedia.org/wiki/British_English#Standardisation*

The Brown Corpus was compiled by Francis and Kucera [1979] by collecting written works printed in United States during the year 1961. The corpus consists of just over one million words taken from 500 American English sample texts, with the intension of covering a highly representative variety of writing styles and sentence structures.

Notable drawbacks of the Brown Corpus include its age, i.e. there are evidently review topics where essential and recurring words used in present day writing was not coined yet or rarely used back 50 years ago. For instance does the Brown Corpus not recognize the words *internet*, *hotspot*

sentences will containing words has found it's way into comon that

Other corpora has been considered [van Eijck and Unger, 2010]

A.1 Tokenizer and tagger

The tokenizer has a very simple task, namely to convert an input string to a list of tokens (lower case words) that represent the symbols of the language. An example of the transformation is shown in (A.1).

$$\text{“Put the pyramid onto the table.”} \rightarrow [\textbf{put}, \textbf{the}, \textbf{pyramid}, \textbf{onto}, \textbf{the}, \textbf{table}] \quad (A.1)$$

A.1.1 Shift-reduce parser

A.2 Find a good title

The initial attempt is simply to construct a parser that

> There are three basic ways to build a shift-reduce parser. Full LR(1) (the ‘L’ is the direction in which the input is scanned, the ‘R’ is the way in which the parse is built, and the ‘1’ is the number of tokens of lookahead) generates a parser with many states, and is therefore large and slow. SLR(1) (simple LR(1)) is a cut-down version of LR(1) which generates parsers with roughly one-tenth as many states, but lacks the power to parse many grammars (it finds conflicts in grammars which have none under LR(1)).
>
> LALR(1) (look-ahead LR(1)), the method used by Happy and yacc, is tradeoff between the two. An LALR(1) parser has the same number of

> states as an SLR(1) parser, but it uses a more complex method to calculate the lookahead tokens that are valid at each point, and resolves many of the conflicts that SLR(1) finds. However, there may still be conflicts in an LALR(1) parser that wouldn't be there with full LR(1).

The state S_τ ...

Formally a rule $\mathcal{R}_\tau$, for the state type τ, is a transformation from a state $s \in \mathcal{S}_\tau$ onto a new set of states $\mathcal{S}'_\tau \subset \mathcal{S}_\tau$ cf. A.2.

$$\mathcal{R}_\tau : \mathcal{S}_\tau \to \mathcal{P}(\mathcal{S}_\tau) \tag{A.2}$$

The state type for analysing CCGs is a 2-tuple, where P is a totally ordered set of ...,

$$\mathcal{S}_{\mathrm{CCG}} : \mathcal{P}(T) \times \mathcal{P}(\mathcal{P}(T))$$

$$\mathcal{R}^{\mathrm{shift}}_{\mathrm{CCG}} \tag{A.3}$$

If all rules in the set is monotone, then the parsing will terminate

Bibliography

[Baccianella *et al.*, 2010] Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*. European Language Resources Association (ELRA), 2010.

[Baldridge and Kruijff, 2003] Jason Baldridge and Geert-Jan M. Kruijff. Multi-Modal Combinatory Categorial Grammar. In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics - Volume 1*, EACL '03, pages 211–218, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[Baldridge, 2002] Jason Baldridge. *Lexically Specified Derivational Control in Combinatory Categorial Grammar.* PhD thesis, University of Edinburgh, 2002.

[Bresnan *et al.*, 1982] J. Bresnan, R. M. Kaplan, S. Peters, and A. Zaenen. Cross-Serial Dependencies in Dutch. *Linguistic Inquiry*, 13(fall):613–635+, 1982.

[Clark and Curran, 2007] Stephen Clark and James R. Curran. Wide-Coverage Efficient Statistical Parsing with CCG and Log-linear Models. *Computational Linguistics*, 33(4):493–552, 12 2007.

[Clark, 2002] Stephen Clark. A Supertagger for Combinatory Categorial Grammar. In *Proceedings of the 6th International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+6)*, pages 19–24, Venice, Italy, 2002.

[Daumé III, 2008] Hal Daumé III. *HWordNet - A Haskell Interface to WordNet.* Url: `http://www.umiacs.umd.edu/~hal/`, 2008.

[Erwig, 2001] Martin Erwig. Inductive Graphs and Functional Graph Algorithms. *Journal of Functional Programming*, 11(5):467–492, 2001.

[Esuli and Sebastiani, 2006] Andrea Esuli and Fabrizio Sebastiani. SentiWordNet: A Publicly Available Lexical Resource for Opinion Mining. In *Proceedings of the 5th Conference on Language Resources and Evaluation (LREC'06)*, pages 417–422. European Language Resources Association (ELRA), 2006.

[Eurostat, 2010] Eurostat. *Population on 1 January by Age and Sex.* Url: `http://ec.europa.eu/eurostat`, 2010.

[Fellbaum, 1998] Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database (Language, Speech, and Communication).* The MIT Press, illustrated edition edition, 1998.

[Francis and Kucera, 1979] W. Nelson Francis and Henry Kucera. Brown Corpus Manual. Technical report, Department of Linguistics, Brown University, Providence, Rhode Island, US, 1979.

[Ganesan *et al.*, 2010] Kavita Ganesan, Cheng Xiang Zhai, and Jiawei Han. Opinosis: A graph based approach to abstractive summarization of highly redundant opinions. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING '10)*, 2010.

[Hockenmaier and Steedman, 2007] Julia Hockenmaier and Mark Steedman. CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396, 2007.

[Hockenmaier *et al.*, 2004] Julia Hockenmaier, Gann Bierner, and Jason Baldridge. Extending the Coverage of a CCG System. *Journal of Language and Computation*, 2:165–208, 2004.

[Hockenmaier, 2003] Julia Hockenmaier. *Data and Models for Statistical Parsing with Combinatory Categorial Grammar.* PhD thesis, University of Edinburgh, 2003.

[Joshi *et al.*, 1975] Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. Tree Adjunct Grammars. *Journal of Computer and System Sciences*, 10(1):136–163, 1975.

[Joshi *et al.*, 1990] Aravind K. Joshi, K. Vijay Shanker, and David Weir. *The Convergence Of Mildly Context-Sensitive Grammar Formalisms*, 1990.

[Knuth, 1998] Donald E. Knuth. *The Art of Computer Programming, vol. 3: Sorting and Searching.* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2nd edition, 1998.

[Leijen, 2001] Daan Leijen. *Parsec, a fast combinator parser.* University of Utrecht, Department of Computer Science, PO.Box 80.089, 3508 TB Utrecht, The Netherlands, 2001.

[Likert, 1932] Rensis Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):1–55, 1932.

[Liu, 2007] Bing Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data.* Springer, 2007.

[Marcus *et al.*, 1993] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[Miller, 1995] George A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[Padró *et al.*, 2010] Lluís Padró, Samuel Reese, Eneko Agirre, and Aitor Soroa. Semantic Services in FreeLing 2.1: WordNet and UKB. In Pushpak Bhattacharyya, Christiane Fellbaum, and Piek Vossen, editors, *Principles, Construction, and Application of Multilingual Wordnets*, pages 99–105, Mumbai, India, February 2010. Global Wordnet Conference 2010, Narosa Publishing House.

[Pang and Lee, 2008] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2(1-2):1–135, 2008.

[Paul and Baker, 1992] Douglas B. Paul and Janet M. Baker. The design for the Wall Street Journal-based CSR corpus. In *Proceedings of the workshop on Speech and Natural Language*, pages 357–362, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

[Pingdom, 2010] Pingdom. *Study: Ages of Social Network Users.* Url: `http://pingdom.com/`, 2010.

[Pollard, 1984] Carl Pollard. *Generalized Context-Free Grammars, Head Grammars and Natural Language.* PhD thesis, Stanford University, 1984.

[Russell and Norvig, 2009] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 3rd edition, 2009.

[Shieber, 1985] Stuart M. Shieber. Evidence Against the Context-Freeness of Natural Language. *Linguistics and Philosophy*, 8(3):333–343, 1985.

[Simančík and Lee, 2009] František Simančík and Mark Lee. A CCG-based system for valence shifting for sentiment analysis. *Research in Computing Science*, 41:99–108, 2009.

[Steedman, 1998] Mark Steedman. *Categorial Grammar*, 1998.

[Steedman, 2000] Mark Steedman. *The Syntactic Process.* The MIT Press, 2000.

[Steedman, 2011] Mark Steedman. *Taking Scope: The Natural Semantics of Quantifiers.* The MIT Press, 2011.

[Tan *et al.*, 2011] Luke Kien-Weng Tan, Jin-Cheon Na, Yin-Leng Theng, and Kuiyu Chang. Sentence-level sentiment polarity classification using a linguistic approach. In *Proceedings of the 13th international conference on Asia-pacific digital libraries: for cultural heritage, knowledge dissemination, and future creation*, ICADL'11, pages 77–87, Berlin, Heidelberg, 2011. Springer-Verlag.

[van Eijck and Unger, 2010] Jan van Eijck and Christina Unger. *Computational Semantics with Functional Programming.* Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[Vijay-Shanker and Weir, 1994] K. Vijay-Shanker and David J. Weir. The Equivalence Of Four Extensions Of Context-Free Grammars. *Mathematical Systems Theory*, 27:27–511, 1994.

[Wagner and Fischer, 1974] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, January 1974.

[Webster and Kit, 1992] Jonathan J. Webster and Chunyu Kit. Tokenization as the initial phase in nlp. In *Proceedings of the 14th conference on Computational linguistics - Volume 4*, COLING '92, pages 1106–1110, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.