Machine Translation
A Transfer Approach

Grace Noone
B.A. (Mod) CSLL
Final Year Project, May 2003
Supervisor: Dr Martin Emms
Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university.

__________________________ May 1, 2003
Grace Noone
Abstract

This paper presents a bidirectional translator, which is based on the transfer approach and implemented in Prolog. The system is based on the sub-domain of weather reports. The methodologies used in this implementation enable accurate translations.
Acknowledgements

Firstly, I would especially like to thank Martin, my supervisor, for all his advise, enthusiasm and constant guidance throughout the entire year.

Secondly, I would like to Carl for all his valuable help and encouragement.

I would also like to express a heartfelt thanks to my Mam, Dad and family for their never-ending encouragement and support.

Last but by no means least I would like to thank Gar and my friends for making my life so enjoyable.
‘Education is a progressive discovery of your own ignorance’
Will Durant
# List of Figures

2.1 Direct Approach .................................................. 14  
2.2 Interlingua Approach ............................................. 15  
2.3 Possible Translation Directions ................................. 16  
2.4 Transfer Approach .................................................. 17  
2.5 Basic German Tree .................................................. 18  
3.1 Forming Plurals .................................................... 22  
3.2 Declination of Definite Articles ................................. 24  
3.3 Declination of Indefinite Articles ............................... 25  
3.4 Adjective Endings following Definite Article ................ 26  
3.5 Adjective Endings following Indefinite Article ................ 26  
3.6 Adjective Endings following no Article ....................... 26  
3.7 Present Form of sein to be ....................................... 28  
3.8 TreeTagger Output .................................................. 30  
4.1 Sample Chart ....................................................... 44  
5.1 Architecture of Entire System .................................. 46  
5.2 Translation Procedure from English to German ............. 47  
5.3 Translation Procedure from German to English .............. 49  
5.4 Key to Météo rules .................................................. 59  
5.5 Météo rule - MET1 .................................................. 60  
5.6 Météo rule - MET2 .................................................. 60  
5.7 Météo rule - MET3 .................................................. 60  
10.1 Pre-edited German Input ....................................... 103  
10.2 Translators Results .............................................. 104  
10.3 Translators Results .............................................. 105  
10.4 English Input to Translator .................................... 108  
10.5 English Translation Results ................................... 109  
10.6 English Translation Results ................................... 110  
11.1 Sample Weather Report supplied by Met Éireann ........... 115  
11.2 Sample German Weather Report - Süddeutsche Zeitung March 25th  . . . 116
# Contents

1 Introduction .......................................................................................... 10  
  1.1 Motivation ....................................................................................... 11  
  1.2 Aims ................................................................................................. 11  
  1.3 Overview .......................................................................................... 12  

2 The Art of Machine Translation .............................................................. 13  
  2.1 Introduction ..................................................................................... 13  
  2.2 Translation Approaches .................................................................. 13  
    2.2.1 The Direct Approach ............................................................... 13  
    2.2.2 The Interlingua Approach ....................................................... 15  
    2.2.3 The Transfer Approach ........................................................... 16  
    2.2.4 Online Interactive Systems .................................................... 18  
    2.2.5 Learning Based Translation ................................................... 19  
  2.3 Conclusion ....................................................................................... 20  

3 The German - English Domain ............................................................... 21  
  3.1 Introduction ..................................................................................... 21  
  3.2 Basic Differences between German and English ......................... 21  
  3.3 German Morphological Issues ....................................................... 24  
    3.3.1 Definite Articles .................................................................... 24  
    3.3.2 Indefinite Articles .................................................................. 25  
    3.3.3 German Adjective Inflection ................................................ 25  
  3.4 Morphology ..................................................................................... 27  
    3.4.1 Alternative Taggers & a Morphology System ....................... 29  
  3.5 The Sub-domain ............................................................................. 31  
  3.6 Characteristics of German & English Weather Data .................... 32  
  3.7 Assumptions made by this Application ......................................... 33  
  3.8 Linguistic Problems during Transfer ............................................. 35  
    3.8.1 Tricky Translations ................................................................. 35  
    3.8.2 Weakly translated Words ...................................................... 39  
  3.9 Conclusion ....................................................................................... 40
### Natural Language Processing Technologies

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Introduction</td>
<td>41</td>
</tr>
<tr>
<td>4.2 Definite Clause Grammars</td>
<td>41</td>
</tr>
<tr>
<td>4.3 Using a Chart Parser</td>
<td>42</td>
</tr>
<tr>
<td>4.4 Conclusion</td>
<td>43</td>
</tr>
</tbody>
</table>

### My Design

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Introduction</td>
<td>45</td>
</tr>
<tr>
<td>5.2 System Outline</td>
<td>45</td>
</tr>
<tr>
<td>5.2.1 Translation from English to German</td>
<td>45</td>
</tr>
<tr>
<td>5.2.2 Translation from German to English</td>
<td>50</td>
</tr>
<tr>
<td>5.3 Resources used</td>
<td>51</td>
</tr>
<tr>
<td>5.3.1 Tagger</td>
<td>51</td>
</tr>
<tr>
<td>5.3.2 Chart Parser</td>
<td>53</td>
</tr>
<tr>
<td>5.3.3 Why use Prolog?</td>
<td>57</td>
</tr>
<tr>
<td>5.4 An Alternative Approach</td>
<td>58</td>
</tr>
<tr>
<td>5.5 Conclusion</td>
<td>61</td>
</tr>
</tbody>
</table>

### The Analysis Stage

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Introduction</td>
<td>62</td>
</tr>
<tr>
<td>6.2 Important Features of the English Grammar</td>
<td>62</td>
</tr>
<tr>
<td>6.2.1 Subcategorisation Information for English verbs</td>
<td>63</td>
</tr>
<tr>
<td>6.3 Important Features of the German Grammar</td>
<td>66</td>
</tr>
<tr>
<td>6.3.1 German Trees</td>
<td>66</td>
</tr>
<tr>
<td>6.3.2 Dealing with German Future Tense Indicators</td>
<td>67</td>
</tr>
<tr>
<td>6.3.3 Valency in German Grammar</td>
<td>69</td>
</tr>
<tr>
<td>6.3.4 Word Order of Complements</td>
<td>73</td>
</tr>
<tr>
<td>6.3.5 Subject-Verb Agreement</td>
<td>74</td>
</tr>
<tr>
<td>6.3.6 Inflecting Adjective Endings</td>
<td>75</td>
</tr>
<tr>
<td>6.4 Conclusion</td>
<td>77</td>
</tr>
</tbody>
</table>

### The Transfer Stage

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1 Introduction</td>
<td>78</td>
</tr>
<tr>
<td>7.2 Translation Memory</td>
<td>78</td>
</tr>
<tr>
<td>7.3 Translation of Verb Complements between Source and Target languages</td>
<td>79</td>
</tr>
<tr>
<td>7.4 Equivalence Rules</td>
<td>83</td>
</tr>
<tr>
<td>7.4.1 Finding the English tree using equivalence rules</td>
<td>84</td>
</tr>
<tr>
<td>7.4.2 Finding the German tree using equivalence rules</td>
<td>87</td>
</tr>
<tr>
<td>7.5 Basic Lexical Transfer</td>
<td>89</td>
</tr>
<tr>
<td>7.6 Conclusion</td>
<td>90</td>
</tr>
</tbody>
</table>
8 Interacting with the Translator

8.1 Introduction ................................................. 91
8.2 Java Interface .............................................. 91
  8.2.1 Why Java interface? .................................. 91
  8.2.2 Interface Design ...................................... 92
  8.2.3 Jasper: the package for mixing Prolog and Java .... 93
  8.2.4 What is a classpath and why do we need them? ... 95
8.3 Prolog shell .................................................. 96
  8.3.1 How to run the Prolog shell ......................... 96
8.4 Conclusion ................................................... 96

9 Useful Debugging Tips used for this Application

9.1 Introduction ................................................. 97
9.2 Test Classes ................................................. 97
9.3 Tracing ....................................................... 98
9.4 Spypoints .................................................. 99
9.5 Drawing Trees .............................................. 99
9.6 Conclusion ................................................... 100

10 Evaluation .................................................... 101

10.1 Introduction ................................................. 101
10.2 Sample Weather Data .................................... 101
10.3 German — English Evaluation ......................... 101
10.4 English — German Evaluation ......................... 106
10.5 Results and Conclusions ............................... 107

11 Conclusion .................................................... 111

11.1 Introduction ................................................. 111
11.2 Future Work ............................................... 111
  11.2.1 Adding a Morphological Plugin .................... 111
  11.2.2 Including an Interface to add to the German Lexicon ... 111
  11.2.3 Telegraphic Chunks .................................. 112
  11.2.4 Accept more than one German Sentence at a Time ... 112
11.3 Transformational Grammar ............................. 112
11.4 Achievements .............................................. 113
11.5 Summing Up ............................................... 113
Chapter 1

Introduction

‘Translation, in its full generality, is a difficult, fascinating, and intensely human endeavor’, Jurafsky and Martin (2000), pg 799.

Translating a text from one language into another is a form of art. The artist must make sure to represent the information in the source language as closely as possible in the target language. Language influences, who we are and how we think. Many cultural differences exist between natives speakers of different languages. Due to these cultural differences, a sentence in one language can not be exactly mirrored in another. A perfect translation is an illusion. It is often said that although during translation semantic content is transferred, subtle characteristics of the source language vanish.

Nevertheless, translation is a necessary and important part of everyday life. Businesses require translators to meet legal requirements and market demands as well as to gain a competitive advantage. Ireland as a member of the European Union encounters translation on a daily basis, so that agricultural, commercial and political information can be transmitted to and from all member states. The travel industry necessitate translators, even holiday makers need to be able to translate. We all have experienced the want for translation at some stage.

Regardless of the high demand for translations, the field of machine translation has barely changed since its earliest days in the fifties\(^1\). This reinforces the fact that machine translation is a difficult challenge. However, it is also a rewarding area, which is advancing slowly but surely.

\(^1\)cited by Kay, Martin (2001)
1.1 Motivation

This project appealed to me for a number of reasons. First of all the German language is very intriguing, it became obvious how much so, after participating in translation classes in Munich last year. It was very rewarding to be able to translate between German and English. Undertaking a project, which required programming seemed like a good step towards gaining some programming experience. Translation was the perfect solution, it is both challenging and multidisciplinary.

The foundation of my translation system is the ‘Transfer approach’, which incorporates knowledge of contrastive linguistics between source and target languages as well as using syntactic structures as the basis for translation. The transfer approach seemed to be the most desirable as it works well on a bilingual basis. In recent years machine translation has been actively pursued in Japan. It seems that the Japanese find the transfer approach to be the most productive.

‘By and large, the Japanese share the general perception that the transfer approach offers the best chance for early success’, Kay, Martin (2001).

The translation system, which I designed is restricted to the area of weather reports. This was for several reasons, firstly it makes use of a limited vocabulary and secondly it is repetitive by nature, which eliminates ambiguity, therefore allowing translation to be accurate.

1.2 Aims

Initially the main objective of this project was to design a transfer based translation system, which could translate weather reports in both directions between the German and English languages. Translation of full sentences, containing a main verb and the items it subcategorises for was the main goal. Here is a list of aims I had in mind, when initially starting out. Some of them were not realised due to time restrictions.

1. Build a German Grammar to analyse and generate grammatical German weather sentences, which could be later used for other purposes, such as grammar checking, testing learners of German on their knowledge of adjective endings, word order etc ...

2. Incorporate an English grammar and tagger

3. Write a translation module, which would contain equivalence rules to map English tree structures to German tree structures and vice versa. This subsequently had
to split into two modules; one containing transfer rules from German — English
and the other rules from English — German.

4. To build an interface, which would insulate the user from ugly Prolog predicates

5. To evaluate the translations produced by the translator

6. Write a transformational grammar to bridge deficiencies of the English parser

7. To evaluate the system using the edit distance algorithm

During the course of the year I managed to complete the first five aims.

1.3 Overview

In the course of this paper, I will outline the art of machine translation, listing some
approaches which are available today for carrying out the translation process (Chapter
2). I will also give a general overview of the linguistic differences which occur between
the German and English languages as well as mentioning morphological differences and
problems when translating between this language pair (Chapter 3). I will describe the
sub domain which my translator is built around. A description of some natural lan-
guage technology, namely definite clause grammars and chart parsers will be provided
next, since both are fundamental components in my system (Chapter 4). I will give an
outline of the main algorithms used in implementing this system and I will describe a
similar translator which is used for the translation of weather reports from French to En-
glish (Chapter 5). Next, descriptions of important areas during analysis of the German
and English grammars, including how the system deals with valency will be discussed
(Chapter 6). Following that the transfer stage of the translation process will be described
(Chapter 7). How to interact with the translator will be made clearer in (Chapter 8).
Tips and useful hints for debugging an application such as this one are mentioned in
(Chapter 9). Finally, illustration of how the translator works as well as an insight to the
results it produces are shown in (Chapter 10).
Chapter 2

The Art of Machine Translation

2.1 Introduction

‘I see translation as the attempt to produce a text so transparent that it does not seem to be translated. A good translation is like a pane of glass. You only notice that it’s there when there are little imperfections — scratches, bubbles. Ideally, there shouldn’t be any. It should never call attention to itself’, Venuti, Lawrence (1995).

There are many different methods for carrying out machine translation. The three main systems are classified by their strategy for carrying out the translation, these are the direct approach, the interlingua approach and the classical transfer approach. Other more recent approaches are the online interactive method and the learning based approach. In this chapter I will describe the main method behind each of these systems.

2.2 Translation Approaches

2.2.1 The Direct Approach

This is the original and oldest translation strategy, which was employed around the 1950s to 60s when a need for machine translation was mounting. Direct systems were the first generation of Machine Translation (MT) and they are usually built with one language pair (translation between two languages) in mind. This procedure involves taking a string of words from the source language, removing morphological inflections from the words to obtain the lemmas, i.e. base forms and then looking up the lemmas in a bilingual dictionary between the source and target language. After a translation of each word is found, the positions of the words in the string are altered to best match the word order of the target language, these may include Subject-Verb-Object (SVO) rearrangements, among others\(^1\).

\(^1\)cited by Hutchins, W. J (1986)
Obviously this system has severe limitations. This is a result of neither syntactic nor semantic analysis taking place. As we all know, most words can convey different semantic content when used in different contexts. Since direct MT treats a sentence as a string of words, interdependencies and semantic grouping among words can be lost and choosing the wrong interpretation for a given word becomes quite likely. Direct translation is linguistically weak, i.e. it lacks taking structure and relationships between words into account. Direct translations loose a feeling of naturalness. Although we all have some cognitive dictionary, which we can access in our heads, we very rarely articulate a sentence by individually choosing words from our mental lexicon. We choose words that are relevant and which are connected to each other, phrases if you like, because of this, breaking the phrases up into individual words as well as trying to retain the overall meaning is very difficult.

‘Each group of words is processed by the brain as a single thought. And because the words are viewed in context, you retain them more accurately than if you processed the words individually’, (Rose Saperstein (poet))

Here is an example of a mistranslation between Russian and English, where analysis at the lexical level proved to be inaccurate and insufficient.

’Ona navarila ščei na neskol’ko dnei’

It welded on cabbage soups on several days
She cooked enough cabbage soup for several days

’My trebuem mira’

We require world
We want peace

Direct translation lacks an intermediate stage and maps directly between source and target languages. It can be used in conjunction with other strategies and can in this way
offer various advantages. Météo is a system I will discuss in section 5.4, which translates weather reports between French and English. It incorporates a semi direct approach and it is very successful.

2.2.2 The Interlingua Approach

Together the interlingua and transfer strategies form the second generation of MT.

![Figure 2.2: Interlingua Approach](image)

The interlingual approach is the most attractive approach for a multilingual system. First of all, a sentence in the source language is analysed, then its semantic content, i.e. meaning, is extracted and represented by a language independent canonical form, i.e. base form. The successful motivation behind this technique is that given this abstract representation, a natural language sentence can be generated using a generation module between the representation language and the target language. To include an additional language to a translator of this type simply add an analysis module and a generation module for the new language to be represented. This offers the advantage that the system grows linearly $2^n$, where $n$ is the number of languages. An interlingual system can translate between all pairs of languages, that were represented. For example, if the system had 6 modules, which are:

- German - Analysis and Generation
- English - Analysis and Generation
- French - Analysis and Generation

Then this machine would be capable of translating in all of these directions:

An interlingual system can translate from English back to English, as mentioned above. This ‘back-translation’ capability could in fact prove very valuable during system development in order to test analysis and generation modules. The back-translating may

---

2A multilingual system is one that can translate between multiple languages
give back a syntactically different sentence, however the meaning should be consistent. Although this strategy is appealing, it also has associated difficulties.

‘A truly 'universal' and language-independent interlingua has defied the best efforts of linguists and philosophers from 17th Century onwards’, Hutchins (1986), pg 123.

Finding a language independent representation which retains the precise meaning of a sentence in a particular language, which can then be used to generate a sentence of a different language is a seriously challenging task. Considerations which must be dealt with are the decision of which representational ontology to use and how to store language specific details in a general representation. Once a sentence has been stored in its interlingual representation, looking back at the source language should not be necessary, as all relevant information should now be stored in its new interlingual form. An example of a difficulty using interlingua would be representing the word ’wall’ in German in a canonical form, since it can be translated as *der Mauer* - the outside wall, or *die Wand* - the inside wall, therefore the interlingual representation of it would have to convey which type of wall the text was referring to. Since both words are represented by the one English word, it requires additional information to be added into the interlingual representation from English → German direction, while supplying redundant or unavailable information in the German → English direction. This problem requires the system to fully disambiguate at all times. However this approach works extremely well in subdomains, for example for holiday reservations, flight reservations etc... because ambiguity is greatly reduced in these situations.

### 2.2.3 The Transfer Approach

This is the approach I adopted for my implementation. The transfer approach is used on the basis of the known structural differences between the source and target language. A
Figure 2.4: Transfer Approach

Transfer system can be broken down into three stages: Analysis, Transfer and Generation. The transfer method presupposes a parse tree of the input in the source language, this is known as the analysis stage, then this parse tree is mapped to a parse tree of the target language. All of this is done in a transfer module which maps semantically equivalent but syntactically different trees of the source language to the target language. Finally after finding the parse tree of the target language it is put into some grammar module, which will take the tree as input and will output the corresponding natural language sentence. When the source language is used to produce a parse tree, this parse tree will usually contain the base form of the words in the sentence instead of their inflected forms. The parse tree will represent the main elements of the sentence, i.e. the verb, complements etc... and many sentences of the source language can be represented by the same tree structure, but with different lexical words in them. An example for this would be:

\[ \text{das Wetter wird sonnig sein} \]
the weather will sunny be

‘the weather will be sunny’

and

\[ \text{Schnee wird fallen} \]
‘Snow will fall’

Both of these can be generalised by this tree:

This was an example of how my German grammar would parse the given input. This will be discussed further in section 6.3.1. Parse trees can be designed to be conventional or abstract. Trees, which are conventional are closely related to the sentence they represent, with words appearing in the tree in the same order as in the sentence etc... Abstract trees on the other hand contrast to conventional trees, in these the word order of sentence constituents is not closely followed in the tree representations. Probably for a good transfer system it should be quite abstract. However, remember that the more abstract
your representations become, the more you move towards an interlingua approach. An advantage of transfer is that when you are using similar languages, which share the same syntax at times, such as Spanish and Portuguese, parts of the transfer system can be shared. Transfer systems are usually bilingual and they map one source language to a limited number of target languages. If the system was to be multilingual, you would need a minimum of:

\[ \frac{n(n-1)}{2} \]

transfer modules for \( n \) number of languages.

This is because in a fully multilingual system, there are \((n-1)\) possible target languages for each of the \(n\) languages. This number \(n\) can be halved if the analysis and generation modules for each language are reversible.\(^3\)

There are three types of transfer systems, namely syntactic, semantic and lexical. My implementation is based on syntactic transfer. In my implementation I have one German module, which is responsible for both analysis and generation. On the English side, I also have one module, which is responsible for analysis and generation. My German module is fully reversible, however, my English module is not reversible.

### 2.2.4 Online Interactive Systems

This type of approach can allow the user to guide the translator to the correct translation. The user must answer all queries which the translator needs to be answered. Typical questions include which part of speech to assign to a word when its meaning is ambiguous. This can be time consuming, but it eliminates guesswork or default answers being supplied by the translator. Direct involvement on behalf of the user is made possible, during the course of online translation. The most typical case is when the context of a word is unclear. When there are many possible meanings for one word and the translation system is unsure of which interpretation is being referred through structural ambiguity, the problem is resolved quick by double checking an interpretation with the

\(^3\)A module is reversible if you only need one module for both generation and analysis.
user. For example, imagine you are a native English speaker and you need to translate a letter to your Japanese correspondent on the Jet program. You want to ask him what kind of clothes you should wear while teaching, casual formal... You know that there are many Japanese verbs equivalent to the verb ‘wear’ in English, but you don’t know which one means ‘to wear a shirt and tie’. The translator simply asks:

- **to translate ‘wear’, please indicate which wear you need by choosing the item ‘wear corresponds to’:**
  1. coat or jacket,
  2. shoes or trousers
  3. hat
  4. ring or gloves
  5. belt or tie or scarf
  6. brooch or clip
  7. glasses or necklace
  8. moustache
  9. general or unspecific

As you can see such a system may be time-consuming but it can also lead to very accurate translations. After weighing up the three approaches, I chose to use the transfer method, this is because I was using the method for a bilingual system, therefore it did not matter that it was expensive to expand the transfer method to incorporate another language. Also, I wanted to translate on the basis of syntax as it is the area of linguistics that interests me most.

### 2.2.5 Learning Based Translation

Another mechanism which is available today is the example based translation method. This method is based on analogical reasoning between two translation examples. It assumes the existence of a bilingual parallel text to derive a translation. This approach offers the advantage of producing results quickly. A further advantage of such an approach is that unlike the other methods I have described systems based on this method to not require large-scale knowledge about the source and target languages, i.e. grammars, transfer modules etc... This type of system generally does not deal with analysing or generating morphology either. Instead it takes translations without considering what case they are in and stores them just in memory without any change to the representation. This can lead to inaccurate morphological inflection of words in a highly inflectional language, if for example, the case of the object in the example translated text conflicts with the case of the object in the sentence to be translated. This could lead to the words being incorrectly inflected given the context. The fundamental component of this type of system would be some kind of pattern matcher. A successful system of this type can be found here Güvenir and Cicekli. (1994).
2.3 Conclusion

This chapter has explored the most common approaches of machine translation available today. I have explained why an interlingual approach is the most attractive for a multilingual translator and have shown that the transfer approach is suitable for a bilingual system. Since my system was to be bilingual this approach seemed most beneficial to me.
Chapter 3

The German - English Domain

3.1 Introduction

The first section of this chapter is provided as a quick rundown of the differences between the German and English languages. This was written for those, who are unaware of the fundamental differences between German and English, including morphological differences. Possible improvements of morphological analysis and generation in this system will be mentioned. Anyone who is already aware of these contrasts could skip this section. The rest of the chapter describes the sub-domain, on which the system is based on and it highlights the characteristics, which were found to be common within this sub-domain. Based on these weather report traits a list of assumptions were incorporated into the system, these are fully explained in section 3.5.

3.2 Basic Differences between German and English

First of all, German is a highly inflected language and English is not, this basically means that, the base form of a word in German is realised in a larger number of distinct inflected forms than it is in English. Sometimes features added are entirely new to English speakers such as gender in German nouns. Here are some facts of the German language in a nutshell:

1. In German, all nouns are distinguished by being written with an initial capital letter.

2. Every noun in German is assigned to one of three genders: masculine, feminine or neuter. This gender differs from natural gender. Only singular nouns are assigned a gender:

   'das Wetter’ - the weather is a neuter noun
'der Regen' - the rain is a masculine noun
'die Winde' - as this is plural, it does not carry gender information

3. German has four cases: nominative, accusative, dative and genitive. Each noun phrase of a sentence is assigned a particular case based on its role in the sentence. For example, the subject or 'doer' of the sentence is always in the nominative case. This allows the word order to be quite free, in comparison to English, where the subject must always precede the main verb of the sentence, to indicate that it is the subject.

4. In English, many nouns simply add an -s to form their plural, there are exceptions to this rule also. In German, there is no one general rule. In English there is a very small number of possible plural suffixes, however in German there is quite a large number of ways to make a plural, some examples are adding -n, -en or addition of an umlaut(¨) and an e, etc... Here are some examples:

<table>
<thead>
<tr>
<th>German(plural suffix)</th>
<th>English(plural suffix)</th>
</tr>
</thead>
<tbody>
<tr>
<td>die Schneeflocke(n)</td>
<td>snowflake(s)</td>
</tr>
<tr>
<td>der Wind(e)</td>
<td>wind(s)</td>
</tr>
<tr>
<td>der Schwall(e)</td>
<td>gush(es)</td>
</tr>
<tr>
<td>der Fang (¨e)</td>
<td>catch(es)</td>
</tr>
<tr>
<td>der Regen(-)</td>
<td>rain(-)(^1)</td>
</tr>
</tbody>
</table>

Figure 3.1: Forming Plurals

5. In English, adjectives are inflected when using the superlative or comparative, but otherwise they remain uninflected. However, in German, inflection occurs in all adjectives except when they are being used predicatively, i.e. being used to complement a noun which is the subject or object of a verb, for example, it is 

\textit{sunny}. Attributive adjectives\(^2\) have endings which indicate case, number and gender. Their endings also depend on what type of determiner precedes them. Endings vary for adjectives following no determiner, \textit{bestimmt} determiners\(^3\) or \textit{unbestimmt} determiners. I will describe each of these fully in the next section.\(^4\)

6. Finally all German verbs subcategorise for objects of a particular case. However, in German the verb can sometimes subcategorise for an object of an unexpected case. \textit{helfen} - to help is an example of such a verb. You would imagine the verb \textit{helfen} would require a nominative object, as its subject and an accusative object

\(^2\)Attributive adjectives are adjectives that precede and describe nouns.

\(^3\)\textit{Bestimmt} determiners are articles relating to definite objects, such as the, that, this etc..

\(^4\)\textit{Unbestimmt} determiners are articles referring to indefinite objects, such as a, some, etc., and pronouns, for example, mein - "my"
as its helpee, its direct object, just like most other German verbs. However this is not the case the verb *helfen* requires its object to be a dative instead of the expected accusative case.

7. Word order is more flexible in German than in English. Evidently, English uses word order to identify the subject and objects of the sentence, while German uses case for this purpose. The subject and objects, in a German sentence are marked with case and they can easily be identified, regardless of their position in the sentence. There are exceptions to this rule, for example when two words have the same inflectional ending attached, but are in different cases,

\[
\text{die Frau traf die Kinder}
\]

"die Frau" in this example could be an accusative or a nominative object and so could the noun "die Kinder" as both nouns are preceded by "die" and this can either represent an accusative or a nominative noun. Usually morphological endings indicate who are the objects and the subject of a sentence, although there are cases when this is insufficient as can be seen above.

8. In German, the finite verb almost always appears in the second position of main clauses. It can be preceded by a time adverbial, subject or object etc... All other parts of the verb, i.e. the past participle, infinitive or separable prefix, are in final position in a multi-verb sentence.

9. German Prepositions govern case (each preposition takes a case, which is nominative, accusative, dative or genitive). As each German preposition is assigned a case, in a prepositional phrase the noun phrase which is headed by that preposition must be of the case. Furthermore some prepositions ‘wechsel präpositionen’ - *changing prepositions* can change case depending on the situation. For example, the preposition *zwischen* - "between" is in the dative case when it is used to indicate a situation without movement. However, it requires an accusative noun when it is referring to a situation involving movement.

\[
\text{die Wand zwischen den Häusern}_{\text{dat}} \\
\text{the wall between the houses} \\
\text{(no movement)}
\]

\[
\text{er geht zwischen die Häuser}_{\text{acc}} \\
\text{he walks between the houses} \\
\text{(with movement)}
\]
3.3 German Morphological Issues

As I have mentioned before, German is a highly inflectional language, much more so than English. This can be seen in the fact that much more frequently than in English, German requires determiners, adjectives and nouns, in certain cases, to be inflected. Here I will explain the differences between determiners in German and English.

English has determiners such as, *the, a, some, your, my etc.*. However in German there are many forms for each of these words. German determiners are declined according to case, number and gender of the noun to which they are attached. Determiners can be subdivided into two types definite articles - 'bestimmt', indefinite articles - 'unbestimmt'. Also adjectives can be used as articles. Here I will explain the three types, definite articles, indefinite articles and adjectives as articles.

### 3.3.1 Definite Articles

Some German articles take the definite article endings, these are,

1. definite article: der, das, den - *the*
2. interrogatives: welcher? - *which*
3. demonstratives: dieser - *this* jener - *every*
4. quantifier: mancher - *some*, aller - *all*

<table>
<thead>
<tr>
<th>Case</th>
<th>Masculine</th>
<th>Feminine</th>
<th>Neuter</th>
<th>Plural</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominative</td>
<td>der</td>
<td>die</td>
<td>das</td>
<td>die</td>
</tr>
<tr>
<td>Accusative</td>
<td>den</td>
<td>die</td>
<td>das</td>
<td>die</td>
</tr>
<tr>
<td>Dative</td>
<td>dem</td>
<td>der</td>
<td>dem</td>
<td>dem</td>
</tr>
<tr>
<td>Genitive</td>
<td>des</td>
<td>der</td>
<td>des</td>
<td>der</td>
</tr>
</tbody>
</table>

![Figure 3.2: Declination of Definite Articles](image)

Here is an example,

'Dieser Rock sieht gut aus'
This skirt looks gut out
*This skirt looks well*

Rock - *skirt* is masculine and here it is in the nominative case. In English there is only one form of this word and it is *'the'*.
3.3.2 Indefinite Articles

This category includes two types, indefinite articles and pronouns, which can be seen here,

1. indefinite articles ein - a, kein - none
2. possessives mein - mine, dein - your

<table>
<thead>
<tr>
<th>Case</th>
<th>Masculine</th>
<th>Feminine</th>
<th>Neuter</th>
<th>Plural</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominative</td>
<td>mein</td>
<td>meine</td>
<td>mein</td>
<td>meine</td>
</tr>
<tr>
<td>Accusative</td>
<td>meinen</td>
<td>meine</td>
<td>mein</td>
<td>meine</td>
</tr>
<tr>
<td>Dative</td>
<td>meinem</td>
<td>meiner</td>
<td>meinem</td>
<td>meinen</td>
</tr>
<tr>
<td>Genitive</td>
<td>meines</td>
<td>meiner</td>
<td>meines</td>
<td>meiner</td>
</tr>
</tbody>
</table>

Figure 3.3: Declination of Indefinite Articles

Example,

'Mein Hund heisst Ralf'
My dog names Ralf
My dog is called Ralf

3.3.3 German Adjective Inflection

In my German grammar, I listed all of the forms for der - the and ein - a. I could not break these down to a general rule as there are not enough similarities between the possible words requiring the endings. However, this is one point that could benefit from including a German morphosyntactic plug in, such as the Treetagger\(^5\). I will discuss this further in the next chapter. Another area where I included possible inflections was for adjectives. I mentioned the differences between German and English adjectives above. These are English adjectives are inflected only when used in a comparative or superlative context, however in German attributive adjectives are inflected in three ways. There are three tables of inflectional suffixes for German adjectives. The first list the ending attached to German adjectives when they are preceded by a ‘bestimmt’ - definite determiner: the second are adjective endings for adjectives preceded by an ‘unbestimmt’ - indefinite determiner:

and the represent adjectives which are not precede by a determiner and which take on the combined role of determiner and adjective:

\(^5\)see http://www.ims.uni-stuttgart.de/projekte/corplex/TreecTagger/DecisionTreeTagger.html
<table>
<thead>
<tr>
<th>Case</th>
<th>Masculine</th>
<th>Feminine</th>
<th>Neuter</th>
<th>Plural</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominative</td>
<td>-e</td>
<td>-e</td>
<td>-e</td>
<td>-en</td>
</tr>
<tr>
<td>Accusative</td>
<td>-en</td>
<td>-e</td>
<td>-e</td>
<td>-en</td>
</tr>
<tr>
<td>Dative</td>
<td>-en</td>
<td>-en</td>
<td>-en</td>
<td>-en</td>
</tr>
<tr>
<td>Genitive</td>
<td>-en</td>
<td>-en</td>
<td>-en</td>
<td>-en</td>
</tr>
</tbody>
</table>

Figure 3.4: Adjective Endings following Definite Article

<table>
<thead>
<tr>
<th>Case</th>
<th>Masculine</th>
<th>Feminine</th>
<th>Neuter</th>
<th>Plural</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominative</td>
<td>-er</td>
<td>-e</td>
<td>-es</td>
<td>-en</td>
</tr>
<tr>
<td>Accusative</td>
<td>-en</td>
<td>-e</td>
<td>-es</td>
<td>-en</td>
</tr>
<tr>
<td>Dative</td>
<td>-en</td>
<td>-en</td>
<td>-en</td>
<td>-en</td>
</tr>
<tr>
<td>Genitive</td>
<td>-en</td>
<td>-en</td>
<td>-en</td>
<td>-en</td>
</tr>
</tbody>
</table>

Figure 3.5: Adjective Endings following Indefinite Article

<table>
<thead>
<tr>
<th>Case</th>
<th>Masculine</th>
<th>Feminine</th>
<th>Neuter</th>
<th>Plural</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominative</td>
<td>-er</td>
<td>-e</td>
<td>-es</td>
<td>-e</td>
</tr>
<tr>
<td>Accusative</td>
<td>-en</td>
<td>-e</td>
<td>-es</td>
<td>-e</td>
</tr>
<tr>
<td>Dative</td>
<td>-em</td>
<td>-er</td>
<td>-em</td>
<td>-en</td>
</tr>
<tr>
<td>Genitive</td>
<td>-en</td>
<td>-er</td>
<td>-en</td>
<td>-er</td>
</tr>
</tbody>
</table>

Figure 3.6: Adjective Endings following no Article
3.4 Morphology

‘If the English language made any sense, catastrophe would be an apostrophe with fur’, (Doug Larson)

In the following section, the current state of morphological analysis will be briefly outlined. The morphological restrictions of my system will be explained and other taggers as well as possible morphological plugins will be introduced. These plugins could perhaps later be used to improve the current system.

In the last 10-15 years computational morphology has advanced more than any other area of computational linguistics. Today morphological analysers are available for all the most commercially important languages. An efficient parser must be able to recognise the canonical form of a word as well as inflected forms and their morphological properties. There are two main problems when trying to build such a parser, they are:

1. Morphotactics ‘the model of morpheme\(^6\) ordering that explains which classes of morphemes can follow other classes of morphemes inside a word. For example, the rule that the English plural morpheme follows the noun rather than preceding it\(^7\) - the parser needs to know what rules are valid for certain parts of speech.

2. the same morpheme may look the same, but mean different things in two different contexts

The approach most commonly used, to find the canonical form of an inflected word is the "Cut and Paste" method. The canonical form of an inflected word was found by cutting off the inflection on the end, while generation involved adding the correct affix onto the word. In the 1960s the first cut and paste machine was developed and it was called MITalk’s DECOMP\(^8\). In the MAGIC system\(^9\), cut and paste rules are applied in advance to produce the correct form of a word. In the 1980s finite state technology was introduced to handle morphological processes. Finite state transducers are able to generate and analyse morphological alternations. Morphological inflections are stored as branches of a network and the surface string is mapped to a sequence of branches in the letter trees using the transducers and computes the lemma\(^10\) from information provided at the branch boundaries.

---

\(^6\)A Morpheme is a minimal meaning-bearing element in a language, e.g. to indicate possession or cat - to refer to an animal.

\(^7\)cited by Karlsson, F. and Karttunen, L (2001), pg 1

\(^8\)Allen, Hunnicutt and Klatt (1987)

\(^9\)Schüller, Zierl and Hauser (1993)

\(^10\)The combination of a surface form and it’s analysis as a canonical form and inflection is called a lemma, cited by Karlsson, F. and Karttunen, L. (2001)
‘Inflection is the combination of a word stem with a grammatical morpheme, usually resulting in a word of the same class as the original stem and usually filling some syntactic function like agreement’. Jurafsky and Martin (2000), pg 799.

As mentioned earlier, German is a highly inflectional language. Each word that is inflected must be recognised as a variant of its base form. In German, nouns, verbs and adjectives, among others can all be inflected. This means that all variants of each verb, adjective, nouns, determiners etc. must be stored in the lexicon in order for a grammar to recognise a word. As you can imagine this requires thousands of extra words to be added to the lexicon. If we just consider one German verb the present form of sein (‘to be’), see figure 3.7, there are many forms and this is just in its present form, not to mention all the other possible tenses.

<table>
<thead>
<tr>
<th>sein</th>
<th>to be</th>
</tr>
</thead>
<tbody>
<tr>
<td>ich bin</td>
<td>I am</td>
</tr>
<tr>
<td>du bist</td>
<td>you are</td>
</tr>
<tr>
<td>er/sie/es ist</td>
<td>he/she/it is</td>
</tr>
<tr>
<td>wir sind</td>
<td>we are</td>
</tr>
<tr>
<td>ihr seid</td>
<td>you(pl) are</td>
</tr>
<tr>
<td>sie sind</td>
<td>they are</td>
</tr>
</tbody>
</table>

Figure 3.7: Present Form of sein to be

It is quite obvious that a lot of words would have to be added to account for all possible inflections of every noun, verb and adjective in German. English on the other hand has a relatively simple inflectional system, nouns, verbs and some adjectives can be inflected and the number of inflectional affixes is usually quite small. However, once again when you consider all possible word inflections of English nouns and verbs the number is huge. Some sample noun inflections in English appear here in bold:

Mary’s

*temperatures

*going

*rain ed

*children

As we can see a more efficient method of recognising and generating inflections of German and English words is needed. The chart parser that I used automatically recognised and generated all inflections of my English data. For the purposes of my implementation, taking time restrictions into account, I decided to add all singular and plural forms of
the German nouns I used and I decided to add both the 3rd person and plural forms of all verbs that I included in my German grammar. The reason for this was that I found this method sufficient enough to cover most of the sentences I wanted to generate and analyse. I was able to restrict verb forms to the 3rd and plural as these forms are the only two that are used frequently in weather reports. Reporters usually speak of the rain, snow, wind and the impersonal ‘it’ as in ’it will rain’ or else they use plural verb forms such as ‘areas of high pressure will spread ...’, or ‘temperatures will increase to ...’ If however, I had more time at my disposal I would have taken a different approach. I would have used a morphosyntactic plugin, such as Supertagger or the Collins Tagger.... I will discuss both of these in the following section.

3.4.1 Alternative Taggers & a Morphology System

I chose to use a tagger and chart parser which I will describe later in section 5.3. However, there are many parsers, which I could have chosen to use. Two successful English parsers which are freely available are the Collins Parser\(^{11}\) and the SuperTagger(97)\(^{12}\). Both of these parsers were developed at the University of Pennsylvania, Philadelphia. Although I found the chart parser sufficient most of the time, there were areas where it was insufficient. One of its downfalls was that it could only parse sentences consisting of a verb group and the complements it subcategorises for. This had implications on the resulting translations, since sentences and the complements their matrix verbs subcategorise for were separated from all other additional elements an this in turn forced time adverbials and location indicators to be translated independently from the sentence they should belong to. An example of this can be seen in section 5.3.2. If I had used one of these parsers instead of the one I chose, I could have overcome this problem, as they both allow sentences to contain a verb group, its complements and additional complements. Here I will briefly describe each of these parsers.

The Collins parser was designed by Michael Collins in 1996. It is trained on a corpus annotated with phrase-structure trees. The most likely parse for a given sentence is determined by the probability of specific word-to-word dependencies. On the other hand the ‘Supertagger’ is trained on a corpus, where each lexical item has been annotated with the tree, that is associated with it in a correct Tree-Adjoining grammar parse, these are known as ‘Supertags’. These ‘Supertags’ are then used to heuristically determine the most likely parse. If I decided to change my system to incorporate either of these parsers, I would gain the ability to parse long sentences, which would include additional information outside the verb’s valency list. Using either of these parsers to analyse the English input would mean implementing a ‘converter’, as neither parser produces an output in the format needed for my transfer modules. The tagger and parser I chose to use was

\(^{11}\)Collins (1996)

\(^{12}\)see http://www.cis.upenn.edu/ mickeyc/SuperTags
successful for most of the translations I wanted to carry out, it produced conventional parse trees, which could be used in my transfer module and it was usually very successful in finding the correct parse, but perhaps using an alternative tagger and parser would prove the capability of the parser in recognising complete sentences.

Morphology systems, part-of-speech taggers and lemmatisers are fundamental natural language tools, which are necessary for many NLP tasks including machine translation, parsing, information extraction etc. Unfortunately, for languages outside of English these tools are rarely available. Although the range of available German NLP tools is considerably lower than the English tools available, I did come across some interesting finds. On the German side I could have used a tagger such as the TreeTagger. TreeTagger is a tool used for annotating text with part of speech and lemma information. It was developed by the Institute for Computational Linguistics at the University of Stuttgart. The TreeTagger is a multi-language tagger, which can tag French, English, German, Greek and Italian. An sample output of the TreeTagger when given the sentence ‘the Treetagger is easy to use’ can be seen in Figure 3.8.

<table>
<thead>
<tr>
<th>word</th>
<th>pos</th>
<th>lemma</th>
</tr>
</thead>
<tbody>
<tr>
<td>The</td>
<td>DT</td>
<td>the</td>
</tr>
<tr>
<td>TreeTagger</td>
<td>NP</td>
<td>Treetagger</td>
</tr>
<tr>
<td>is</td>
<td>VBZ</td>
<td>be</td>
</tr>
<tr>
<td>easy</td>
<td>JJ</td>
<td>easy</td>
</tr>
<tr>
<td>to</td>
<td>TO</td>
<td>to</td>
</tr>
<tr>
<td>use</td>
<td>VB</td>
<td>use</td>
</tr>
</tbody>
</table>

Figure 3.8: TreeTagger Output

Another possible tagger I could have used was the Brill-Tagger. The Brill-Tagger was created by the computational linguistic group at the University of Zurich. This tagger is based on a training corpus of around 58,000 words. I could first enter my German input sentences into either of these taggers which would then find the base form of each word in the input. These baseforms could then be added into the German tree representation of the input and this tree could be used in the equivalence rules during the transfer stage in order to find the equivalent English tree. This could be easily done as I designed my German trees to contain the baseform of the words in the tree and not the inflected form of words. This was sufficient because when my German grammar is given a German tree it can return the inflected words of the sentence represented by that tree. This

---

13 For further information on the TreeTagger see http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html
14 For more information on the Brill Tagger see http://www.mozart-oz.org/mogul/doc/lager/brill-tagger/
tagger can produce the lemma and part of speech for a word. However, this would have implications for the generation process. The tagger would work very well during analysis of input. However, the tagger cannot produce inflected words given their base forms and morphological features, since it does not store all morphologically inflected words and their features in a lexicon. While both of these would increase the German analysis of the translation process by being able to a recognise a larger number of inflected forms and map them to their lemma, this would only improve my system in translating from German to English. This would not improve translation from English to German since it could not provide inflected forms of words given a lemma and a list of features. In order to achieve this another morphological plugin would need to intervene.

There is currently a system available on the internet which combines both a German part-of-speech tagger and a morphology-system combined; it is called Morphy\textsuperscript{15}. It can be downloaded from the internet. This in contrast to the other taggers mentioned above can both recognise the lemma form of a given inflected word as well as being able to inflect a base form, when given a word and its morphological features. This system would immensely improve my current system as it would improve German analysis and generation. Of course to incorporate this software into my system would involve writing converters to link its representation of features etc .. to my current representation. Among other morphological features morphy considers vowel mutation, shifts between ß, and ss as well as pre- and infixation of markers for participles. At the start of this chapter I introduced morphological analysis and described the ‘cut and paste method’ for processing morphology, Morphy uses this ‘cut and paste method’. It cuts off all possible suffixes on analysis of a word and on generation attaches some possible suffix. This approach means that the system is slower than a system which contains a lexicon containing all possible inflections of a word, but it also offers the advantage of only requiring a small space to store it’s lexicon.

3.5 The Sub-domain

Achieving a good quality translation is often the result of human intervention. Translation machines translate what they can and then their output is post-edited or sometimes the text is pre-edited beforehand, to ensure a successful translation. Another alternative is to restrict input data to a sub-domain of a language. This is the approach I opted for in this system. The domain I chose was weather reports. The reason for my choice was that weather report data has a limited vocabulary and makes use of a limited number of basic phrase types. This automatically reduces ambiguity, since the words are assumed to be in a weather report context. This means that when the phrase 'Snow will fall in the east' is given as input to the translator, the system can confidently translate it as:

\textsuperscript{15}Lezius (2000)
'Schnee wird im Osten fallen'
Snow will in the east fall
'Snow will fall in the east'

and not as

'Schnee wird im Osten abstürzen'
Snow will in the east fall
here the system finds the incorrect translation of 'fall'

'abstuerzen' does means to fall however it is a different meaning of fall, namely to fall off a bike. The idea of a "subdomain" originated from work at Montreal in Canada on a system designed specifically to translate weather forecasts from English to French using a system known as Météo. I will describe Météo and explain how it is successful later in section 5.4. Initially I chose to base my system on full English sentences instead of these abbreviated phrases. However, later on I decided to incorporate the translation of telegraphic phrases\textsuperscript{16} into my system.

My Subdomain will consist of:
1. German Weather Report Terminology
2. English Weather Report Terminology

3.6 Characteristics of German & English Weather Data

In this section, I will highlight the characteristics which were common in the weather data which I collected. Initially, I felt that weather reports were a useful and practical subdomain for three reasons. Firstly weather reports are used on a daily basis and are important for people travelling between Germany and Ireland. Secondly, weather reports are repetitive by nature and therefore make successful translations possible, finally weather reports are a source which has heavy demands laid upon it, every day people want to find out what the weather will be like. I began looking on the internet at various sites in English and German, which provided weather information. These were usually newspaper sites and holiday guides. However, I found that the internet was not a great source for finding written data as most weather information was given as pictures and abbreviated telegraphic phrases. I did find some full descriptions of forecasted weather

\textsuperscript{16}Telegraphic phrases are small pieces of information in the form of short phrases as opposed to full sentences
on the internet, however they were few and far between. A typical internet style would be,

teilweise bewölkt - *at times, cloudy*
maximal 11 - *maximum 11*
minimal -2 - *minimal -2*
*(taken from yahoo in German)*

At the beginning I read about systems such as Météo, described later in section 5.4, and saw that their fundamental purpose is to translate these telegraphic chunks. However, I decided that I would rather my translator translate grammatical sentences. Initially, I also collected numerous weather reports from German newspapers and I wrote to Met Éireann, the Irish meteorological service, and received template and sample weather reports in English. I also collected a number of samples from Irish newspapers. Samples of these can be seen in Appendix A. Here I found a lot more grammatical sentences than on the internet. However, in these reports also, telegraphic descriptions were quite common. It was at this point I realised that weather reports without telegraphic style descriptions are very uncommon. So I decided to incorporate them too into my translator. Here are some characteristics of both German and English weather reports,

1. Sentence are almost always in the future tense

2. Past tense is rarely included - there’s no point in predicting what already has happened, so I decided not to include past tense sentences into the transfer rules, however they are included in the German grammar, this would enable past tense translation if I changed the domain later on.

3. telegraphic speech is quite common

By the end of the project I had realised that the sub-domain of weather reports is not so small after all and although ambiguity is reduced, it covers a very wide variety of vocabulary and structures. I then realised why Météo and such systems have restricted their sub-domain to a smaller set of phrases and structures in side the sub-domain of weather. In order to optimise the translations outputted by my translator, I too had to decide on what parts of this sub-domain I would deal with. I will discuss these decisions in the next section.

### 3.7 Assumptions made by this Application

I decided to include translations for all these types of phrases:

- Prepositional phrases such as ‘with some rain’ - *mit etwas Regen*, ‘with sunny spells’ - *mit sonnigen Perioden*, this was to incorporate translation of the commonly used telegraphic style of phrases found in weather reports
• Time adverbials including ‘on Monday’ - am Mittwoch for all days of the week, ‘on Friday evening’ - am Freitag Abend, ‘during the day’ - tagesueber - ‘for the entire day’ - fuer den ganzen Tag, ‘in the early hours of the morning’ - in Fruehe . . .

• Spatial adverbials of the form ‘in the east’ - im Osten, ‘in the north west’ - im Nordwesten . . .

• Sentences including all complements required by its verb group. This means if a verb requires a nominative noun phrase and an accusative noun phrase, the sentence must include both of these and no other additional complements. This is because the English chart parser which I use to analyse the English input can only recognise sentences including a verb group, its complements and nothing else. Additional sentence constituents, including prepositional phrases indicating time . . . will be translated separately as individual phrases. Therefore if the translator is given the English sentence ‘On Monday, it will be a cold day’, it will break it into two parts to be translated into German, these are ‘on Monday’ and ‘it will be a cold day’. Both of these will be translated into English and returned separately. Further discussion on this topic can be found in chapter 6.

• The translator can translate a selection of English sentences in the future and present tense into German.

• It will translate a selection of German sentences in the future and in the present tense into English.

• Sentences in the past tense will not be translated in either direction. However the German grammar can analyse some sentences in the past tense, I included this initially before being aware that the past does not occur frequently in weather reports.

• Relative clauses and passive sentences very rarely occur in weather reports. The central subject of weather talk is ‘it’, the subjectless noun phrase as in ‘it is raining’ and in these cases a passive is not possible. In other sentences, it is also very unusual to see the passive in these reports. Relative clauses do not occur frequently in weather forecasts either and I think this is simply because weather reports use simple language. They use short concise sentences and very rarely need to refer back to a noun or add any further description. The chart parser, which I used cannot analyse passive sentences or relative clauses. If either passives or relative clauses occurred more often in weather reports I would have added them to the translation system; however, since they don’t, I decided not to include them in my German grammar either, and therefore my translator cannot translate relative clauses or passives.
Although the English chart parser cannot handle additional elements outside of what the main verb subcategorises for, the German grammar can. I have included rules within the German grammar to allow sentences have additional time adverbials, prepositional phrases etc.. This allows the German grammar to be used independently from the translator to check the grammaticalness of German sentences.

3.8 Linguistic Problems during Transfer

3.8.1 Tricky Translations

‘A reflection involves a mirroring, a copy of an original; a refraction involves changes of perception, and this is an image that is useful to describe what happens when a text crosses from one culture to another’, Bassnett Maguire. S (1991), pg xvii.

As with all areas of Natural Language processing, there are some problem areas in machine translation too. Structure differences between the source and target languages, as well as semantic and lexical variations lead to divergences and mismatches during the translation process. According to Trujillo(1999), pg 126.

‘A translation divergence normally implies that the meaning is conveyed by the translation, although syntactic structure and semantic distribution of meaning components is different in the two languages. By contrast, a translation mismatch implies a difference in information content between the source and target sentence’

Although translation divergences are common in a successful translation system, translation mismatches should ideally not appear. In this chapter, I would like to introduce some translation divergences, that faced me when translating the weather reports between German and English. When there were no clear examples of a divergence in the weather data I used, I will use data outside my sub-domain to illustrate the divergence at hand. The divergences I will highlight during the course of this chapter are categorial divergences, lexical gaps, structural divergences, collocation divergences, lexicalization, multi-lexeme & idiomatic divergences and head switching.

**Categorial Divergences** occur when two semantically equivalent words are represented by different parts of speech in their respective languages. I did not come across an obvious categorial divergence in the context of weather reports, but I will use an example outside my domain to illustrate. In English, there is to the best of my knowledge no verb, which simply means to watch television. Perhaps ‘to flick’, but I think this means more to go through the channels looking for something interesting. However in
German there is a separable verb which means to watch television and no accusative noun phrase accompanies it. The verb is ‘fernsehen’ - to watch television.

‘Ich sehe fern’
I watch television

Please note here that fern does not correspond to the English word television, fern is just a separable prefix and can only be used in this context with the verb sehen. Without the word sehen preceding it, ”fern” would mean distant or far-off, Collins German Dictionary (2001). So as you can see the English noun television is being substituted in German by a verb.

**Lexical Gaps** occur when one single-word concept can only be fully represented by a phrase in the other language. This occurs quite frequently in German, where a number of nouns can be joined together to form a compound noun. This does not occur as often in English as it does in German. An example of this is:

‘Schönwetterperiode’
a spell of good weather

**Structural Divergences** arise when the valency requirements for semantically inter-translatable verbs differ. An example of this,

‘Liverpool spielte gegen Bayern’
Liverpool played against Bayern
Liverpool played Bayern

Notice that here, the English sentences are grammatical both with and without the preposition ‘against’, however in contrast the German verb must have a prepositional phrase headed by gegen in order for the sentence to be grammatical.
Another structural divergence that is common in translation involving German is the **separable prefix**. Some German verbs must be accompanied by a separable prefix, which goes at the very end of the clause headed by its verb in the present tense.

'Es bringt kalte Luft mit'

*it brings cold air with*

*it brings cold air*

When a separable verb is not used in its finite form, but instead in the future tense with the auxilliary *werden*, the prefix and verb are fused together. Here is an example,

'Es wird kalte Luft **mitbringen**'

*it will cold air bring with*

*it will bring cold air*

**Collocation Divergences** are the product of default translations being unsuccessful (i.e. in some situations a successful translation can result from a word by word lexical substitution from one language to another, however this is often not the case).

'Wolken ziehen sich aus dem Westen'

*Clouds move themselves from the west*

*Clouds are moving from the west*

Inclusion of a **reflexive pronoun** is quite common in the German language and this usually does not have a corresponding English equivalent as reflexives are not usually included on the English side. I have included reflexive pronouns as a member of the valency list for verbs requiring them in my German grammar (see section 6.3.3 for more information on how reflexives are handled in the system).
**Lexicalization**, Talmy (1985)\(^\text{17}\) describes as the transfer that must happen with ‘languages which distribute semantic content differently within a sentence’ Trujillo (1999) pg 126. An example that springs to mind is,

> 'Ich fahre mit dem Fahrrad'
>
> *I travel with the bicycle*
>
> *I cycle*

In German, you have to say you travel with the bike, there is no standalone verb equivalent to *cycle* in German. Here we can see that in some cases German is more complex, however remember that German simplifies some complex English phrases by using compound nouns:

> 'Areas of high pressure'
>
> *Hochdruckgebiete*

**Head Switching** arises when the syntactic head of an expression in one language is not translated as the syntactic head of the expression in the target language. Instead it may be translated as a modifier, a complement or some other constituent. I did not come across divergences of this kind in the weather data but here is a example of a head switching divergence between English and German.

> 'Ich schwimme gern'
>
> *I swim with pleasure*
>
> *I like swimming*

In the English sentence like is the syntactic head of the verb group. However in the equivalent German sentence ”schwimmen” — to swim is the syntactic head and it is modified by the adverbial ”gern”.

**Multi-lexeme and idiomatic divergences** crop up when an idiom of one language is translated into an idiom of another language, without there being a clear translation relationship between the two. One idiom is substituted for another if you like. These sort of translations are usually stored in translation memory, where entire idioms of one language are mapped to the corresponding idiom of the other language. This is necessary due to the lack of a relationship between the word pairs of the phrases. Here are two fine examples of this:

> 'Es regnet in Strömen’
>
> *it rains in streams*
>
> *it’s raining cats and dogs*

\(^{17}\)cited by Trujillo (1999)
'Das Hemd ist näher als der Rock'
the shirt is nearer than the shirt

'Charity begins at home'

Note how *it rains* can also be *it’s raining* in English. This luxury is not available to German speakers, since the -ing form of a verb can only be used during nominalisation of verbs

'gefrierender Frost’  ‘strömender Regen’

*freezing fog*  *pouring rain*

An important point to note during the translation of idioms is that substitution is made, not on the basis of the linguistic elements in the phrase, nor on the basis of a corresponding or similar image contained in the phrase, but on the function of the idiom. Dagut(1999)\(^{18}\) remarks that the problems of translating metaphor are interesting when applied to the problem of tackling idioms:

‘Since a metaphor or idiom in the source language is, by definition, a new piece of performance, a semantic novelty, it can clearly have no existing ‘equivalence’ in the target language: what is unique can have no counterpart’, Dagut (1999)

### 3.8.2 Weakly translated Words

‘The fundamental error of the translator is that he stabilizes the state in which his own language happens to find itself instead of allowing his language to be powerfully jolted by the foreign language’, Rudolf Pannwitz (trans. Richard Sie Burth). Venuti (1995)

Words often occur in the source language, which do not have a clear equivalent in the target language. During initial analysis of German and English texts it became apparent that there are words in German weather reports which cannot be translated into English, while also retaining their precise meaning in German. In the introduction, it was said that no translation is perfect, the following are examples of this. They are attemptx translations of German words into English, however they loose some of their German flavour in transfer.

- German adverbs of direction, including ‘hin’ and ‘her’ are used to express direction away from or towards the speaker in a more systematic and consistent way than is possible in English. ‘Hin’ indicates that something is moving away from the

\(^{18}\)cited by Venuti (1995)
speaker, while ‘her’ denotes motion towards the speaker. Adverbials like these are not usually included in English, so it is probably better to leave them out, the translation from German to English would then lose this truly German style, if these words were missing in an English to German translation, the resulting translation would not sound very fluent. It would be easily recognised as a translation. However, translation from English to German would sound more fluent, if these words were excluded.

- In English, we can say ‘some snow’ and this is how ‘etwas Schnee’ is translated in this system. ‘etwas’ can only be used with mass nouns in German, such as ‘Schnee’ - snow, or ‘Regen’ - rain. The word ”some” appeared to be the best translation for ‘etwas’. However, ‘etwas Schnee’ is a typical style used by Germans and its true meaning is more than some snow, it is a considerable amount of snow, but this translation would sound awkward.

- The noun ‘Wolkenfelder’ is used repeatedly in German weather reports. This literally means field/area of clouds. Once again the best solution available to me, this was patches of cloud, although I don’t think it entirely covers the concept. I think it implies that the sky is covered in clouds but there are some really thick patches, some thin patches etc... it is more a contrast of the density of cloud in the than the lack of cloud in places.

There were other cases like the three mentioned above, but these were the most prominent.

3.9 Conclusion

In this chapter linguistic differences and similarities between German and English have been listed. Some light has been shed on areas of German morphology and a brief description is given of how morphology is dealt with by linguistics today, along with some suggestions on improving the area of morphology in this system. The sub-domain of the system has been presented and some assumptions of weather data have been outlined. Last but not least structural divergences which occur between German and English, including some which this translator has dealt with are mentioned.
Chapter 4

Natural Language Processing Technologies

4.1 Introduction

The German grammar of this system is written using DCGs. Here I would like to take the opportunity to explain what this means, in case anyone has forgotten what a DCG is. I will also mention why I decided to employ this representation to write my grammar. I will also provide a quick reminder of what a chart parser is. Anyone who does not need reminding is free to skip this chapter.

4.2 Definite Clause Grammars

Definite Clause Grammar (DCG) interpreter is a Prolog preprocessor that takes DCG grammar rules, and adds difference lists to the goals. DCG clauses are really syntactic sugar, i.e they provide a nice simple user-friendly way to define grammars, however when the rules are consulted by Prolog the DCG rules automatically become Prolog clauses, in terms of difference lists. DCGs allow context free grammars to be written very simply in DCG clauses. DCGs provide a top-down, depth-first, search just like the Prolog search strategy itself. A DCG parser can then analyse a string of words and decide if the sentence is grammatical based on the defined grammar rules. This spares the user from having to deal with difference lists and therefore reduces the possibility of errors associated with the representation of the string span licensed by any rule. When using DCGs, we can add as many features to words and categories as we like and Prolog deals with them for us. This is an especially useful characteristic since natural language does contain many features such as number, person etc.. The use of DCGs allow us to force agreement between adjectives and nouns in highly inflectional languages and control subject verb agreement in English and German without difficulty. The extra argument features are simply passed up the tree by ordinary unification and depending on whether
it can correctly unify or not the sentence is rendered grammatical or not. This feature
controls the facts of German case and agreement neatly and simply. The feature passing
mechanism of DCGs makes it possible to give a surprisingly straightforward account of
long distance dependencies, this is very relevant when dealing with complex features such
as gap threading. Threading in turn allows us to represent long distance relationships
easily, such as those in relative clauses but for my purposes this is not so relevant. I
chose to use the DCG notation to define my grammar for a number of reasons:

- DCGs are very declarative and they provide a straightforward way of describing
  legal sentences of a language as well as ensuring that certain configurations of
  features and categories are licensed.

- Agreement features can be implemented with ease.

- The fundamental reason why I chose to use a DCG grammar is because the process
  of analysis is reversible. This meant that I could use the DCG based German
  grammar both for analysis and generation.

- Data structures could be added as an extra feature easily and this was vital for
  translation by transfer, as I needed to include agreement as well as tree structures
  as features.

However they do have an associated disadvantages, these are

- DCGs maintain linear ordering of words. Words in a sentence, which are accepted
  by the grammar must appear in the same order as they did in the rules of the
  grammar. This restricts describing languages with relatively free word ordering
  without having to add additional rules for each possible ordering.

- DCGs have no limits on the number of features that can be assigned to a word.
  So grammars can become aesthetically messy very quickly.

- They can not handle left recursion (i.e they cannot include a left recursive grammar
  rule, such that the head of the rule is identical to the item in the left corner of the
  body) as this kind of rule will continuously call itself and will loop infinitely. Here
  is an example of a left recursive grammar rule:

\[
np \rightarrow np, \text{prepositionalphrase}
\]

4.3 Using a Chart Parser

I used a bottom up chart parser to analyse the English input, here I will describe briefly
what a chart parser is. This section is included as a recap for somebody thinking of
developing this project further, although I’m sure none of you have forgotten what it is.
A chart is a data structure that is adopted for parsing. It is more efficient than most other parsers used in ATN (augmented transition network) systems, this is because it eliminates backtracking. A chart parser checks an input string against it’s grammar rules and it stores all successful subconstituents and the positions where the subconstituent starts and then finishes. This ensures that once the chart parser has found a successful subconstituent, it records it and therefore will never need to waste time checking for the rule at this position again. A successful subconstituent is known as an edge. Edges can be active or inactive. An inactive edge is licensed by a grammar rule where all of the right hand side has been found and therefore the rule is saturated and characterised by the left hand side of the licensing grammar rule. An active edge is a rule where only part of it has be satisfied and it needs to absorb other categories to become complete. A chart parser finds edges and successful categories, but it also stores rules and records how successful they have been so far in a parse. So for example, if you have a rule

\[ S \rightarrow \text{NP VP} \]

and you have found a successful inactive NP edge

\[ \text{NP} \rightarrow \text{Det N}. \] (this is an inactive rule)

then the S rule would look like this:

\[ S \rightarrow \text{NP. VP} \] (this is an active rule)

this means that an NP has been found but not an S, since we still need the remaining categories following the dot. Gazdar and Mellish (1994), pg 189 describe a chart parser as:

‘A chart is basically a data structure in which the parser records its successful attempts to parse subconstituents of the string of words’

Why should I use a chart parser? Each time a successful edge is found, the chart parser checks the search space to see if the edge is already recorded, if not is adds it to the chart otherwise it does nothing, this means that it is very cost efficient. Here is an example of what a chart looks likes, which spans over the sentence ‘the man saw the girl with the telescope’.

4.4 Conclusion

This chapter has provided descriptions of definite clause grammars and chart parsers. I also explained why I chose to use both of these natural language processing tools.
Figure 4.1: Sample Chart
Chapter 5

My Design

5.1 Introduction

In this chapter, the methods and algorithms of this system which are used when translating from German to English and vice versa will be outlined. Following that two of the fundamental components of the system will be described. These are the tagger and the chart parser. Prolog is the language which was used to implement the system. Later on in this chapter, I will take the opportunity to explain why I chose to use Prolog. At the end of the chapter a successful weather report translator Météo will be presented which has been actively functioning since 1976. The aim in doing so is to give the reader some insight into differences which occur between Météo’s approach and mine.

5.2 System Outline

This translator is bidirectional, i.e. it can translate from English to German and from German to English. The system can be used either through a Prolog shell or through an interface, both will be described in chapter 8. As can be seen in Fig 5.1, the system can take one of two paths depending on the source language. If the source language is German the path beginning with GER INPUT will be taken otherwise if starting with English as the source language the path begins with ENG INPUT. During the first half of this chapter, the translation process beginning at either path will be outlined. Diagram 5.1 is broken down into two diagrams 5.2 and 5.3 to illustrate each of the possible translation paths.

5.2.1 Translation from English to German

The translation process from English to German will be outlined here. In the diagram 5.1 each stage is given a name. These stages will be referred to throughout this section in italics. Initially, the user provides some English input. This input must then be tagged by the tagger during the TAGGER stage. This is because the chart parser requires
Figure 5.1: Architecture of Entire System
Figure 5.2: Translation Procedure from English to German
CHAPTER 5. MY DESIGN

CHAPTER 5. MY DESIGN

tagged input. Details of the tagging process are described in section 5.3.1. After the input has been processed by the tagger a list of tags are returned. These tags are passed into the chart parser and the translation process moves to the **CHART PARSER** stage. The chart parser will take the tagged input and produce a chart of the input. Information on the chart parser can be found in section 5.3.2. After the **CHART PARSER** stage a chart will have been produced. This chart contains information on which categories are contained in the input as well as information on exactly where these categories occur in the input string. A method `parse_output` is called on the chart, this method will extract the left most longest edge which spans the input, i.e. this method will extract the tree of the left most longest edge contained in the input, for example it would chose a sentence tree over a noun phrase tree if it could find both (presuming that the sentence tree would be longer). After the **PARSE OUTPUT** stage, a tree or a list of trees will be returned (if one tree includes all of the words in the input, it will be returned otherwise a list of trees which together span the entire input will be returned).

As this system is transfer based the next stage, namely **ENGLISH TRANSFER MODULE** is crucial. It is very important that the English tree/trees carry all the necessary information needed to find the equivalent German Tree. This was an important consideration when deciding on the tree representation ontology, see section 6.3.1. The English tree/trees are passed into the English to German translation module. This module consists of equivalence relations, which map English tree structures to German tree structures. The equivalence relations try to unify the first argument of the equivalence rule with the English tree and then output the corresponding German tree as the second argument. These equivalence relations break down an English tree into its subcomponent trees and then all subcomponent trees are mapped to their equivalent German counterparts. Following this, all of the subtrees are put back together and the entire German tree is returned.

Once the transfer module has found an equivalent German tree, the German DCG grammar is used in reverse to generate the words from the tree. This was one of the benefits of using a DCG grammar. It is able to generate a tree given words as input and also has the ability to produce words when given a tree. This happens in the **GERMAN GRAMMAR** stage. The tree which is found as the equivalent German tree only includes the base forms of verbs, adjectives, and nouns etc.. However, when the tree is given to the grammar all verbs, nouns and adjectives are returned fully inflected, i.e. the equivalence trees do not contain morphological information instead when a German tree is passed into the German grammar the grammar produces the inflected words. This German to English translation process can also be clearly seen in Figure 5.3.
Figure 5.3: Translation Procedure from German to English
5.2.2 Translation from German to English

The underlying algorithm of translation from German to English is more straightforward than the process of translating from English to German. I will refer to diagram 5.3 when describing the German to English translation process. The user provides a German sentence. This sentence is then passed into the German DCG grammar to be analysed. The input must be grammatical, including correctly inflected words etc ... in order for the grammar to be able to parse it. If the grammar can parse the input it will return the tree representation of the input. This is better known as the analysis stage. The input can be a sentence (only including verbs and the items on the main verb’s comp list), noun phrases, prepositional phrases, time adverbials and spatial adverbials. If a German tree has been found at this point, it will be passed on to the equivalence relations which map German trees to their corresponding English trees. These German to English equivalence rules will be described later on in detail in chapter 7. If the German grammar could not parse the input, then the translation process will automatically fail. After a tree has been found which represents the input sentence, it is passed into the GERMAN MODULE. This module contains equivalence relations which map German trees to their corresponding English trees. If a corresponding English tree can be found then the transfer stage was successful. After an English tree has been found the convert_tree_to_words method is called. This method takes a English tree and extracts the words from it. The equivalent English trees which are found by the equivalence relations do not contain morphological information apart from in the verb part, during the transfer process a method called find_feature is used which extracts the number and person features from the German tree and this information is then put into the English tree. This ensures that when the English verb is extracted from the English tree using the convert_tree_to_words method it will be correctly inflected. All the words, which are extracted from the English tree are returned as the translated sentence. As shown in Figure 5.2, the translation process from German to English can prematurely end (the translation process may successfully complete some of the stages but not all). A point where the process can premature end is,

- If a German tree is found, which represents the German input, but the transfer rules cannot find an equivalent English tree

In this case, I would like the analysed tree to be printed to the screen instead of just reporting that translation was unsuccessful. I feel that at least by doing so, shows that the input was at the very least processed in some way and it can also give some insight as to where the translation process ended. Perhaps then the problem could be found and fixed.
5.3 Resources used

1. Tagger
2. Chart Parser
3. Prolog
4. Java

My Application made use of a pre-existing Tagger and Chart Parser. Further information on the chart parser can be found here:

/users/Public/CSLL/4thYrProjects/Software/Parser/Doc/parser_doc/parser_doc.html

Here is a link to further information on the tagger:

/users/Public/CSLL/4thYrProjects/Software/Thought/Docs/

5.3.1 Tagger

The Chart parser, which is used to analyse English input sentences requires tagged input. Raw English input (i.e. words) cannot be put straight into the chart parser. First each word of the input must be tagged and then this tagged input can be given to the chart parser to parse. There are various methods for tagging the English input, these are:

1. you can produce the input using the tagger
2. you can tag the words by hand
3. you can correct what the tagger produces

Initially, I used the tagger interface to manually tag input. I then stored these pre-tagged words as well as the sentences/phrases they represented and then used the tags as input to the chart parser. This was time consuming, as every phrase I wanted to parse had to be pre-tagged and stored in a file. Using a combination of 2 and 3 above was however, very helpful, during the development of the system for testing the parser and then later the translator. Later on I began using a predicate called `dotagging`, this predicate is stored into the saved state the parser. It takes a string of English words as input and will return a tag for each word of the input, based on what it decides is the most likely Tag. I could then use these tags directly as input to the chart parser. This made it easier to tag phrases and therefore, I was able to test sentences and their output quicker. I found this tagger very useful and it usually tagged words correctly, based on their context in the weather reports. However, there were a few problematic sentences. Although not very frequently, sometimes the tagger found alternatives tags to the ones expected for
individual words and this in turn made the translation impossible. Here is an example of a weather sentence which was incorrectly tagged using the dotagging method. These tags were then passed to the chart parser, which produced the following type of parse for the sentence ‘the weather changes’:

```
np
  det n n
  the weather changes
```

When the actual parse, which was wanted in this context was,

```
s
  np vp
  det n v
  the weather changes
```

This translation system makes use of the dotagging predicate. The tagger and chart parser together came up with the first tree above, the translator was unable to translate the sentence correctly. However, I then used the tagger interface to tag the input sentence. The tagger found all possible tags and I then removed the noun tag for the word changes, in order to force the parser to parse the word changes as a verb. In doing so an alternative parse was found for the input and translation was successful. Perhaps, it would be better to supply the parser with all possible tags for each word of the input using an alternative dotagging predicate. Supplying more than one tag for each word would help improve the finding of correct parses in sentences. In the example above, when the chart parser is given all possible tags for the input the correct sentence is found. I will discuss this later as a possible method of evaluation.

Although initially pre-tagging the input was time-consuming and would be an unrealistic task when pre-tagging all words of the weather data lexicon, it allowed me to have more control over assigning words the correct tags and therefore, due to lack of disambiguation or incorrect tags being assigned to words, I was able to assign otherwise ambiguous words the correct tags. When producing English trees to represent the English input, it was more efficient to allow the tagger disambiguate between tags itself for words and this for the most part was very efficient, although naturally some errors did occur. The tagger couldn’t handle the word ‘Sunday’ for some strange reason because it always tagged ‘Sunday’ as an adjective, even though it had no problem recognising other days of the week as nouns! For the sentence ‘I will go on Sunday’, here is the tagged output that the tagger would produce:
Perhaps if I was to extend this weather translation project I could add some additional sub-domain specific rules to the tagger, to help it chose correct tags for words in weather data reports. The insertion of punctuation made it easier for the tagger to disambiguate the elements correctly. This was especially important when tagging telegraphic style phrases, for example, 'sunny spells dry in the east' produced the following tags:

```
tag('sunny', [cat('A', 'sunny', ['b'])]).
tag('spells', [cat('N', 'spell', ['m'])]).
tag('dry', [cat('V', 'dry', ['G1e', 'G2e', 'G1m', 'G2m', 'G3m'])]).
tag('in', [cat('PREP', 'in')]).
tag('the', [cat('DET', 'the')]).
tag('east', [cat('N', 'east', ['e'])]).
```

while 'sunny spells, dry in the east' produced tags like this:

```
tag('sunny', [cat('A', 'sunny', ['b'])]).
tag('spells', [cat('N', 'spell', ['m'])]).
tag(',', [cat('pun', ',')]).
tag('dry', [cat('A', 'dry', ['b'])]).
tag('in', [cat('PREP', 'in')]).
tag('the', [cat('DET', 'the')]).
tag('east', [cat('N', 'east', ['e'])]).
```

Here you can see that with the comma the tagger found dry to be an adjective, however without the comma is found dry to be a verb. Although in grammatically correct English the comma is necessary, when a user types input into the the translator, sometimes they leave out these commas, and as I have shown this would produce an incorrect tree and as a result the translator would be unable to find the correct translation. This meant that without commas, it was impossible to predict the parses for English telegraphic reports.

### 5.3.2 Chart Parser

The chart parser can take tagged input, with either one tag per word or with multiple tags per word. It will parse the input and return a chart which contains a list of possible parses which span the input. The chart includes categories which occur in the input and the locations of where these categories occur within the input. One category (perhaps a sentence) might span the entire input or alternatively when there is no category which
alone can span the input, a number of categories may span across it. The chart parser
takes in some tagged input and matches it against its grammar rules, the chart parser will
try to find all possible parses within the input and since it never needs to satisfy an edge
twice, it is much more efficient than the DCG approach. Using this parser meant that I
could immediately parse English sentences and produce tree structures to represent them.

The English tree structures produced by the chart parser are very conventional. The
left to right word order of words in the sentence is maintained within the tree. The verb
information part of the trees produced by the chart parser were broken up into a right
hand side and left hand side to store the complements of the sentence. The left hand
side usually remains empty, while the right hand side incorporates the subcategories that
the main verb requires. However, sometimes the left hand side of the tree includes verb
information, when there are more than two verbs in a sentence. Here is an example of
how an English tree looks using the sentence the weather will be sunny:

```
  s
   
  np
   
  det n
  |    |     
the   weather
    
  vp(3,sing,pres)
   
  lhs             rhs
lexeme([will],(157,F))       vp(inf)
   
  lhs
lexeme([be],(111,F))
   
  rhs
be
    
   ap
   
asunny
```

Notice here, that the word order of the sentence is exactly the same as the order of the
words in the tree.
The grammar supplied with the chart parser can not recognize relative clauses or passives. This was not problematic for my purposes, since initially I decided to exclude sentences and clauses using the passive or relative clauses from my translator and German grammar as they are seldom used in weather reports. So I decided it would not be worthwhile to include them in the translator. The chart parser worked well and gave expected parses for the majority of grammatical sentences. Using this chart parser saved me having to write an English grammar just like the German grammar that I wrote. I decided this was more beneficial for several reasons. Firstly the English chart parser can recognize all English morphologically inflected verbs and nouns. My German grammar as mentioned before, only recognizes the 3rd person singular and 3rd person plural form of verbs and only plural and single forms of a subset of English nouns and verbs relating to weather. The English chart parser in contrast to the German grammar can recognize a much larger subset of data. The chart parser is very efficient as it does not need to backtrack and recheck for an edge every time it is asked to parse two sentences, sharing the same phrase, as mentioned before it stores the starting and finishing position of a phrase contained in a string, as well as storing what category occurs between the two points. Also since backtracking was unnecessary the chart parser was time efficient and faster than my German Grammar although its domain was much larger. The chart parser can recognize sentences with three or more verbs, such as 'it will continue to be sunny’ while my German grammar can only process sentences with one or two verbs in German. The pitfall with using this chart parser and its fixed grammar for English was that sentences can only consist of the verb group and complements that the verb subcategorizes for. No extra prepositional phrases, time adverbials or spatial information can be included. When the chart parser is given the sentence. 'On Monday it will remain sunny for the entire day’ it recognizes three separate trees inside this phrase these are:

1. 'On Monday’

   pp(npf(\text{eat},0))
   \hspace{1cm}
   p
   \hspace{1cm}
   np(npf(sing,0))
   \hspace{1cm}
   on
   \hspace{1cm}
   n(npf(sing,0))
   \hspace{1cm}
   \text{monday}

2. 'it will remain sunny’
this is both advantageous and disadvantageous for the following reasons:

1. When the chart parser cannot find a parse for the entire sentence in the input, it will not automatically fail, instead it offers a number of trees representing individual parts of the trees.

2. Each parse tree of the sentence will be translated individually, if translation is possible.
3. When individual trees are translated the words cannot simply all be put back together like on the English side to produce a grammatical sentences in German. Normal German word order constraints must be upheld. So when the individual subtrees of an English phrase are translated into German each tree is given back as an individual translation. The semantic information which was contained in the original sentence is still returned, perhaps however in small grammatical phrases.

4. I could have written a transformational Grammar to collect all of the partial translation and then restructure the words to comply with German word ordering constraints. This would have been beneficial because of the nature of weather reports, particularly when telegraphic style speech is used. However I did not have enough time at my disposal to try this out. Telegraphic style also occurs frequently in German weather reports, in these cases a transformational grammar would be useless since I would like telegraphic speech in one language to be translated to telegraphic speech of another language.

5. I therefore chose to translate all components separately. When a sentence is given as input, it is broken down by the parser and each part is then listed along with its category. Each individual piece is translated on its own and the corresponding translations are output. This means that the translator will give back a full translation of all the parts individually, as separate chunks, and they are not all put together as one string. If you like the parser will still give back all the relevant information of the input sentence, but it will give it back in short phrases and sentences and the semantic information will still be maintained between the target and source languages.

5.3.3 Why use Prolog?

Prolog was first introduced around 1970. The pre-defined DCG Parser is very attractive when writing a grammar, as it is clear and concise. This is not the only reason I used Prolog, but it definitely had a strong influence. In Prolog, it is easy to describe language and the relationships which hold between individual words. Prolog is also a very useful language in applications of natural language processing, it is an expressive high level language and can be used when the representation of abstract trees are needed. This characteristic was essential in my application as all sentences and phrases had to be represented as trees for the transfer stage to take place.

‘Prolog allows us to specify complex structures concisely in terms of abstract patterns’, Gadzar and Mellish (1989), pg 18

Recursion plays a fundamental role in both prolog and NLP, this is essential to allow predicate definitions call themselves. This feature was very important when I translated
from English to German using the predicate translatedcg. This predicate took a list of inactive English edges found in the English input, each edge was represented by a tree structure that was to be translated and all edges were stored in a list. Each element of the list was translated individually by one of the translatedcg predicates. There is a translatedcg predicate to translate NP’s, another for PPs etc... After one tree was translated the rest of the list was recursively passed back into the predicate to be translated. This feature of recursion in a situation like this is vital. As there were numerous equivalence rules to map an English tree structure to a German one, the source tree had to be given as input to each equivalence rule until it found the suitable one, otherwise it failed. This required all possible sentence equivalence rules to have the same name, so that they could all be checked before Prolog would give up and fail. This is another Prolog trait that is essential in NLP and that is not available in other commercial programming languages such as Java is that several mutually exclusive predicates can share the same name and failure to succeed in one, results in backtracking before Prolog tries the next alternative.

5.4 An Alternative Approach

Météo is a system which was installed in 1976. It translates weather bulletins from English into French. It was developed by the TAUM group in Montreal. Météos success is achieved by the restrict of its sub-language of weather forecasts. The need for a system like Météo arose in 1956 when the Canadian government released its bilingual policy. Since May 1977 Météo has been used daily to translate weather forecasts for the Canadian press and television networks. Météo is a mixture of a direct and a transfer system. The restriction to the idiosyncratic nature of meteorological bulletins and the limitation to English and French translation in one direction - English arrow French lead to Météo’s unique approach. The analysis stage of Météo lacks a morphological component. Initially a dictionary look up is used, which deals with any problems of collocations, ’idiomatic’ or ’compound expressions’. This is the most typical feature of a direct system. (i.e. Météo carries out lexical transfer before any syntactic analysis). Here I will list some of Météos characteristics:

- Météos sub-domain is highly restricted
- Météos input is pre-edited to a suitable form. Anything in the reports, which Météo is incapable of translating is sent to human translators.
- Météos pre-edited text is formatted by a system called Q-System, which eliminates punctuation and regularised the text (replaces inflected words with their base forms).
• Météo initially uses three bilingual dictionaries, each containing one of the following:
  1. Place Names
  2. Idioms
  3. General Meteorological Vocabulary

• Syntactic analysis occurs, but only for five basic phrase types.

Here I will shown what types of phrases Météo can analyse syntactically. I think it is very interesting that Météo can be so successful and famous, although it doesn’t contain many equivalence rules and it is mainly based on the direct approach. Here are the five basis structure types processed by Météo.

1. Places names, such as RED RIVER, INTERLAKE etc...

2. The next three structures are captured by rule formalisms. They all represent structures typical in weather reports, Here I will list the rule for each and a sample of a sentence using the rule. Here is a key to what the symbols in the rules mean:

The first rule is called MET1.

\[
\text{MET1 rule} \\
(C(adj/GN, [Cmod]), [T], [L])
\]

\[
\text{MET2 rule} \\
(GN(highs/lows/etc..), [T], [L], GN(Temp to Temp/Temp)[T], [L])
\]

\[
\text{MET3 rule} \\
(GN(outlook for T,C(adj/GN, [Cmod]), [T], [L])
\]

3. The last type of translatable material is the systems name, the time, date and a list of the days which have been forecasted for in the report.
Figure 5.5: Météo rule - MET1

Figure 5.6: Météo rule - MET2

Figure 5.7: Météo rule - MET3
One a report has been analysed, it is broken up into its constituent parts and its three dictionaries are put into use. As you can see here Météo orientates itself around telegraphic chunks, it does not translate sentences, which include verbs and their complements. However it efficiently translates telegraphic style weather reports. This system and my system vary considerably both in their approach and the sub-domain of weather data, which they aim to translate. This just goes to show that although the sub-domain of weather reports is restricted to a small subset of natural language, it still possesses a wide scope of language, within itself. This system and my system are two systems based on the same sub-domain, but it would seem that they are entirely unique from each other.

5.5 Conclusion

This section has outlined the basic algorithms of both German to English translation and vice versa. It has allowed the reader to understand how translation is carried out. It also introduced the reader to the components used in the translation process, i.e. the chart parser, tagger, grammars and equivalence relations. Many methods and features which were mentioned in this chapter will be elaborated on in the following chapter. In the chart parser section the English tree structures which are produced by the German grammar were described. In the following chapters I will explain the analysis, transfer and generation processes in greater detail.
Chapter 6

The Analysis Stage

6.1 Introduction

‘a grammar for a language is a set of rules for specifying what sequences of words are acceptable as sentences of that language’, W.F Clocksin and C.S Mellish, (1994)

In this chapter the algorithms and techniques used to analyse the input sentences will be described. Analysis is carried out by the German and English grammars. I will not list the grammar rules of either the German nor English grammars as these can be clearly seen with a glance of the code. Instead an effort will be made to explain features of the grammar, which are not intuitive when looking at the code. The German grammar can be found here:

/users/Public/CSLL/4thYrProjects/Grace/GerGrammar/german_grammar.pl

English tree structures will not be described, as they have already been covered in section 5.3.2. However, German tree ontology is illustrated. Throughout this chapter, the algorithms used to ensure the complements of a German sentence match its valency list will be explored, as well as discussing the possible word order of these complements and the order of these complements in their respective trees. Initially, I will begin by explaining the valency feature of the English grammar.

6.2 Important Features of the English Grammar

Using a ready available chart parser allowed me to avail of its pre-written grammar. For this reason I did not directly use the English grammar independently. My only contact with it was that I had to process its output trees so all information in these trees could be passed into the corresponding German trees. The English grammar, unlike the German grammar is not reversible so a separate method is used to extract the words from the English trees. Further information on the English grammar can be seen here:
6.2.1 Subcategorisation Information for English verbs

In the English grammar each verb is assigned a subcategorisation number. This number is used to find the subcategorisation term for that verb. A verb’s subcategorisation number can be used in a method called `english : subcat_decode` to find the valency information for that verb. A verb’s subcategorisation term is built out of atomic categories. These atomic categories are of type *Kategory*, these *Kategory* types eventually make up the labels of edges produced by the chart parser. *Kategory* items can be one of two forms, these are:

1. **Non Terminal Categories** which look like this,

   `cat(f0,...,fn,Tree)  n>=0`

   `f0,...,fn` represent the associated features of the category. A category may contain no features and `Tree` is the syntax analysis of the category.

2. **Terminal Categories**

   `[i0,...,in]  n>=0`

   `i0,...,in` are lexical items. These may be words or tagged words.

Subcategorisation terms - Subcat - are defined as part of the following simultaneous inductive definition:

1. **Subcat.Atomic**: `A0, ...`
   - `cf(K)` - K is a non-terminal Kategory
   - `ts(T)` - T is a terminal Kategory
   - `atomoder(A0,A1)` - this is a disjunction of categories, based on context

2. **Subcat**: `Sub0, ...`
   - `cat(cf(K),lhs(Seq1),rhs(Seq2))`
   - `catoder(Sub0,Sub1)`

3. **Catseq**: `Seq0, ...
   - `seq([A0,...])`
An English verb’s subcategorisation term is needed during translation in both directions. Here I will discuss what happens in the English to German direction. First the English input sentence is tagged and parsed resulting in a tree representation of the input. Each verb in this tree is assigned a subcategorisation number. For example, take the verb *bring* in the sentence ”strong winds will bring low temperatures”, *bring* in this sentence has a valency number of 113. This valency number can be given as a parameter to the method called `subcat_english:decode(Valencynumber, ValencyList)` and the subcategorisation term for this verb will be returned.

```
*this takes a verb’s number 113 and finds its valency list*/ -?
subcat_english:decode(113,Valency_English).

Valency_English=cat(cf(vp(_A,_B)),lhs(seq([])),rhs(seq([cf(np(_C,_D))])))
*the valency list for the verb is one noun phrase, this is the object of the sentence since the subject of a sentence is not included on the verbs valency list*/
```

Here is another example of finding a verb’s valency information. This verb’s valency list is empty. The reason for this is that the subject of the sentence does not form part of the verbs valency list. Example, sentence is ”snow flakes fall”, the verb fall has a valency number of 53,

```
*this takes a verb’s number 53 and looks for its valency list*/ -?
subcat_english:decode(53,Valency_English).

Valency_English=cat(cf(vp(_A,_B)),lhs(seq([])),rhs(seq([])))
*this verb’s valency list is empty*/
```

When the verb’s subcategorisation term has been found, the English verb must be mapped to its German equivalent. The subcategorisation term `Valency_English` must be converted into a more basic form in order to be used in the transfer module. In the transfer module there are equivalence rules which map an English verb to a German verb. The equivalence rule for the verb ”bring” looks like this:

```
/* the English verb bring has two noun phrases on its valency list and its equivalent German verb is the separable German verb mitbringen which subcategorises for a nominative noun phrase and an accusative noun phrase */
equiv_verb([bring],[np(_)],np(nom),np(acc)], mit).
```
Therefore the subcategorisation term must be broken down into transfer form which looks like this: \( np(\_), [np(\_\_, \_)] \). This takes place in the `subcat_flatten` method. After the subcategorisation term has been simplified the equivalent German verb can be found and the translation process can continue. `subcat_flatten` is reversible and can be used for translation in both directions, the order of the method is reversed going in opposite directions. The subcategorisation term consists of a left hand side and right hand side just as the English verb which it represents. This valency list is subdivided into the left hand side and right hand side and this method takes subparts from the verb’s right and left hand side of the subcategorisation term and creates a simpler version called `Flat_EngVal`. The opposite takes place in the other direction. Here is the code for `subcat_flatten`.

```prolog
subcat_flatten(Valency_English, Flat_EngVal):-
    Valency_English =
    cat(cf(A0), lhs(seq(CompsSeq1)), rhs(seq(CompsSeq1))),
    flatten_seq(CompsSeq2, Seq2F),
    flatten_seq(CompsSeq1, Seq1F),
    Seq1F = [], /* LHS usually empty */
    Seq2F = Flat_EngVal.
```

The method `flatten_seq` just converts the complex subcategory entries into simple ones and verse versa like so...

```prolog
flatten_seq([cf(Cat)|_Rest], [Cat|R]):-
    flatten_seq(Rest, R).

flatten_seq([ts([tag(Cat,_F)])|Rest], [Cat|R]):-
    flatten_seq(Rest, R).

flatten_seq([ts([tag(from,_F)])|Rest], [from|R]):-
    flatten_seq(Rest, R).
```

etc ...

In the opposite direction, when translating from German to English, the English equivalent of the German verb is found using the `equiv_verb` method. This time the English verb’s subcategorisation information is in transfer form (simplified form) and it must be converted back into a subcategorisation term. The reason the transfer form has to be converted to into a subcategorisation term is so that the English grammar form can be given to the `subcat_decode` method and the valency number corresponding to the valency list can be returned. This number can then be entered into the equivalent English tree.
6.3 Important Features of the German Grammar

6.3.1 German Trees

Both English trees and German trees use a similar ontology with categories being represented as:

\[
f(\text{mother} \& \text{daughterlist}) \text{ with } f(\text{leaf} \& []) \text{ for leaves}
\]

Although German and English trees use the same design to indicate the mother and daughters of categories. The word order and verb representation of the sentences they describe vary considerably. German trees unlike the English trees are quite abstract. Here is the basic tree structure that I chose to represent a German sentence,

\[
f(s(\text{Valency_German}) \& [\text{Verb_German} \mid \text{Comps_German}])
\]

Valency_German is the valency list of the main verb in the sentence, i.e. the finite verb in sentences containing one verb and the infinitive in sentences containing two verbs. Verb_German is the verb group contained in the sentence, this can be either one or two verbs. Finally Comps_German is the list of complements which the main verb subcategories for. Comps_German contains all of the trees of the individual complements, including the subject object etc... while Valency_German only contains the valency list of category names which are listed in the valency rules, for example [np(nom), np(acc)]. An example instantiation of this general tree is,

'Schnee fällt'

Snow falls

Snow will fall

The tree for this sentence looks like this,

\[
f(s([\text{np(nom)}]) \& [f(3) \& [f(\text{fallen} \& [])]])\), f(\text{np(nom)} \& [f(n\&[f(\text{Schnee}' \& [])]])])}
\]

and when drawn, it looks like this:

\[
\begin{align*}
s([\text{np(nom)}]) \\
\text{v(3) np(nom)} \\
\text{fallen n} \\
\text{Schnee}
\end{align*}
\]

I decided on this ontology to represent my German trees for a number of reasons, which I will outline here:
1. Valency. German is contained at the beginning of the tree. The reason for this is so that the German grammar could be later expanded to incorporate multiple verb groups. In my German grammar only sentences with a maximum of two verbs in the verb group are allowed. English sentences in the future tense are always made up of the verb 'will' and an infinitive. In this situation, only the infinitive verb subcategorises for the complements on it’s valency list. The auxiliary verb subcategories for the infinitive and its complements but nothing additional. However in the future somebody may want to allow the grammar to recognise more than two verbs in a group and here it would be possible that more than one verb could subcategorise for complements. On this basis, I decided that it would be quite useful for the complements of all the verbs in the group to be stored in a list at the start.

2. After listing the complements, I then list the verbs of the verb group. This can at present include either a finite verb like in the tree for "Schnee fällt" above or an auxiliary verb and an infinitive like the sentence below.

   'Schnee wird fallen'
   Snow will fall

Basic tree structure:

\[
\text{f(s([np(nom)])&[f(aux&[f(werden&[])])],f(inf([np(nom)])&[f(fallen&[])]),f(np(nom)&[f(n&[f('Schnee'&[])])]))}
\]

which is represented by this tree:

```
  s([np(nom)])
     /       \\
    aux       inf([np(nom)])
     |                |  np(nom)
    werden    fallen    n
             |                |
                      |    Schnee
```

### 6.3.2 Dealing with German Future Tense Indicators

The future tense in German can be indicated in two different ways:

1. A verb in the future tense (present tense form of werden + infinitive)

2. A sentence in the present tense + future time adverbial

The chart parser will not recognize a time adverbial in the English sentence as being part of the sentence and this means that although a time adverbial may be present, it
will not be included in the English tree. Due to this, when an English sentence of the form: X will VERB Y, such as "X will reach Y" is given to be translated I decided to offer two translations. These are:

1. Firstly, the auxiliary "will" is ignored and a translation is made as if the sentence were "X reaches Y"

2. Secondly, a translation tree is made including a German auxilliary (werden) and the infinitive form of the verb.

The mapping which occurs is:

<table>
<thead>
<tr>
<th>English Tree</th>
<th>German Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>NP</td>
<td>Verbgroup</td>
</tr>
<tr>
<td>VP</td>
<td>Comps</td>
</tr>
<tr>
<td>L</td>
<td>NP</td>
</tr>
<tr>
<td>Aux</td>
<td>LHS</td>
</tr>
<tr>
<td>R</td>
<td>RHS</td>
</tr>
<tr>
<td>vp</td>
<td>V</td>
</tr>
<tr>
<td>LHS V RHS</td>
<td></td>
</tr>
</tbody>
</table>

The Verbgroup Part of the German Tree varies according to whether translation one or two above is taking place, i.e. finite verb or auxiliary and infinitive.

Here is an example, when the translator is given the English sentence ‘the temperatures will reach ten degrees’ as input, it provides two possible translations these are:

**die Temperaturen erreichen zehn Grad**

*the temperatures reach ten degrees*

(with a time adverbial this German sentence would be in the future tense)
6.3.3 Valency in German Grammar

The German grammar contains a list of valency facts. These facts contain a verb and its valency list, i.e. the list of subcategories which they require to be grammatical.

Complements which can occur on a verbs valency list are noun phrases, prepositional phrases, reflexive pronouns, adverbials, predicate adjectives - indepadyj and 'it' - snp, the subjectless noun phrase used in it’s raining. Here is an example,

valency(liegen, [np(nom), pp(p(bei))])

this rule states that the verb liegen - ‘to lie’ subcategorises for a nominative noun phrase and a prepositional phrase headed by the preposition bei - ‘at’. If a German verb is separable, then its prefix must also be added to the valency facts as a third argument, like this example using the separable verb mitbringen to bring, the ’mit’ prefix is removed from the verb and put to the end of the clause in the present tense.

valency(mitbringen, [np(nom), np(acc)], mit)

Some verbs can have multiple meanings and can for example, be transitive and intransitive in two different situations. In this case they have different valency lists in each situation. Therefore, the same verb name is allowed to subcategorise for different complements in varying contexts, for example liegen can also occur in the following contexts within the German grammar. Here liegen requires a nominative np and a pp headed by um — ”around”

valency(liegen, [np(nom), pp(p(um))])

In this case liegen requires a nominative np and a pp headed by zwischen — ”between”

valency(liegen, [np(nom), pp(p(zwischen))])

Valency is used in two ways in this system:

die Temperaturen werden zehn Grad erreichen
the Temperatures will ten degrees reach
the temperatures will reach ten degrees
1. Analysis in German

2. Generation in German

1. A single valency entry for a verb can license several complement orders. For example, consider the verb *geben* - to give, in the German grammar the valency list for give is:

\[
\text{valency}(\text{geben}, [\text{np(nom)}, \text{np(acc)}, \text{np(dat)}]).
\]

Given this valency list, the German grammar will accept both the sentences,

"er hat ihm \text{acc} ihm \text{dat} gegeben"
he had him him gave
he gave him to him

and

"er hat ihm \text{dat} ihn \text{acc} gegeben"
he had him him gave
he gave him him

Both of the above sentences are considered grammatical by the German grammar, even-though only the complements of the first sentence are in the same order as the complements on the verb’s valency rule. The reason both are allowed to be seen as grammatical is that because word order is free in German, I have allowed the order of the complements in the German sentence to be grammatical if they occur in any permutation of the valency list. This is carried out by the rules `subcat_agree` and `valency1`.

`subcat_agree` looks like this:

/*this method ensures that all the comps are legal complement of the verb*/
subcat_agree([Val|Vals],Trees) :-

/* take one of the complement trees of the sentence select(T,Trees,RestTrees),

/* try to match the tree to a member of the valency list */
subcat_match(Val,T),

/* check the rest of the complements against the valency list */
subcat_agree(Vals,RestTrees).

/* succeed all comps have been checked and the valency list is empty*/
subcat_agree([],[]).
All trees of the complements of the sentence are stored together in the German tree as mentioned in section 6.3.1 as well as the verbs valency list. \textit{subcat\_agree} takes the list of complement trees and the valency list as parameters. It then takes any of the complement trees and tries to match it against the first element of the valency list. It continues this process until there is nothing left on either list and the sentence is rendered grammatical otherwise the sentence is considered ungrammatical. The method used to check each complement tree with a member of the valency list is called \textit{subcat\_match}, this can be seen below.

\begin{verbatim}
/* for valency = np(nom) */
subcat_match(Cat,f(Cat & _)).

/* for valency = np(nom) */
subcat_match(rfxpro,f(rfxpro(_,_,_) & [Entry])).

/* for valency = pp(p(bis)) */
subcat_match(pp(p(Prep)),f(pp & [f(p & [f(Prep & [])]) | _])).

/* final case if valency item is a tree term */
subcat_match(Tree,Tree).
\end{verbatim}

An attempt is made to try and match the tree to the head of the valency list. I decided allow reflexive pronouns occur on the valency list for German verbs instead of allowing reflexives to fall under the noun phrase category. This simplified things when I was checking an English verbs valency list and a German verbs valency list. The German verb is allowed to contain a reflexive pronoun on its valency list while the English valency list remains empty.

\begin{verbatim}
equiv_comps([],[],f(rfxpro & Dtrs),rfxpro).
\end{verbatim}

An example of this kind of divergence can be seen in section 7.3. As mentioned before a verb’s valency information, as listed in the \textit{valency facts} is essential for two reasons.

Firstly, during analysis, when the grammar is parsing a verb, it must find the inflected form of the verb in the grammar and it must then look up the valency list associated with this verb. Here is a lexical entry for the verb form ”verbreitet” found in the German grammar:

\begin{verbatim}
v(V,Valency,3) --> {V = f(v(3) & [f(verbieten & [])])},
\end{verbatim}
The valency list of the verb is instantiated and carried with verb so that subsequent complements of the sentence including this verb can be checked against this valency list to check if the sentence is grammatical. Here is the code for valency1 which assigns a verb's valency list to the valency list supplied by the grammar on analysis:

    /* for analysis */
    valency1(Verb,V) :-
        var(V), !,
        valency(Verb,V).

2. On the other hand, valency during Generation in German works differently. First the English verb's base form and valency are checked against the equiv_verb rules of the English transfer module. equiv_verb may turn the English valency list into something not identical to the lexically assigned German valency. For example, given the sentence x gives y z. The following conversion (C1) takes place.

<table>
<thead>
<tr>
<th>English</th>
<th>German</th>
</tr>
</thead>
<tbody>
<tr>
<td>np, np, np</td>
<td>np(nom), np(dat), np(acc)</td>
</tr>
</tbody>
</table>

However, when the English sentence x gives y to z is being translated the following conversion (C2) must take place.

<table>
<thead>
<tr>
<th>English</th>
<th>German</th>
</tr>
</thead>
<tbody>
<tr>
<td>np, np, pp(to)</td>
<td>np(nom), np(acc), np(dat)</td>
</tr>
</tbody>
</table>

Only the converted valency in C2 is identical to the valency for geben - to give listed in the German grammar. However the first equivalence C1 is also licensed. This is down to the following rule:

    /* generation */
    valency1(Verb,V) :-
        valency(Verb,Val),
        is_perm_of(V,Val).

The rule above allows any permutation of the verbs valency list to appear as the valency of a sentence during generation.
6.3.4 Word Order of Complements

The word order of the sentence is reflected in the trees. Once the grammar parses a sentence, the complements of the verb appear in the tree, in the exact same order as they did in the sentence. As already pointed out the valency list of the main verb always appears first at the start of a sentence tree. The order of this valency list stays the same as it appears in the valency facts, regardless of how the complements occur in the sentence. Due to word order in German, both these sentences are grammatical,

‘Die Temperaturen liegen bei neun Grad’
the temperatures lie at nine degrees

\[ s([\text{np(nom)}, \text{pp(p(bei))}]) \]

\[
\begin{array}{c}
\text{v(6)} \\
\text{liegen} \\
\text{det(nom,pl,bst)} \\
\text{die} \\
\text{np(nom)} \\
\text{n}\end{array}
\begin{array}{c}
\text{np(dat)} \\
\text{bei} \\
\text{Temperatur} \\
\text{pp} \\
\end{array}
\]

‘Bei neun Grad liegen die Temperaturen’
at nine degrees lie the temperatures

\[ s([\text{np(nom)}, \text{pp(p(bei))}]) \]

\[
\begin{array}{c}
\text{v(6)} \\
\text{liegen} \\
\text{pp} \\
\text{bei} \\
\text{det} \\
\text{np(nom)} \\
\text{n}\end{array}
\begin{array}{c}
\text{det(nom,pl,bst)} \\
\text{die} \\
\text{np(dat)} \\
\text{Temperatur} \\
\text{det(nom,pl,bst)} \\
\text{neun} \\
\text{grad}\end{array}
\]

However, although the order of the complements are different in the two trees, the valency list for the main verb remains the same. Note here that when the grammar is
given a German tree and it has to find the corresponding words, it can only find the German sentence with the complements following the ordering of the complements in the tree and cannot find all possible grammatical sentences. This has an implication in the translation process. When an English sentence is given to the translator a German tree is produced and the words of this tree are looked up, the result will be a German sentence that maintains the same order of complements which occurred in the English sentence and nothing else. This is best demonstrated by an example, If you consider the English sentence ‘Snow will come from the west’ when translated into German, only the first sentence underneath will be given as a translation.

‘Schnee wird aus dem Westen kommen’
  snow will from the west come
‘Snow will come from the west’

‘Aus dem Westen wird Schnee kommen’
  from the west will snow come
‘From the west snow will come’

*(this is not ungrammatical, although unusual in English, it could perhaps be used to stress where the snow is coming from!)*

Although the second sentence is also a grammatical translation, the translator will not offer it as an answer as the order of complements is implicit and preset in the German tree. So, the German grammar will only generate the words of the sentence based on the order of how the complements occur in the tree.

### 6.3.5 Subject-Verb Agreement

Just like in English, in German too the subject of the verb and the verb must agree.

‘The verb in German is linked through it’s endings with the subject of the verb and is said to ‘agree with’ the subject’, Durrell (1999)

The subject and the finite verb must refer to the same *number* and *person* arguments. In English ‘I am’ referring to the 1st person, singular is grammatical and ‘I are’ is not and the same rule applies to German. As I have already said DCGs allow features to be included in the grammar easily and that the downside to this is that the grammar can end up with many features and therefore become messy to work with. On this basis, I decided to reduce the person and number agreement features down to one feature called *Verb_Num* at the noun phrase level. This meant that I represented these features as so:
I allowed pronouns to contain number and person features, however I felt that when dealing with nouns and proper nouns (in the German grammar proper nouns are called -en, this stands for ‘Eigennamen’ - proper nouns in German) that the person feature was redundant. Nouns and proper nouns are always either the 3rd person, plural or the 3rd person, singular and this is determined by their number feature and not their person feature. So, after finding the noun of a noun phrase I used its number feature in the `verb_agree` predicate to find it’s `Verb.Num`.

```prolog
\ For nouns, en, nps etc...
verb_agree(sing,3).
verb_agree(pl,6).
verb_agree(mass,6).
```

This `Verb_num` had to refer to the same number as the verb of the sentence and then both the subject and verb of the sentence would agree. For pronouns, I used both their number feature and their person feature to find their corresponding `Verb.Num` feature and then require that this `Verb_num` was shared by the verb of the sentence.

```prolog
\ For pronouns
verb_agree(sing,1,1).
verb_agree(sing,2,2).
```

All noun phrases carry a `Verb.Num` feature which is got by extracting features from its subcomponents and this `Verb.Num` feature must match with the `Verb.Num` feature of the verb in the sentence. All verbs are assigned some corresponding `Verb.Num` (1-6) based on what form of the verb they represent.

### 6.3.6 Inflecting Adjective Endings

As I have mentioned before German is highly inflectional. I added each German form of the determiners ‘the’ and ‘a’ to the system. Each of these words can be realised with 16 alternative sets of features, which are case, number and gender. Although there are sixteen different possible feature combinations, some of them are realised by the same word.
CHAPTER 6. THE ANALYSIS STAGE

\[ \text{det(nominative,masculine,singular)} - '\text{der}' \]
\[ \text{det(dative,feminine,singular)} - '\text{der}' \]

I decided that each lexical entry for the English word \textit{the} should be included in the German grammar, this was easier than creating a rule to inflected each accordingly, as the only part of the word, which is common to the possible (16 combinations) is the initial ‘d’. Here are some sample article entries included in the German code which all are translations of the English word \textit{the}.

\[ \text{det(D,nom,sing,masc,bst)} \rightarrow \{D = f(\text{det(nom,masc,bst)} & [f(\text{der \\}])]), \}
\[ \text{der}. \]
\[ \text{det(D,dat,sing,masc,bst)} \rightarrow \{D = f(\text{det(dat,masc,bst)} & [f(\text{der \\}])]), \}
\[ \text{dem}. \]

Here a finite state transducer would be very helpful and would allow the inflections of all German determiners to be dealt with efficiently. I decided not to include all forms of all possible German determiners as there are currently morphological plugins, such as the Collins Parser\textsuperscript{1} for English and Morphy\textsuperscript{2}, discussed in chapter 4 that have this capability, such systems could be attached to the system at a later stage.

The system has two procedures for dealing with inflection and adjectives:

1. Analysis of inflected adjective
2. Generation of inflected adjective

When generating inflected German attributive adjectives the system simply takes the base form of an adjective and attaches a suffix onto the end using the predicate \textit{phono_add}. This suffix is determined by the features of this adjective. \textit{phono_add} simply converts the adjective into its ascii representation and then appends the ascii representation of the appropriate suffix to the end of it before returning the inflected form. There is a list of rules representing all possible German adjective endings and their features. Here is an example of one of these:

\[ \text{adj(A,sing,masc,nom,bst)} \rightarrow \{A = f(\text{adj & [f(AdjBase & []])})}, \]
\[ \{\text{AdjInf}, \}
\[ \{\text{phono_add(AdjBase,"e",AdjInf)}\}. \]

This rule states the adjective ending to be attached to an adjective which precedes a nominative masculine noun and proceeds a "bestimmt determiner" (see section X) is

\textsuperscript{1}Collins (1996)
\textsuperscript{2}Lezius (2000)
"e". It can also mean that if a adjective has an extra "e" attached onto it, it could be a nominative, masculine noun which proceeds a "bestimmt determiner".

Apart from phono_add being able to produce grammatical adjective endings, the predicate also allowed all inflected forms of adjectives to be recognised as variants of one base form. It would be pointless to include a adjectives into the grammar if the grammar was unable to identify the connection between the inflected and the base form of the adjective. Therefore, on analysis the suffix was removed and the baseform was identified. The code for phono_add can be seen here:

```
phono_add(Base,Ending,Inflected):-
    /* generation of inflected ending */
    var(Inflected),
    -> (a(Base), /* find the base form of the adjective */
        name(Base,BaseList), /* convert it to ascii repres */
        append(BaseList,Ending,InflectedList),
        name(Inflected,InflectedList))
    ;
    /* analysis of inflected ending */
    (name(Inflected,InflectedList),
     append(BaseList,Ending,InflectedList),
     name(Base,BaseList),
     a(Base)).
```

6.4 Conclusion

This chapter has given the reader some insight into tricky issues which were needed to allow valency play a large role in the translation process. I also described the representational structures used for the German trees. So now that the user is acquainted with the analysis stage, the next chapter will move on to the next process in the translation process, namely the transfer stage.
Chapter 7

The Transfer Stage

7.1 Introduction

A central part of a translator which is based on the transfer approach is its transfer stage. Here I will explain what goes on during the transfer stage of this translation machine. In the first half of this chapter I will introduce the idea of using translation memory which is essential when translating idioms or lexicalization divergences etc ... I will highlight structural divergences which I encountered when mapping complements of a sentence in German to their equivalent English complements and vice versa. The equivalence rules are the same for translation in both directions however the predicates inside them are called in the opposite order. The equivalence rules and lexical transfer rules will be examined during the second part of this chapter.

7.2 Translation Memory

‘Words in one Language, Elegantly us’d
Will hardly in another be excus’d
And some that Rome admir’d in Caesars time
May neither suit our Genius nor our Clime
The genuine sense, intelligibly told
Shews a Translator both Discreet and Bold’;
Earl of Roscommon

As mentioned in chapter five, lexicalisations occur between German and English. This means that a concept in one language cannot be literally translated into another. The reason being that although phrases of two different languages mean the same, two entirely different concepts are used to portray them. Translation memory provides the ability to store and recall previously translated segments of one language into another. This allows different concepts of two languages to be easily mapped to each other. The result is that the translation sounds fluent because no literal translation takes place. The more
translation memory that is stored in a translation system, the more natural and realistic
the translations appear. This system uses translation memory to store several different
types of phrases, these include:

1. Noun phrases such as ‘Hochdruckgebiete’ - *areas of high pressure*, ‘Schönwetterperiode’
   - *bright spells* and ‘ein Nordwestwind’ - *a north-westerly wind* etc ...

2. Location adverbials as in ‘im Südoest’ - in the south-east etc ...

3. Time adverbials, for example, ‘in Frühe - *in the early hours of the morning*, ‘am
   Samstagabend’ on *Saturday evening* plus many more... All of which can be seen
   in the German and English transfer modules.

In the translation memory the tree structure of an expression in German is mapped to
the tree structure of the equivalent English expression. I did not simply map lexical
entry to lexical entry.

7.3 Translation of Verb Complements between Source
and Target languages

Just to recap a structural divergence is when a complement of the verb in one language
is a different type of complement in the other language, i.e. if a verb phrase in English
consists of a verb and a noun phrase while the equivalent German verb phrase consists
of a verb and a prepositional phrase. Many structural divergences occur when transfer-
rning verb complements between English and German in this translation machine. All
transferring of verb complements between German and English are done by the predicate
called `equiv.comps` and this method is the one used for translating verb complements
in both directions. Here I will show the structural divergences and other complement
transfers, which occurred and which were successfully dealt with by `equiv.comps`. I
will list them all first, then explain each one individually and include an example with
trees representing the complement transfers. In the table below features of the English
complements are left empty, however when these features become instantiated they will
contain information of number and person but not case, since the English grammar does
not include case information. However in the table below I will include the case in bold
for English complements. These are only included for illustration purposes and are not
part of the English grammar.
<table>
<thead>
<tr>
<th>No.</th>
<th>English Complement</th>
<th>German Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>np(_) nom</td>
<td>np(nom)</td>
</tr>
<tr>
<td>2.</td>
<td>np(_, _) acc</td>
<td>np(nom)</td>
</tr>
<tr>
<td>3.</td>
<td>np(_, _) acc</td>
<td>np(acc)</td>
</tr>
<tr>
<td>4.</td>
<td>[ ]</td>
<td>reflexive pronoun</td>
</tr>
<tr>
<td>5.</td>
<td>np(_, _) acc</td>
<td>np(dat)</td>
</tr>
<tr>
<td>6.</td>
<td>adjective</td>
<td>indepadj</td>
</tr>
<tr>
<td>7.</td>
<td>adjective</td>
<td>adjplus</td>
</tr>
<tr>
<td>8.</td>
<td>np(_, _) acc + Tag</td>
<td>pp</td>
</tr>
<tr>
<td>9.</td>
<td>np(_, _) acc + Tag</td>
<td>np(dat)</td>
</tr>
</tbody>
</table>

No1. This is not a structural divergence, this simply allows the English nominative subject to be expressed in German as the nominative subject. A simple transfer of a nominative noun phrase in one language to a nominative noun phrase in the other. The English subject is always represented in the English trees as a noun phrase with one feature argument np(\_). English noun phrase complements (objects) which occur in the subcategorisation list (not including the subject, as the subject does not appear on the subcategorisation list for verbs) have two features np(\_, \_). These features are number and person but not case. Here is an example,

‘Schnee wird fallen’

Snow will fall

No2. In German, as I mentioned in chapter 3 some verbs can subcategorise for complements of unexpected cases. In English the verb to be subcategorises for a nominative and an accusative object, however the equivalent German verb ‘sein’ subcategories for two nominative objects.

‘Es ist ein sonniger Tag\_nom’

it is a sunny day\_acc
No3. An accusative object in English becomes an accusative object in German, this like No.1 is just a simple transfer and not a divergence. An example is,

‘Dichte Wolken trüben den Himmel_{acc}’

dense clouds cover the sky_{acc}

No4. In German, there are ‘True’ reflexive verbs, with these, the reflexive pronoun is an integral part of the verb. This reflexive pronoun is omitted in their English equivalents. The code that allows this is:

equiv_comps([],[],f(rfxpro & Dtrs),rfxpro)

There are verbs outside the context of weather, where it would be necessary to include the reflexive in both languages. In the sentence, I wash myself, for example. In German, it is ‘ich wasche mich’. I decided to add this reflexive pronoun as a subcategory on the German verb’s valency list. I felt this was appropriate as it is required in order for the sentence including the verb to be grammatical. An example of the sort of divergence is,

‘das Wetter ändert sich’

the weather changes itself

the weather changes
CHAPTER 7. THE TRANSFER STAGE

No5. Some German verbs require a dative object instead of an accusative object; these verbs are said to govern a sole object in the dative. *Folgen* - ‘to follow’ or *bleiben* - ‘to remain’ are verbs of this type. These verbs lack a clear English equivalent as English does not distinguish between accusative and dative, however they are usually just represented as an accusative object in their English translation. Here is an example of such a verb *bleiben* - ‘to remain’. *Bleiben* requires a nominative object and a dative object. This example will illustrate the differences in verbs like these:

\[ \text{'es bleibt einem sonnigen Tag'} \]

\( \text{it remains a sunny day} \)

No6 & No7 There are English verbs which require an adjective and they are realised in the German translation also as adjectives. My German grammar distinguishes between three types of adjectives, these are *adj* - this stands for an adjective, which precedes a noun and is inflected accordingly, an *indepadj* - this is an adjective which describes the subject of the sentence and an *adjplus* - this is an adjective plus an adverbial of degree, such as *sehr* - very. An example is,

\[ \text{'der Regen bleibt sehr anhaltend'} \]

\( \text{the rain remains very persistent} \)

No8 & No9 This point refers more specifically to the differences between the design of the German representation structures in the German grammar and the English representation structures produced by the chart parser and not to general differences between
the English and German languages. The English chart parser allows verbs to subcate-
gorise not just for a noun phrase or prepositional phrase, but also a word tag. This word tag acts like an adverbial, such as over in 'Come over'. However the German grammar represents these as prepositional phrases or noun phrases in the dative case, this example will illustrate the transfer,

‘Höchdruckgebiete werden nach Süden abgelenkt’
areas of high pressure will towards South diverted
areas of high pressure will divert towards the south

7.4 Equivalence Rules
Syntactic transfer systems such as this one rely on mappings between the surface struc-
ture of sentences. A collection of tree-to-tree transformations is applied recursively to
the analysis tree of the source language in order to construct a target language analysis
tree. The transfer stage of this system is contained in two modules. There is a transfer
module for translating from German to English, and it can be found here:

\users\Public\CSLL\4thYrProjects\Grace\GerGrammar\german_module.pl

and there is a separate module containing the equivalence rules which map English syn-
tax trees to their corresponding German syntax trees, which can be found here:

\users\Public\CSLL\4thYrProjects\Grace\GerGrammar\english_module.pl

The equivalence rules in both of these transfer modules are very similar. The same
predicates are called in both sets of equivalence rules but in the opposite order. This
can be clearly seen when comparing the German to English equivalence rule at the end
of section 7.4 with the same rule going in the English to German direction which can be
found at the end of section 7.5. The reason the equivalence rules are rearranged in each
module is to prevent looping when they are initially supplied with a different tree, (i.e.
the equivalence rules of the English module are given the English tree and must first
extract all the information that it can from this and then transfer this information into a
German tree. While the German module is initially given a German tree and it must pass
all the necessary information form this German tree into the English equivalent). The
reason a German to English rule would loop if it was given an English tree is because the
rule would be all called using variables which were not instantiated. This is why in most
cases a simple reordering of the predicates inside the equivalence rules was enough to
carry out translation successfully. I will run through how the equivalence rules function,
by firstly using the equivalence rules given a German tree and looking for its English
equivalent. Then I will illustrate the transfer rules which are given an English tree and
have to find the corresponding German tree.

### 7.4.1 Finding the English tree using equivalence rules

This equivalence rule is given a German tree and has to find the corresponding English
tree. There are a set of equivalence rules, each of which deal with a certain type of
tree as input, some of these trees represent sentences including finite verbs, verbs with
separable prefixes, verb groups including more than one verb etc.. All of these rules
ultimately do the same thing, they all call the same series of predicates with some minor
alterations. When the German tree is passed into the equivalence method it is unified
with the basic tree structure representing its sentence type, in this case the basic tree is,
the one representing a sentence in the present tense (i.e. with one verb).

\[ f(s(Valency_German) \& [Verb_German|Comps_German]) \]
This makes it easy to refer to the different parts of the tree more generally. Here is what happens then

1. Number and Person information is extracted form the German verb. The tree will contain the German verb and this will include a number between 1-6, the function of this number is discussed in 6.3.5. This number is broken down to find the Number and Person features of the German verb. These features are then passed to the equivalent English verb.

2. All complements of the German sentence contained in the variable Comps\_German. All of the comps contained in Comps\_German are then matched to their equivalent English comps, this technique is used to achieve complement translation and was explained in section 7.3.

3. Both the Valency list of the German sentence Valency\_German and its complements Comps\_German are used to find the equivalent English verb.

    \[
    \text{equiv\_verb}(-\text{EngVerbName}, -\text{EngComps}, +\text{GerVerbName}, +\text{GerComps})
    \]

4. At this stage the base form of the English verb and its valency list should have been found. The first complement on the English valency’s list becomes the subject of the English sentence.

5. On the English side, each verb is assigned a number. Using this number in a method called

    \[
    \text{subcat\_english:decode}(+\text{Number}, -\text{Valency\_List})
    \]

    you can find the subcategorisation term for the verb. This subcategorisation term is made up from atomic categories which were described in section 6.2.1. However, at this point all we have is the list of English complements which have been found as the equivalences of the German complements. As each German complement was mapped to an English complement earlier, its category type was added to a list called FlatEngVal, this list just contains the basic subcategory information such as, np(., .) or np(.). This flat subcategory information has to be converted into the equivalent subcategorisation term in order to be able to find the verb’s number. This is done in a predicate called subcat\_flatten, this takes the basic valency list of the English verb and converts it into the abstract form required by the English chart parser.

6. When you have found the verbs subcategorisation term, it can be passed into the subcat\_english:decode(+\text{Number}, -\text{Valency\_List}) method. This method is usually
given a verbs number and it will return the verbs subcat list. However when used
in reverse you can give the methods the subcategorisation list and it will return
the subcategorisation number of the verb. This number is then put into the verb
variable of the English tree. This allows the English verb to be looked up in the
grammar, when converting the words in the tree to their inflected forms.

7. The complements are then broken up into three parts the subject, the left-hand
side and the right-hand side so they can be added to the English tree. All of the
complements are added into the right hand side and the left hand side remains
empty.

Here is an example of what an equivalence transfer rule looks like, given the German
tree as input.

/* unifies the German tree with the basic German tree structure
and sets up the basic translated English tree (as variables which
will be later instantiated) */
equiv(f(s(_)&[NP,f(vp(Feats)&[f(lhs&L),Verb_English,f(rhs&R)])]),
f(s(Valency_German)&[Verb_German|Comps_German])) :-

/* extract the feature info and base form of the German verb */
Verb_German=f(v(Feature_No)&[f(Word_German&[])]),

/* using the German feature info find the number and person
information needed for the equivalent English verb */
find_features(Feature_No,Person,Number),

/* using the German sentence’s valency information and collected
complements find the equivalent English complements */
equiv_comps((NPCAT,FlatEngVal),Comps_English,Valency_German,Comps_German),

/* using the base form of the German verb and its valency list
find the equivalent English verb */
equiv_verb(Word,(NPCAT,FlatEngVal),Word_German,Valency_German),

/* set up the English subject */
NP=f(NPCAT & _),

/* using the basic valency list for the English verb convert it
into the equivalent subcategorisation term which will be
recognised by the English grammar */
subcat_flatten(Valency_English,FlatEngVal),
/* using the subcategorisation term find the verbs valency number to be added into the English tree */
subcat_english:decode(ValN,Valency_English),

/* add English verb’s number and base form into the verb part of the English tree */
Verb_English=f(lexeme(Word,(ValN,_Desc))&[]),

/* put all of the English complements into the right hand side of the English tree and leave the left handside empty */
merge_comps(NP,L,R,Comps_English),

7.4.2 Finding the German tree using equivalence rules

When you have got an English tree, which was supplied by the chart parser, it must be given to the equivalence rules in the English module called eng_trans, in order to find the equivalent German tree. Just like in the German module the English module also possesses many different equivalence rules, which are designed to unify with German trees of different types of sentences. However the underlying algorithm remains the same regardless of the tree types. Here I will describe the algorithm of the transfer rules, which map an English tree structure to its German equivalent.

1. English trees, as I have mentioned before are quite conventional. They are also of a deeper structure than their German equivalents. Verb group information as well as its components are stored in the verb section of the English tree and this is broken up into a left-hand side and a right-hand side. Unlike in the German grammar, verbs in the English grammar can subcategorise for infinitive verbs. The subject of an English sentence always appears at the front of the tree before the verb information. Using the method merge_comps the subject and both the left and right-hand side of the verb are collected together and stored as a list. This list forms the complement list for the English sentence.

2. Next the base form of the verb and its valency number are extracted from the English tree.

3. The verb’s valency number, which was extracted from the tree is used in the method.

        subcat_english:decode(+ValN,-Valency_English)

This method returns the verb’s subcategorisation term.
4. A list of the basic subcategory names which are contained in this subcategorisation term are needed without all the extra information, which is needed by the chart parser. The method below does exactly this.

\[
\text{subcat_flatten(Valency\_English,FlatEngVal)}
\]

It takes the verb’s subcategorisation term, given by the chart parser and returns the list in its basic form, which is stored in \(\text{FlatEngVal}\).

5. This new \(\text{FlatEngVal}\) can be used to compare the English valency list with the German verb’s valency list. English verbs, unlike the German verbs do not include the subject of the sentence on their valency list. So next of all, the English verbs base form and the English verbs valency list (which includes \(\text{FlatEngVal}\) and the subject of the sentence put together) are used to find the corresponding German verb and its valency list.

6. In doing so, the base form of the equivalent German verb is found and is added into the German tree.

7. Using the transfer process outlined in 7.3, the complements of the English sentence are mapped to their equivalent German complements.

Here is an example of what an equivalence transfer rule looks like, given the English tree as input. Notice how the ordering of the predicates inside the equivalence rule is the opposite to that of the equivalence rules, given the German tree as input.

\[
\text{/* unifies the English verb with the basic English tree structure and sets up the basic German structure */}
\text{equiv(f(s(\_)&[NP,f(vp(Feats)&[f(lhs&L),Verb\_English,f(rhs&R)])]),}
\text{f(s(Valency\_German)&[Verb\_German|Comps\_German])) :-}
\]

\[
\text{/* collects all of the English complements together including the subject, the objects on the verbs left hand side and right hand side */}
\text{merge_comps(NP,L,R,Comps\_English),}
\]

\[
\text{/* extract the English verb’s valency number */}
\text{Verb\_English=f(lexeme(Word,(ValN,_Desc))&[]),}
\]

\[
\text{/* finds the English verbs valency information */}
\text{subcat\_english:decode(ValN,Valency\_English),}
\]
7.5 Basic Lexical Transfer

In both modules there are rules called equiv_np and equiv_pp. These rules break down noun phrases and prepositional phrases into their basic components, for example,

\[ \text{np} \rightarrow \text{determiner, adjective, noun.} \]

These subcomponents of the phrases are then looked up in the lexical transfer rules of the transfer module. These lexical transfer rules are exactly the same in both the German to English and English to German modules. Here is an example of some lexical look ups. Take the noun phrase "the heavy rain" in German this becomes "der starke Regen". First the equiv_np rule breaks this noun phrase down into:

- article — the  
- adjective — heavy  
- noun — rain

Each of these words are then looked up in the transfer modules and the equivalent German words and their trees are found. All of them are then put together to form a German noun phrase at the end of the equiv_np rule after they have been lexically looked up. Here are the lexical look ups for these words in the transfer modules. The English article "the" becomes the German determiner "der". The base form of the determiner is given in the lexical transfer rule. This will be inflected when the tree is given to the German grammar to find the German words.
equiv_det(f(det(detf(cscpm,_,})) & [f(tag(the,_,) & [])]), f(det(A,G,bst) & [f(der & [])]))

The English adjective ”heavy” becomes the German adjective ”stark” - this is uninflected in the lexical tree however just like the determiner before, it will be inflected when the tree is given to the German grammar to find the words.

equiv_adj(f(ap & [f(a & [f(heavy & [])]])], f(adj & [f(stark & [])]))).

Finally, the English noun ”rain” translates as the German noun ”Regen”

equiv_n(f(n(npf(sing,0)) & [f(rain & [])]), f(n & [f(‘Regen’ & [])]))).

This breaking down of noun phrases and prepositional phrases into their individual components and then translating each part individually before returning them altogether as a phrase in the target language is the foundation for translation of telegraphic phrases.

7.6 Conclusion

In this chapter I have highlighted key areas in the transfer stage. I have explained how the equivalence rules find a target translation sentence when given a source sentence and how the same rules can be reversed to incorporate a bilingual transfer ability.
Chapter 8

Interacting with the Translator

8.1 Introduction

There are two options for interacting with the translator, these are using a Java interface or in a Prolog shell. Originally just the later was available, but I decided to implement a java interface to insulate the user from having to manually enter the Prolog commands. In this chapter, I will describe the two interaction options and explain how to use them. I will begin with the Java interface.

8.2 Java Interface

8.2.1 Why Java interface?

When I had written my translator in Prolog, I found that I had to use many Prolog queries in order to find any results. Trying to remember these queries and the order that they came in was challenging enough for me to remember, after working with the project for months, never mind someone who was unfamiliar with the system. So, I decided to design an interface which would hide all the messy Prolog modules and predicate names from the user and would produce results quickly and effectively. I could have written an interface mixing Java and Prolog or C and Prolog, but I decided to choose Java because in the first year of my degree, I studied Java and I found that we spent most of our time learning about object orientated concepts, although this was all very useful, I did feel that I never took the opportunity to design an interface and therefore learn how to use real applications alone. I really enjoyed programming in Java and I felt that I hadn’t had much practice with it of late. Therefore, I decided to brush the cobwebs off the book and learn how to create a user interface.
8.2.2 Interface Design

The interface which can be seen in Appendix C was designed to be simple. The interface contains the following:

1. Language pair option button: this button is a drop down button which allows the user to choose the translation direction. The options are German — English or English — German. The system cannot automatically determine which language the source input is in. The button must be set correctly, i.e. if you enter German text then the button must be set to German — English. Otherwise translation will not be possible.

2. Display tree button: this button gives the user the option of viewing the tree representations of the source and target trees. Trees are printed into the Prolog shell from which the translator was ran.

3. Input text field: the user must enter the text to be translated in here. There are restrictions on the input in order to produce successful translations, these are described below.

4. "Check It" button: this button sets the translation wheels in motion. The translation process will either succeed or fail. The results are also printed into the Prolog shell which is running the interface.

The text which is entered into the interface should be a certain format. Here I will first explain the format expected for German text which is entered into the text field.

**German Input Rules:**

- The initial letter on all nouns must be capitalised as expected.

- Only one sentence or phrase can be translated at a time. This is something which could easily be improved and will be discussed in future work.

- No punctuation is allowed.

- Sentences may only contain a verb group and the items on the main verb’s valency list. This is due to limitations in the English grammar.

- The input must be grammatical, all words should be correctly inflected. This could be easily changed and will also be discussed in the future work section. However, I decided that it would advantageous to include this feature. This would allow people to test their adjective endings and word order etc... using the German grammar.

- Numbers must occur as complete words, not abbreviations or symbols.
English Input Rules:

- Punctuation is allowed and can help to eliminate ambiguity.
- Multiple sentences can be translated at once.
- Sentences may only contain a verb group and the items on the main verb’s valency list.
- The initial letter of days, months etc... should appear in uppercase, but not the initial letter for all nouns as in the German case.
- Numbers must occur as complete words, not abbreviations or symbols.

Violating any of the above rules will lead to automatic failure.

Creating a Java interface involved mixing Java and Prolog. Initially, this was quite tricky as there is not much literature available on the topic. Here I will explain basically how both languages were joined in my application to produce a user-friendly and efficient interface. The Java package which was necessary for the combination of both languages is called Jasper, I will describe this here.

8.2.3 Jasper: the package for mixing Prolog and Java

Jasper is a bi-directional interface between programs written in Java and programs written in Prolog, although I used a Java runtime system as my top-level application, i.e. I called Prolog predicates from Java and not Java methods from Prolog. The Java-side of the interface consists of a Java package (Jasper) containing classes representing the SICStus emulator. When the Java Runtime System is used as a top-level application, the SICStus runtime kernel is loaded into the Java Runtime System. Calling Prolog from Java is done by importing the Java package jasper into your main Java class and using it. Jasper contains a set of Java classes, which can be used to create and manipulate terms, ask queries and request one or more solutions. It is necessary to set the classpaths to contain the path to the SICStus libraries, so that Java knows exactly where to look when it wants to use Jasper and the Sicstus classes and therefore to enable communication between Java and Prolog. The list of commands which are necessary to set the classpath variable are listed in section 8.2.4. Multiple SICStus objects are not supported (the SICStus emulator can only be initialised once), it also must be initialised before any predicates can be called. The SICStus emulator is initialised by creating a new SICStus object. All of the Prolog code must be loaded into the java class, I initialise the SICStus object and load all my Prolog code into the main interface class, which is called:

FrameDemo18.java.
When mixing Java and Prolog you have to create SPPredicate terms, these tell Java the names of the predicates it must call in Prolog, as well as how many arguments they take and it passes in the SICStus object that will be used to create the query, see example,

```java
pred = new SPPredicate(sp,"engToGer",2,"");
```

`engToGer` is the name of a prolog predicate. *this says, call the predicate in the SICStus object sp, the predicate that needs to be called is engToGer and it takes two arguments*

After the predicate objects are created, arguments for the query have to be created. The arguments are placed in an array which is passed to a suitable method to make the query. Arguments of a query consist of class SPTerm objects. For example when giving an English sentence we want to find the German translation so, we would need one string object called engsent and another variable to query prolog:

```java
engsentence = new SPTerm(sp, engsent);
    this creates an argument containing the String engsent

gersentence = new SPTerm(sp).putVariable();
    this will become the answer of the query
```

When querying Prolog I am interested in getting all possible results. I decided to store all these solutions in an array and then print the first five solutions to the terminal, when there are at least five solutions. This is because multiple trees can form the answer to one query. In German, for example, as mentioned before, you can use both the future tense or the present tense plus a future time adverbial to indicate the future tense, therefore both need to be offered as correct answers. I decided to count and store the first five answers so that both future tense sentences and present tense sentences would be available for the user. For getting multiple answers Jasper offers the method:

```java
openQuery(pred, args)
```

Each time you ask it for the next solution it will give all the possible solutions until it fails, this is the predicate I chose for finding possible translations. Other available methods are

```java
query(pred, args)
```

If you are only interested in finding the first solution to a goal this is method to use:

```java
queryCutFail(pred, args)
```

This method finds the first solution, then cuts (i.e. eliminates backtracking) and fails.
8.2.4 What is a classpath and why do we need them?

Java strongly encourages the notion of reuse. There are many pre-written Java classes and packages, that provide the user with methods and objects that are already defined. This saves the user time, as they don’t have to rewrite methods and classes that somebody else has already worked out. They are so many packages available that not all classes are part of the original Java platform. When Java needs to make use of classes that are not part of the original Java platform it needs to be told explicitly where to look to find the necessary classes and methods. The location of a package which is given to Java is known as a Classpath.

‘An important statement in a Java class causes a look up process via the classpath’, Bishop, Judith. (2001), pg 129.

When running my interface a number of classpaths have to be set. In order to set these open a shell and enter the following commands. After this your classpaths will have been set. Here is the list of commands needed to set the classpath variable. Number 1 only has to be set if the user will be interacting through the interface.

- `setenv CLASSPATH.:/usr/local/depot/SICStusProlog-3.8.6/lib/sicstus-3.8.6/bin/jasper.jar`
  - this classpath is needed to mix Prolog and Java


- `setenv LD_LIBRARY_PATH /usr/local/depot/db-4.0.14/lib:$LD_LIBRARY_PATH$

- Go to: users/Public/CSLL/4thYrProjects/Grace/Inflecter and type:
  
  perl lemma_to_form.pl lemmatest

How to start up the Java Interface

Here are the instructions for starting up the interface.

- First of all, set up the classpath variable, as explained above in section 8.2.4.

- After the classpath has been set. Move into the directory called

  \users\Public\CSLL\4thYrProjects\Grace\Interface

- Then simply type

  java FrameDemo18
8.3 Prolog shell

The alternative to using to interface is manually querying the translator in a Prolog shell. This is more time-consuming, but it does give the advantage of making it clear where the process has ended if a translation is unsuccessful. All of the input restrictions which apply to the interface, also apply to the Prolog shell. The interface calls one method which calls all the necessary predicated needed for translations. The German — English translation method is called:

\texttt{translatedcg}

For translation from English — German the method, this method is used:

\texttt{engToGer}

Both these methods can be seen in the code here:

\users\Public\CSLL\4thYrProjects\Grace\GerGrammar\consultall.pl

8.3.1 How to run the Prolog shell

The following section will explain how to run the translator in a prolog console.

- All of the classpaths above apart from number 1 must be set.

- Move into the directory called:

  \users\Public\CSLL\4thYrProjects\Grace\GerGrammar

- Then type the following command into the console:

  consult(consultall).

  consultall is a file which consults all of the necessary files needed to run the translator.

8.4 Conclusion

During the course of this chapter I have described both ways of interacting with the translator, i.e. via the interface or manually using a Prolog shell. Some sample queries which are useful when using the translator through the Prolog shell for example for drawing trees and translating etc ... can be seen in the following directory:

\users/Public/CSLL/4thYrProjects/Grace/README

The notion of classpaths was described in this chapter and I have provided information both on how to set the classpath variable needed to interact with this system and on how to start it up using either the interface or the Prolog console.
Chapter 9

Useful Debugging Tips used for this Application

9.1 Introduction

‘No rule is so general, which admits not some exception’
Robert Burton, (The Anatomy of Melancholy)

When I was implementing this translator, I found some methods of debugging more helpful than others. Here I will list and explain what I found to be most helpful when debugging my code.

9.2 Test Classes

I made test clauses all the time to check each component of the translator every time I tried something new. This let me know immediately, if I had changed something, which affected another part of the system. Implementation of this translator involved successfully completing a series of stages. These stages initially included writing a DCG grammar, which could accept all the German sentences corresponding to the translation of the English input. Firstly, I began writing the German grammar. At this point, I began using test classes. These test classes contained a list of German sentences, which were successfully accepted by the German grammar. So each time I added a new rule to the German grammar, I could call on a predicate, which would run all the German sentences through the German grammar. In this way, I could check if the new rule affected any of the other previously accepted sentences and if it did I could sort out the problem then and there instead of having to fix it at a later stage, when it would become difficult to see which rule was the culprit. In stage two, I began writing equivalence rules between the two languages. Here I used test classes again and each time a new type of sentence was successfully translated I added it into the test class. I ended up having test classes which contained sentences and phrases along with the list of tags for each word.
The testing began with the tags being put into a predicate to produce a chart. The chart parser produced all active edges for all the given tags. At this point the test class would print to the screen that a successful chart was produced. Then each edge was put into English to German equivalence rules and if an equivalent German tree was found, it was printed to the screen. This was very helpful as I could see what tree was being produced as the equivalent German tree and if it was incorrect I would know that the problem lay in the equivalence rules and not in the chart parsers interpretation of the English tree as I had already seen the chart parsers parse printed to the screen. As you can see these test classes were time-consuming as all sentences had to be written into the test classes, but they were certainly worth while in the long run. Since the translator had to be tested at many levels test classes were very informative and showed me exactly where the path of success had ended. I was able to immediately identify where a problem lay during the translation process and I would advise anyone taking on such a project or continuing on this one to seriously consider using test classes. Some of my test classes can be seen in my code here:

users/Public/CSLL/4thYrProjects/Grace/GerGrammar/sample_Eng_Input

and

users/Public/CSLL/4thYrProjects/Grace/GerGrammar/sample_Ger_Input

These contain sentence tree which have been pre-parsed and which are used to test if the equivalence rules from one language to another are functioning. While the code to run the tests on these trees is contained in the next two respectively.

users/Public/CSLL/4thYrProjects/Grace/GerGrammar/engtest

and

users/Public/CSLL/4thYrProjects/Grace/GerGrammar/ger_tester

9.3 Tracing

Another method I used to debug my project was tracing. While tracing mode is in force, every time you type a goal at the top level, the debugger starts creeping through the goal immediately. I found tracing through the code to be very informative, it gives a detailed account of what is going on inside each predicate. At the beginning, when I had very few equivalence and grammar rules it was easy to see which rule Prolog was trying to succeed, however as the number of rules grew it became impossible to follow which rule was being checked. Imagine when you have ten predicates called equiv_np and you are not sure which on of these predicates is being traced. You would spend all of your time jumping from buffer to buffer checking the first, second, ... line of each
equiv_np predicate checking which one it is and probably end up getting lost. Blindly tracing through a predicate, which occurs once in your code can be useful. However in a translation system it is very unlikely to have each predicate uniquely named. This was the main reason why pltrace was so good. This allows you to trace as normal, but a separate buffer will pop up, which highlights the line of code being traced and shows exactly which particular method is being analysed. I could then easily identify which rule was being tested. This prevents blind tracing and I found this to be an indispensable source of information when tracing through my equivalence rules.

Here’s how to load pltrace:

1. In a buffer type Esc-X run-prolog or Esc-p
2. Esc-X load-file then /users/Public/CSLL/4thYrProjects/pltrace.el
3. Esc-X pltrace-on
4. If the code is written in DCGs, it will trace through difference list representation of your DCG rules, to prevent this reconsult your code
5. trace.

9.4 Spypoints

When tracing through a predicate, which contains three sub-predicates inside it, you may already know that the first two succeed or you may only be interested in the third one. Here you should use a spypoint. Spypoints allow you to indicate where the tracer should stop on a per-predicate basis. First you would set a spypoint on that predicate.

/* stop when you get to equiv_np and lets see what’s going on */
spy(equiv_np)

Then you trace as normal, but if only interested in the spypoint position type ‘l’ into the prompt and the tracer will leap to the spypoint position, this often saves time. Multiple spypoints can be set at one time.

9.5 Drawing Trees

In a application, such as this tree structures must be used both during analysis, transfer and generation. These trees are often long and difficult to comprehend when they are written to the screen. Here the tree drawing predicate was invaluable. Trees were able to be drawn on the screen in an easily comprehensible manner. I found this was very useful as it avoided looking at some flat tree structure, trying to figure out, which parts belong to each other. The draw tree predicate can be found here:
9.6 Conclusion

In chapter 9 I have listed some things I found useful when trying to debug my application. I found debugging the system and trying to figure out what’s wrong was time-consuming, especially when more often than not it is some tiny dot misplaced or some syntax error!! Hopefully this chapter will alleviate some stress. It is provided to help anyone taking on a similar project.
Chapter 10

Evaluation

10.1 Introduction

In this chapter my aim is to illustrate what kind of translations the translator will produce using two sample weather report texts. I chose two weather reports which have a good mix of translations on offer. I felt in doing so, I can clearly show parts of the reports which are translated accurately, and can also point out parts of the reports which I have chosen to exclude from my system.

10.2 Sample Weather Data

The German weather report article which will be used to illustrate the system in action translating from German to English can be seen in Appendix B (Figure 10.10). The English weather report used to show translation from English to German can also be seen in Appendix B (Figure 10.9).

10.3 German — English Evaluation

Both articles for my evaluation were chosen so that areas of weather reports which can be translated easily by my system as well as areas which are not incorporated as of yet will be highlighted. If the entire German weather report in Figure 10.10 was entered into the translator without any pre-editing a successful translation would not be possible. As I have mentioned in section 5.3.2 the German translator can just translate one sentence at a time. This is something which could be easily improved and which I will discuss in the future work section. Also, due to the English grammar restriction, i.e. only sentences including their verb group and complements which the main verb subcategorises for are allowed be part of a sentence and nothing else. This meant that the German sentences had to be of the same form to be able to produce an English translation tree which would be the same as an English tree produced by the chart parser. Here I have pre-edited the
German text to optimize the translation. These are the steps I took in pre-editing the text:

1. The number 17 is changed to seventeen.

2. I have broken down the text based on my knowledge of the systems capabilities.

3. Punctuation has been removed.

Next I will number each German segment which was entered into the translator and I will follow this table (10.1) with another two tables (10.2 10.3) with the results which were produced. I will then give the translations which were produced and add some comments.

Comments on Translations:

- ":das, mit einer Warmfront ..." - this sentence cannot be translated. I listed in my assumptions in section 3.7 that I decided not to include the translation of relative clauses. Relative clauses are not included in the English grammar rules, therefore I decided to leave them out in the German grammar too, since they usually do not occur very often in weather reports. These could be added to the system’s capabilities at a later stage.

- When the system cannot translate an entire phrase such as the relative clause above, doing a direct translation of every word is a solution. This may lead to inaccurate translations but at least the translator gives its best attempt. In the case of weather reports this usually works out quite accurately.

- In the text of Figure 10.10 translation number 6 has an extra ”n”, i.e. ”mit leichten Regenschauern”. During pre-editing I removed this ”n”. The reason for this is that German nouns which are plural and in the dative case get an extra ”n” on the end of them. This feature is not included in the morphology of the system in it’s current state.

- At times the translator produced multiple translations, see translation six from German to English. The reason translation six produced four answers is because there are two entries in the lexicon for ”leicht”, these are light and moderate. Nouns are included in their tree representations. The plural form of a noun is contained in the tree of a plural noun and the singular form of a noun is contained in the tree of a singular noun. The word ”Regenschauer” is used to represent one shower and also many showers. Therefore in this translation the translator does not known which one is needed, so it offers both as possible translations.

- ”ein” can be translated as the article ”a” as in a dog or also as ”one” as in one dog.
### Translation No. 1 Pre-edited German Input

<table>
<thead>
<tr>
<th>Translation No.</th>
<th>German Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>in Hessen</td>
</tr>
<tr>
<td>2</td>
<td>die neue Woche beginnt mit dichten Wolken</td>
</tr>
<tr>
<td>3</td>
<td>und</td>
</tr>
<tr>
<td>4</td>
<td>vereinzelt</td>
</tr>
<tr>
<td>5</td>
<td>auch</td>
</tr>
<tr>
<td>6</td>
<td>mit leichten Regenschauer</td>
</tr>
<tr>
<td>7</td>
<td>Schuld daran ist</td>
</tr>
<tr>
<td>8</td>
<td>ein Tief</td>
</tr>
<tr>
<td>9</td>
<td>über den britischen Inseln</td>
</tr>
<tr>
<td>10</td>
<td>das</td>
</tr>
<tr>
<td>11</td>
<td>mit einer Warmfront</td>
</tr>
<tr>
<td>12</td>
<td>feuchte Luftmassen</td>
</tr>
<tr>
<td>13</td>
<td>heranfürt</td>
</tr>
<tr>
<td>14</td>
<td>die Sonne wird sich zeigen</td>
</tr>
<tr>
<td>15</td>
<td>nur</td>
</tr>
<tr>
<td>16</td>
<td>zeitweise</td>
</tr>
<tr>
<td>17</td>
<td>bei einem starken Südwestwind</td>
</tr>
<tr>
<td>18</td>
<td>am Nachmittag</td>
</tr>
<tr>
<td>19</td>
<td>die Temperaturen erreichen siebenzehn Grad</td>
</tr>
<tr>
<td>20</td>
<td>am Dienstag</td>
</tr>
<tr>
<td>21</td>
<td>vor allem</td>
</tr>
<tr>
<td>22</td>
<td>in Frühe</td>
</tr>
<tr>
<td>23</td>
<td>dichte Wolken kommt</td>
</tr>
<tr>
<td>24</td>
<td>und</td>
</tr>
<tr>
<td>25</td>
<td>Regenschauer</td>
</tr>
<tr>
<td>26</td>
<td>in den Nachmittagstunden</td>
</tr>
<tr>
<td>27</td>
<td>die Niederschläge klingen ab</td>
</tr>
<tr>
<td>28</td>
<td>die Temperaturen ändern sich</td>
</tr>
<tr>
<td>29</td>
<td>nur wenig</td>
</tr>
<tr>
<td>30</td>
<td>das Wetter wird wolken bleiben</td>
</tr>
<tr>
<td>31</td>
<td>dann</td>
</tr>
<tr>
<td>32</td>
<td>unbeständig</td>
</tr>
<tr>
<td>33</td>
<td>danach</td>
</tr>
<tr>
<td>34</td>
<td>es geht weiter</td>
</tr>
<tr>
<td>Translation No.</td>
<td>English Translation</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1</td>
<td>in, ’Hessen’</td>
</tr>
<tr>
<td>2</td>
<td>the, new, week, begins, with, dense, clouds</td>
</tr>
<tr>
<td>3</td>
<td>and</td>
</tr>
<tr>
<td>4</td>
<td>[individual];</td>
</tr>
<tr>
<td>5</td>
<td>[isolated]</td>
</tr>
<tr>
<td>6</td>
<td>[also]</td>
</tr>
<tr>
<td>7</td>
<td>[with, light, rain, shower];</td>
</tr>
<tr>
<td></td>
<td>[with, light, rain, showers];</td>
</tr>
<tr>
<td></td>
<td>[with, moderate, rain, shower];</td>
</tr>
<tr>
<td></td>
<td>[with, moderate, rain, showers]</td>
</tr>
<tr>
<td>8</td>
<td>[Schuld daran ist]</td>
</tr>
<tr>
<td>9</td>
<td>[a, cyclone];</td>
</tr>
<tr>
<td>10</td>
<td>[one, cyclone]</td>
</tr>
<tr>
<td>11</td>
<td>[above, the, british, islands]</td>
</tr>
<tr>
<td>12</td>
<td>[the]</td>
</tr>
<tr>
<td>13</td>
<td>[heranfuehrt]</td>
</tr>
<tr>
<td>14</td>
<td>[the, sun, will, appear]</td>
</tr>
<tr>
<td>15</td>
<td>only</td>
</tr>
</tbody>
</table>

Figure 10.2: Translators Results
during a heavy south westerly wind on the afternoon, on the afternoon, 'On', the afternoon, in the afternoon
the temperatures, reach ten, degree on, 'Tuesday'
above all in the early morning
and dense, clouds, come; thick, clouds, come
and
rain, shower; rain, showers
during the afternoon
the rainfall, 'lexeme([wear, off], (53, 913))'
the, temperatures, change
only little
the, weather, will, remain, cloudy
then
changeable
afterwards
it, 'lexeme([continue], (53, 803))'

Figure 10.3: Translators Results
• Translation no 13 was unsuccessful. I have decided that the translator should return the words back unaltered if translation is not possible. This will be successful if the word is a place name but otherwise it won’t. This improves the systems robustness. If it can not translate a word it just returns the word unaltered instead of crashing. "heranführt" in this case cannot be translated as I have not included a rule to translate individual verbs, this could easily be implemented at a later stage.

• In translation number 14 there is reflexive pronoun present on the German side but it is absent on the English side. This method was discussed in section 7.3.

• Translation 18 is an example of one of the associated problems with translation by transfer. As the system does not have any semantic information at its disposal, it cannot chose the best choice, if a number of possible translations arise. In the Prolog console all alternatives will be offered. However using the interface only one possible answer will be returned, i.e. the first answer which is found. In German "am" can mean on or in, therefore both alternatives are presented to the user.

• Translation 22 is an example of a translation produced by translation memory. Both German and English trees are stored in the lexicon and when one is found it is automatically mapped to the other.

• Translation number 27 shows translation of a separable verb. The prefix is not translated into English directly.

• Sometimes English words cannot be recognised on analysis, examples of these are "later", "today", "above all" etc ... in this case I have mapped them lexically. (a tree is created containing this word, 

\[f(\text{later} \& \ [\ ])\]

This is then mapped to the equivalent German tree and the word can be translated.

As can be seen from the results above the translation process has been a success. Although it is time-consuming to pre-edit the input, it does lead to accurate translation. Météo is an example of a system which requires pre-edited text. My translation system is capable of translating both telegraphic phrases and sentences. Although the system cannot translate long sentences with additional complements, it is capable of successfully translating such a sentence when it is broken down. Developing this system further would lead to improvements in this area.

10.4 English — German Evaluation

Once again on the German side pre-editing has taken place. This was to improve the translation outcome. Here is what I did to pre-edit the text:
1. Broke up "central and southern area’s" into two phrases. The translator could not translate this as one phrase although the chart parser could recognise it all as a complete phrase. There are rules in the system to cover translation for this type of noun phrase, but perhaps in altering something else I have changed them.

2. "a strong to gale-force southerly wind" is also broken down. I did not expect that the translator would have been able to translate this as the hyphenated word would not have been recognised by the tagger.

3. "7-9" changed into word form. As the translator does not work with numbers only words. Since numbers are universal between German and English, a rule could be added to the system to allow the numbers to remain the same in both languages.

During translation from English — German the entire text can be entered at once and the translator will try to translate it. However, the parser will still break the text up into grammatical chunks. Figure 10.4 shows the pieces which were recognised as individual segment by the translator. Sometimes I rearranged the input to allow a successful translation which the translator otherwise would not have gotten. These rearranged inputs are marked with a *. The translation from English to German, just like German to English was very successful. A large amount of the input text was accurately translated. Here I will outline translations which I feel need explaining:

**Comments on Translations:**

- Translation number 5 and 10 produce two possible translations for each sentence. This is because an English sentence in the future tense can be converted into a German sentence including the auxiliary werden + an infinitive an also into a sentence including just a finite verb. This was discussed in detail in section 6.3.2.

- Translation 9, 10 and 6 were unsuccessful. In all cases [untranslated] was returned, this was added to make the system more robust. Alternative approaches could have been taken to deal with words and phrases which cannot be translated.

- For translation number 3 and other four possible answers are returned. This is because the case of the noun phrase is not specified, therefore the translator provides the noun phrase in all four possible cases.

- The preposition "to" can have different meanings in various contexts. Therefore in translation 17 the translator provides all possible translations of two which are stored in it’s lexicon.

**10.5 Results and Conclusions**

In this chapter, I have provided two sample translations produced by my translator. The results were very encouraging. The translator was capable of producing accurate
<table>
<thead>
<tr>
<th>Translation No.</th>
<th>English Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a gusty night</td>
</tr>
<tr>
<td>2</td>
<td>rain will linger in the north</td>
</tr>
<tr>
<td>3</td>
<td>central areas*</td>
</tr>
<tr>
<td>4</td>
<td>southern areas*</td>
</tr>
<tr>
<td>5</td>
<td>southern areas will have rain</td>
</tr>
<tr>
<td>6</td>
<td>giving</td>
</tr>
<tr>
<td>7</td>
<td>way</td>
</tr>
<tr>
<td>8</td>
<td>clear intervals</td>
</tr>
<tr>
<td>9</td>
<td>although</td>
</tr>
<tr>
<td>10</td>
<td>showers will spread</td>
</tr>
<tr>
<td>11</td>
<td>in</td>
</tr>
<tr>
<td>12</td>
<td>from the west</td>
</tr>
<tr>
<td>13</td>
<td>a strong wind*</td>
</tr>
<tr>
<td>14</td>
<td>a southerly wind*</td>
</tr>
<tr>
<td>15</td>
<td>minimum temperature*</td>
</tr>
<tr>
<td>16</td>
<td>seven degrees</td>
</tr>
<tr>
<td>17</td>
<td>to nine degrees</td>
</tr>
</tbody>
</table>

Figure 10.4: English Input to Translator

translations for most of the input texts.
<table>
<thead>
<tr>
<th></th>
<th>English Translation Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[eine, stürmische, 'Nacht']</td>
</tr>
<tr>
<td>2</td>
<td>['Regen', bleibt im, 'Norden']</td>
</tr>
<tr>
<td>3</td>
<td>[zentrale, 'Bereiche']</td>
</tr>
<tr>
<td></td>
<td>[zentrale, 'Bereiche']</td>
</tr>
<tr>
<td></td>
<td>[zentralen, 'Bereiche']</td>
</tr>
<tr>
<td></td>
<td>[zentraler, 'Bereiche']</td>
</tr>
<tr>
<td>4</td>
<td>[südliche, 'Bereiche']</td>
</tr>
<tr>
<td></td>
<td>[südliche, 'Bereiche']</td>
</tr>
<tr>
<td></td>
<td>[südlichen, 'Bereiche']</td>
</tr>
<tr>
<td></td>
<td>[südlicher, 'Bereiche']</td>
</tr>
<tr>
<td>5</td>
<td>[südliche, 'Bereiche', haben, 'Regen']</td>
</tr>
<tr>
<td></td>
<td>[südliche, 'Bereiche', werden, 'Regen', haben]</td>
</tr>
<tr>
<td>6</td>
<td>[untranslated]</td>
</tr>
<tr>
<td>7</td>
<td>['Weg']</td>
</tr>
<tr>
<td></td>
<td>English Translation Results</td>
</tr>
<tr>
<td>---</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>8</td>
<td>[heitere,'Auflockerungen']</td>
</tr>
<tr>
<td></td>
<td>[heitere,'Auflockerungen']</td>
</tr>
<tr>
<td></td>
<td>[heiteren,'Auflockerungen']</td>
</tr>
<tr>
<td></td>
<td>[heiterer,'Auflockerungen']</td>
</tr>
<tr>
<td>9</td>
<td>[untranslated]</td>
</tr>
<tr>
<td>10</td>
<td>['Schauerregen',verbreiten]</td>
</tr>
<tr>
<td></td>
<td>['Schauerregen',werden,verbreiten]</td>
</tr>
<tr>
<td>11</td>
<td>[untranslated]</td>
</tr>
<tr>
<td>12</td>
<td>[von,dem,'Westen']</td>
</tr>
<tr>
<td></td>
<td>[aus,dem,'Westen']</td>
</tr>
<tr>
<td>13</td>
<td>[ein,starker,'Wind']</td>
</tr>
<tr>
<td></td>
<td>[einen,starken,'Wind']</td>
</tr>
<tr>
<td></td>
<td>[einem,starken,'Wind']</td>
</tr>
<tr>
<td></td>
<td>[eines,starken,'Wind']</td>
</tr>
<tr>
<td>14</td>
<td>[ein,suedlicher,'Wind']</td>
</tr>
<tr>
<td></td>
<td>[einen,suedlichen,'Wind']</td>
</tr>
<tr>
<td></td>
<td>[einem,suedlichen,'Wind']</td>
</tr>
<tr>
<td></td>
<td>[eines,suedlichen,'Wind']</td>
</tr>
<tr>
<td>15</td>
<td>[minimal,'Temperatur']</td>
</tr>
<tr>
<td>16</td>
<td>[sieben,'Grad']</td>
</tr>
<tr>
<td>17</td>
<td>[zu,neun,'Grad']</td>
</tr>
<tr>
<td></td>
<td>[nach,neun,'Grad']</td>
</tr>
<tr>
<td></td>
<td>[bis,neun,'Grad']</td>
</tr>
</tbody>
</table>

Figure 10.6: English Translation Results
Chapter 11

Conclusion

11.1 Introduction

Throughout the last 10 chapters, I have described the translation system and how it carries out its processes. In this chapter I will suggest possible ways of enhancing this system and will also highlight what I feel I have gained from taking on this project. I will finish off with a quick summary of the ten chapters.

11.2 Future Work

There is major scope for improving the translator. Time restrictions meant that I could only do some of the things I would like to have added to the system. Some additional goals which could not be undertaken due to time restrictions are mentioned in section 1.2. Here I will mention some suggestions for further work.

11.2.1 Adding a Morphological Plugin

As I have already mentioned in chapter 3. The lexicon of the current German grammar could benefit immensely from a morphological plugin. This would allow a greater vocabulary to be analysed and generated and would improve on the system’s current handling of German morphology.

11.2.2 Including an Interface to add to the German Lexicon

An interface could be added to the system which would allow the user to choose which type of word they would like to add to the German grammar (verb, noun, adjective).
The user could then select a number of buttons which would add the word into the lexicon. The same interface could add lexical equivalence entries such as those described in section 7.5. This would eliminate the user having to enter words directly into the Prolog code. Equivalence rules for complete sentences and phrases are not general enough to be added by an interface, however single word entries could be added easily.

11.2.3 Telegraphic Chunks

The system could be further developed to deal specifically with telegraphic style reports, like Météo does. Initially when starting out my focus was on translating sentences, but now that this has been achieved moving on specifically to telegraphic reports would improve the systems capabilities.

11.2.4 Accept more than one German Sentence at a Time

On the English side, an entire text can be entered to be translated at a time. This is not the case in translation from German to English. I have written code which should do this, though I never had a chance to test it. As mentioned before the method translatedcg takes a German phrase and tries to parse a sentence structure, noun phrase, prepositional phrase etc ... from it with nothing else left over. In the code I have written an alternative translatedcg rule which still looks for one of these categories but it allows something else to be left over in the phrase and not just the empty list. This however also has an implication on accepting ungrammatical phrases. For example, if the noun phrase ”die\textit{fem} Wetter” - the weather was given to the system to be translated it would fail because Wetter is a neuter noun and therefore there are conflicting features between the article and noun making the noun phrase ungrammatical. However, if the new translatedcg predicate was added to it ”die\textit{fem} Wetter” would be accepted because first the translator would find the word ”die” and would translate it and then look at what was left over namely ”Wetter” and this then would also be independently translated. The likelihood of finding a translation would improve.

11.3 Transformational Grammar

When an English sentence is translated via a number of smaller individual translations, this could then be enhanced if all of these could be put back together, and rearranged to be grammatical. This could include using a transformational grammar to rearrange phrases. Alternatively the words could be taken out of the trees and sorted in their most
likely word order position, using some statistical knowledge base.

11.4 Achievements

This project has definitely been an invaluable learning experience. I feel like all areas of my degree have come together within my project. My initial goals have definitely been achieved and I am very happy with the outcome. Initially I found the task to be quite daunting, when starting out with a blank canvas but as time went on the project began to develop and I began enjoying it. I feel that this project was both a challenge and an enjoyable experience, in which I gained programming experience in both Prolog and Java, as well as improving my computational linguistic and German knowledge. I had never written a user interface before and I feel that in doing this I have boosted my confidence in programming through Java.

11.5 Summing Up

The most time consuming part of this project was writing the equivalence rules which map English to German. Initially I began developing the system in an English to German direction. This took much longer than reversing the system to incorporate translation from German to English. This is due to many of my methods being fully reversible. During the course of this paper I have highlighted many areas of my system. In chapter one I mentioned my motivation for choosing this project, as well as my aims when starting off. Chapter two presented an overview of the art of machine translation and described some approaches on offer. Chapter 3 outlined the differences between German and English, and also introduced the reader to my sub-domain of weather reports. Chapter 4 provided a revision section on definite clause grammars and chart parsers. Chapter 5 introduced the main methodologies implemented in my translator and explained how the bidirectional translation processes were carried out. Chapter 6 and 7 explain areas of my implementation during the analysis, transfer and generation stages. Chapter 8 explains that the user can access the translator through an interface or a Prolog shell and provides information on various technical matters. Chapter 9 shows how the system performed when translating two random weather reports. This chapter describes what I feel I have gained from this project and it also outlines further models of the project that could potentially be implemented.
Appendix A
Hello, and thank you for calling Weatherdial, Met Éireann’s forecast service. Here is the forecast for the Province of Leinster updated at 1800 hours on Thursday, 06 March 2003

Apart from a few isolated showers, the evening and early night will be dry, and cool at first, with clear spells in most parts. Cloud will thicken from the west overnight bringing some outbreaks of rain to many areas dawn. Southerly winds will continue to freshen through the evening and night becoming fresh to strong south to southeast by morning. Lowest temperatures will range 4 to 7 Celsius but somewhat milder by morning.

Outbreaks of rain will spread quickly to all parts of the province early Friday morning and gradually turn more persistent, giving some heavier bursts in places during the morning and afternoon. The rain will turn more showery late in the afternoon and clearer drier weather reach all areas during the evening. Highest temperatures 8 to 10 Celsius. Winds will be fresh to strong south to southeast at first in all areas, become moderate variable in northern and midland parts of the province for a time, then fresh to strong west to northwest later. In the extreme south of the province the fresh to strong south or southeast winds will gradually veer fresh to strong west to northwest during the afternoon.

......... and that’s the short range forecast for Leinster until Friday evening and now here’s the outlook for Leinster for the following few days

The early part of Friday night will be dry and rather cool. Thickening cloud will bring further rain or drizzle overnight and early Saturday. The rain will become patchy during Saturday and good dry periods will develop in most places. The day will be mild with fresh southwesterly winds. Little change Saturday night - it’ll be mild and windy with patchy rain or drizzle, but good dry periods.

A spell of heavier and more persistent rain on Sunday will be followed by clearer, fresher showery conditions. Winds will be fresh to strong south to southwest.

Monday will be cool and windy with widespread showers, many of which will be heavy.

There’ll be further showers on Tuesday, but the weather looks like becoming dry and settled for the second half of the week.
Figure 11.2: Sample German Weather Report - Süddeutsche Zeitung March 25th
Appendix B
Figure 11.3: English Evaluation Weather Report

Source:
The Evening Herald Newspaper
Mon 27th of April 2003
Figure 11.4: German Evaluation Weather Report

Source:
www.wetterbericht.de
Mon 27th of April 2003
Appendix C
Figure 11.5: The Interface
Figure 11.6: Sample Translation (English — German)
Figure 11.7: Sample Translation (German — English)


Karlsson, Fred and Karttunen, Lauri (2001) Sub-Sentential Processing. Rank Xerox Research Centre, Meylan, France. Can be found online at: http://cslu.cse.ogi.edu/HLTsurvey/ch3node4.html


124


SuperTagger (1997). Can be seen online at: http://www.cis.upenn.edu/ mickeyc/SuperTags


TreeTagger - A Language Independent Part-of-Speech Tagger. Can be found here: http://www.ims.uni-stuttgart.de/projeckte/corplex/TreeTagger/DecisionTreeTagger.html


Wolfgang Lezius, Reinhard Rapp and Manfred Wettler, A Morphology-System and Part-of-Speech Tagger for German.

Collins German Dictionary, 2001

http://leo.dict.org

http://www.cs.tcd.ie/Martin.Emms/NLP_01_02/Parsing/Lexicalised/

http://cswww.essex.ac.uk/TechnicalGroup/sicstus/sicstusunderscore12.html
http://www.afm.sbu.ac.uk/logic-prog/

http://java.sun.com

http://www.nzz.ch/wetter

http://www.wetterbericht.de