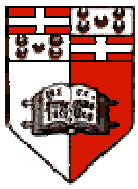
An Automatic Machine Translation System for English-Maltese

Paulseph-John Farrugia

Supervisor: Michael Rosner



Department of Computer Science & AI

University of Malta

June 1999

Submitted in partial fulfilment of the requirements for the degree of

Bachelor of Science (Hons)

I Abstract

Machine Translation is a research field in Computer Science whose aim is to automate the process of translating text from one language to another using computers. This field is of social, political, commercial, scientific, intellectual and even philosophical interest. Research in MT has been going on since the 1940s, but unfortunately, good results have been difficult to obtain due to the inherent difficulty of the translation task.

Briefly specified, an Automatic Machine Translation system operates by first parsing text in a source language into some kind of representation, and then using that representation to generate the corresponding target language text. Of course, this process must preserve the meaning, and possibly some of the style, of the original text.

The kind of representation used influences heavily the general design of the system. While there is no recognised standard form of representation or generation technique, two distinct approaches can be described in the abstract. The first and most easily comprehensible is the interlingual approach, in which the representation is a natural language free meaning representation, that captures only the semantic content of the input sentence. This structure is then used as the basis for target language synthesis.

The second kind of design is the transfer approach in which the representation structure remains much closer to the source language and still retains some of its linguistic characteristics. Transfer rules are used to transform this into another representation structure conveying the same meaning but containing the linguistic characteristics of the target language. A final phase translates this last structure into the output text.

Such a specification of the translation process is highly over-simplified, and belies many details and problems that have to be overcome in order to achieve reasonable quality in translation output. Some of the problems, such as the syntactic and semantic analysis of texts, are ones that exist in the Natural Language Processing field of research. Others are specific to Machine Translation.

This report documents a project that aimed to implement a Machine Translation system for translating between English and Maltese. The implementation of a general system capable of translating any arbitrary text is a task of huge proportions, so the translation was to be based on a text archive that would have to be created specifically for the purpose. This project was carried out by the undersigned, Paulseph-John Farrugia, in partial fulfilment of the requirements for the degree of Bachelor of Science (Hons).

In order to provide a context for the main documentation of the project, various aspects of the field of Machine Translation are discussed first. These illustrate the history of this field, the issues involved, the results obtained from years of research, as well as the current state of the art. Describing in detail each of these aspects would require a report of huge proportions, so only the major ideas are given. A comprehensive literature list indicates where to refer to for further information.

II Acknowledgements

I am greatly indebted to Mr. Michael Rosner, my project supervisor, who also introduced me to the subject concerned, for his help and guidance in the development of this project during the last months. His patience and commitment merit special gratitude.

In addition, I would also like to thank the other lecturers and members of staff at the Department of Computer Science and AI, and at the Board of Information Studies, as well as a few other supportive persons, whose aid during these last four years in hell was highly appreciated.

Thanks.

Grazzi.

III Table of Contents

I ABSTRACT I

II ACKNOWLEDGEMENTS III

III TABLE OF CONTENTS IV

IV LIST OF TABLES IX

V LIST OF FIGURES X

1 INTRODUCTION 1

1.1 AIMS AND OBJECTIVES OF THE PROJECT 3
1.2 OUTLINE OF WORK DONE 4
1.3 STRUCTURE OF REPORT 6

2 WHY TRANSLATION IS DIFFICULT 9

2.1 AMBIGUITY 10
2.2 LEXICAL AND STRUCTURAL MISMATCHES 11
2.3 TRANSLATING IDIOMS AND COLLOCATIONS 12
2.4 MEANING AND INTERPRETATION 13

3 OVERVIEW OF MT 14

3.1 CATEGORIES OF MT SYSTEMS 14
3.2 MOTIVATION AND AIMS FOR MT 16
3.3 A BRIEF HISTORY OF MT 18
3.3.1 Birth and Early History 18
3.3.2 The Georgetown Project 19
3.3.3 The ALPAC Report 19
3.3.4 Renewed Interest: SYSTRAN and Controlled Languages 20
3.3.5 A Real MT Success: METEO and Sublanguages 21
3.3.6 Increase in Use and Research: EUROTRA 21
3.3.7 Modern Day MT: Verbmobil 22

4 MT ARCHITECTURES 24

4.1 A Typical Software Architecture 24
4.1.1 The Long Term Memory *25*
4.1.2 The Short Term Memory *25*
4.1.3 The Translation Process *25*
4.2 Transformer Systems 26
4.3 Linguistic Knowledge Architectures 27
4.3.1 The Transfer Approach *27*
4.3.2 The Interlingual Approach *31*

5 LINGUISTIC REPRESENTATION AND PROCESSING 35

5.1 Types of Linguistic Knowledge 35
5.2 Syntax and Grammars 37
5.2.1 Grammars and Constituent Structure *37*
5.2.2 Grammatical Relations *40*
5.3 Semantics 41
5.4 Pragmatics and World Knowledge 43
5.5 Linguistic Information in MT Systems 45
5.6 Processing 47
5.6.1 Parsing *48*
5.6.2 Generation *49*

6 EVALUATION OF MT SYSTEMS 51

6.1 Types of Evaluation 51
6.2 Evaluating Output Quality 53
6.2.1 Intelligibility *53*
6.2.2 Accuracy *54*
6.2.3 Error Analysis *54*
6.2.4 Using the Grading Derived *55*
6.2.5 Selecting a Text Suite *55*

7 CREATING A BILINGUAL TEXT ARCHIVE 57

7.1 Obtaining the Text Data 57
7.2 A Note on the Maltese Alphabet 59

8 DEVELOPING A NATURAL LANGUAGE CHART PARSER 61

8.1 Developing a Parser for Analysis 61
8.2 Chart Parsing 61

8.3 A CKY Chart Parser 65
8.4 An Earley Style Chart Parser 68
8.5 Tokenisation of the Input 72

9 AUGMENTING THE PARSER WITH FEATURES 74

9.1 Feature Structures 74
9.2 Basic Concepts 74
9.3 Subsumption 76
9.4 Unification 77
9.5 Feature Structure Specification 78
9.6 Lexical Templates 79
9.7 Data Structures for Representing Features 81
9.8 Modification Operations 84
9.9 Implementing Unification 86
9.10 Feature Equation Satisfaction 88
9.11 Duplicating a Feature Structure 88
9.12 Extracting a Textual Representation 89
9.13 Specifying Features in the Lexicon and Grammar 91
9.14 Using Features During Parsing 93
9.15 Using JavaCC for Lexicon and Grammar Readers 94

10 SELECTING A TRANSLATION APPROACH 96

10.1 Representing Trees with Feature Structures 96
10.2 Defining a Tree-Combining Formalism 97
10.3 Feature Trees for a Transfer Formalism 100
10.4 Building a Generator Module 100

11 IMPLEMENTING THE APPROACH WITH THE AVAILABLE CORPUS 102

11.1 Creating the Lexicon 102
11.2 Creating the Grammar 106

12 ENHANCING THE SYSTEM 107

12.1 Adding Syntactic Processing to the Generator 107
12.1.1 Rules for the Maltese Article 107
12.1.2 Particles which join with the Article 108
12.1.3 The Particles 'bi' and 'fi' 109
12.2 Handling Unknown Words 109

12.3 PARTIAL PARSING 111
12.4 EXHIBITING PREFERENCES IN TRANSLATION 112
12.5 PROVIDING A VISUAL AND ON-LINE INTERFACE 113

13 EVALUATING THE SYSTEM 115

13.1 QUALITY OF OUTPUT 115
13.2 ADEQUACY EVALUATION 117
13.3 ARCHITECTURAL CONSIDERATIONS 118

14 IMPLEMENTATION DETAILS 120

14.1 LANGUAGE CHOICE AND DESIGN STRUCTURE 120
14.2 PACKAGE MT.FEATURES 121
14.2.1 Class FStructure 121
14.2.2 Class FStructureFormatException 122
14.2.3 Class FeaturePath 122
14.2.4 Class FeatureEquation 122
14.2.5 Class FeatureEquations 122
14.2.6 Class FeatureEquationConstant 122
14.2.7 Class FeatureEquationConstantSet 122
14.2.8 FeatureEquationPath 123
14.2.9 Class FeatureEquationPathSet 123
14.3 PACKAGE MT.LEXICON 123
14.3.1 Class Lexicon 123
14.3.2 Class LexiconEntry 123
14.3.3 Class LexiconReader 123
14.3.4 Class LexiconTemplate 124
14.3.5 Class LexiconTemplateList 124
14.3.6 Class LexiconTemplateNameList 124
14.4 PACKAGE MT.GRAMMAR 125
14.4.1 Class Grammar 125
14.4.2 Class GrammarParameters 125
14.4.3 Class GrammarRule 125
14.4.4 Class Grammar Reader 125
14.5 PACKAGE MT.PARSER 126
14.5.1 Class Chart 126
14.5.2 Interface ChartListener 126
14.5.3 Class Category 126

14.5.4 Class CategoryList ... *127*
14.5.5 Class Edge ... *127*
14.5.6 Class EdgeList ... *127*
14.5.7 Class LexicalEdge ... *127*
14.5.8 Class RuleEdge ... *127*
14.5.9 Class Input ... *127*
14.5.10 Class InputException ... *127*
14.6 PACKAGE MT.TRANSLATION ... 128
14.6.1 Class Translator ... *128*
14.7 PACKAGE MT.VISUAL ... 128
14.7.1 Class Translator ... *128*
14.7.2 Class TranslatorApplet ... *129*
14.7.3 Class TranslatorOptionPane ... *129*

15 USER MANUAL ... 130

15.1 RUNNING THE PROGRAM AS AN APPLICATION ... 130
15.2 RUNNING THE PROGRAM AS AN APPLET ... 130
15.3 USING THE PROGRAM ... 131

16 COMMENTS AND CONCLUSIONS ... 136

BIBLIOGRAPHY ... 140

APPENDIX A TEXT ARCHIVE SAMPLES ... 143

A.1 SAMPLE ENGLISH ENTRIES ... 143
A.2 SAMPLE MALTESE ENTRIES ... 144

APPENDIX B LEXICON AND GRAMMAR BNF ... 146

B.1 LEXICON BNF ... 146
B.2 GRAMMAR BNF ... 147

APPENDIX C BILINGUAL LEXICON SAMPLES ... 148

APPENDIX D TRANSFER GRAMMAR ... 160

APPENDIX E PARSE SAMPLES ... 166

APPENDIX F EVALUATION FORM SAMPLE ... 170

IV List of Tables

TABLE 7.1 THE MALTILEX CHARACTER CONVENTION 60
TABLE 11.1 LEXICON CATEGORY COUNTS 105
TABLE 12.1 PARTICLES THAT CAN JOIN WITH THE ARTICLE 108
TABLE 12.2 PARTICLES THAT MUST BE JOINED WITH THE ARTICLE 109
TABLE 14.1 CODE PACKAGE STRUCTURE 120
TABLE 15.1 PROGRAM FACILITIES 132

V List of Figures

FIGURE 3.1 CATEGORIES OF MT SYSTEMS 14
FIGURE 4.1 TYPICAL SOFTWARE ARCHITECTURE 24
FIGURE 4.2 ORGANISATION OF A TRANSFER SYSTEM 29
FIGURE 4.3 TRANSFER VERSUS INTERLINGUA 32
FIGURE 5.1 A SIMPLE GRAMMAR 38
FIGURE 5.2 A SIMPLE LEXICON 38
FIGURE 5.3 REPRESENTING SYNTACTIC TREES 39
FIGURE 5.4 FEATURES FOR SEMANTIC INFORMATION 42
FIGURE 5.5 A SEMANTIC TYPE HIERARCHY 42
FIGURE 5.6 A SEMANTIC NETWORK 45
FIGURE 6.1 AN OUTPUT QUALITY PROFILE 56
FIGURE 7.1 A WEATHER REPORT FORM 58
FIGURE 8.1 A COMPLETE CHART 62
FIGURE 8.2 THE CKY CHART PARSER ALGORITHM 67
FIGURE 8.3 THE *LOOKAHEAD* FUNCTION 69
FIGURE 8.4 THE *PREDICTOR* PROCEDURE 69
FIGURE 8.5 THE *COMPLETER* PROCEDURE 70
FIGURE 8.6 THE *SCANNER* PROCEDURE 70
FIGURE 8.7 THE *ADD-EDGE* PROCEDURE 71
FIGURE 8.8 THE *CHART-PARSE* FUNCTION 71
FIGURE 9.1 A COMPLEX FEATURE STRUCTURE 82
FIGURE 9.2 A COMPLEX FEATURE STRUCTURE WITH SHARED VALUES 82
FIGURE 9.3 FORWARD POINTERS 83
FIGURE 9.4 THE UNIFICATION ALGORITHM 85
FIGURE 9.5 UNIFYING DIRECTED GRAPHS 86
FIGURE 9.6 RESULT OF UNIFICATION 87
FIGURE 9.7 SATISFYING A $\langle f_1\ f_2 \ldots f_n \rangle = \langle f_1' f_2' \ldots f_m' \rangle$ CONSTRAINT 88
FIGURE 9.8 DUPLICATING A FEATURE STRUCTURE 89
FIGURE 9.9 BUILDING A MAP OF INFORMATION ON SHARED FEATURES 90
FIGURE 9.10 EXTRACTING A TEXTUAL REPRESENTATION OF A FEATURE STRUCTURE 91
FIGURE 9.11 LEXICON ENTRY FORMAT 92
FIGURE 9.12 GRAMMAR RULE FORMAT 92

FIGURE 9.13 JAVACC FOR LEXICON AND GRAMMAR READERS 95
FIGURE 10.1 REPRESENTING *N*-ARY TREES AS FEATURE STRUCTURES 97
FIGURE 11.1 USE OF LEXICAL TEMPLATES IN THE LEXICON 102
FIGURE 11.2 ENTRIES FOR *SUNNY* 103
FIGURE 11.3 CORRESPONDING FEATURE STRUCTURES FOR *SUNNY* 104
FIGURE 11.4 COLLOCATIONS AND IDIOMS IN THE LEXICON 104
FIGURE 11.5 A TRANSFER RULE IN A GRAMMAR ENTRY 106
FIGURE 13.1 EVALUATION RESULTS 116
FIGURE 13.2 SYSTEM ARCHITECTURE 118
FIGURE 15.1 APPLET TAG FOR RUNNING THE APPLICATION ON-LINE 130
FIGURE 15.2 MAIN PROGRAM WINDOW 131
FIGURE 15.3 DISPLAYING PARSING PROGRESS 133
FIGURE 15.4 DISPLAYING PARSE RESULTS 134
FIGURE 15.5 BROWSING THE LEXICON 135

1 Introduction

Natural languages are the most complete and flexible form of communication developed by humans. They also have great cultural value, in that they reflect the culture of the communities in which they are spoken, and allow access to huge amounts of information and literature to those who know the language.

However, the different languages have also created barriers for communication since time immemorial to those who do not possess such knowledge. In such cases, the only alternative to learning the language was to have the original text or speech translated by someone having a good grasp of both languages concerned, besides adequate translation skills.

The advent of computers gave great hope in the late 1940s that the translation process could be automated using machines. This hope was fuelled by the success in using machines for cryptography at the time. However, it soon became clear that translation requires an understanding of the meaning of the text, as well as detailed knowledge about the world, whereas code breaking depends only on the syntactic properties of messages.

The quality of results obtained was thus far below the expected *Fully Automated High Quality Machine Translation* (FAHQMT). However, research continued, at varying degrees, in this new field of *Machine Translation* (MT), and while there has been no fundamental breakthrough, there has been real progress. In fact, there are now dozens of MT systems in everyday use that save money over manual techniques.

Some of these systems achieve a relatively high level of output quality by operating on texts that are quite stylised and regular. One of the most successful in this respect is the TAUM METEO system, developed at the University of Montreal,

which translates weather reports from English to French. Due to its highly restricted domain, METEO is capable of translating adequately nearly all of its input, while the rest is forwarded to human translators for manual translation.

Systems applied to domains that are more open may produce less impressive results[1]. However, when a human translator is given machine-translated text as an initial guideline, he or she is able to work two to four times faster. Sometimes, even a monolingual human can post-edit the output without having to read the original. Since *Human Translation* (HT) is a slow process, and requires a high level of skill that demands a corresponding highly salary, both situations are effective in saving translation costs. This is of special important to companies having large amounts of translation requirements, such as the translation of manuals for products exported to foreign countries.

In any case, the implementation of an MT system has to handle various difficulties, some of which, such as the problems of syntactic and semantic ambiguity, are well known to the field of *Natural Language Processing* (NLP). Others are more specific to MT. The latter includes the crucial aspect of how to turn the *Source Language* (SL) text into its *Target Language* (TL) equivalent. In general, the source text is parsed into some form of representation, which is then used to generate the output text. The design of an MT system depends heavily on the kind of representation used.

There are two main approaches to handling this task. The simplest to understand is the *interlingual* approach, where an *interlingua*, a natural language free meaning representation, is used to capture only the semantic content of the input sentence. This meaning structure is then used as the basis for synthesising the TL

[1] For an example, see the sample output of the Spanish-English SPANAM system given in [21].

output. Despite its conceptual simplicity, the interlingual approach is very difficult to implement in general, but may work well in restricted domains.

A second form of design uses the *transfer* approach, in which the representation structure remains closer to the SL and retains some of its linguistic characteristics. The next step involves the transfer from this SL representation to a corresponding TL representation. Finally, the latter is used to generate the output text. Actually, the transfer and the interlingual approaches are interrelated, and, in addition to these, there exist other, perhaps older and less valid, approaches.

1.1 Aims and Objectives of the Project

The project described here concerned the implementation of an automatic MT system for translating between English and Maltese in a restricted domain based on a text archive created specifically for this purpose. The choices for representation and for the translation technique were initially left unspecified, and had to be selected based upon the characteristics of the text archive. However, in the abstract, the following main subtasks, with corresponding deliverables, were identified.

- Identification, collection and codification of bilingual texts from some suitable domain.
- Derivation of lexical and grammatical description of both languages based on the text archive, with the format depending on their intended use.
- Development of a parser able to use the grammars and lexicons in order to produce analysis structures of the SL text at varying degrees of semantic depth.
- Development of a transfer module or transfer rules, if appropriate to the translation model used.
- Development of a generator able to synthesise the TL text from representation dependent specifications.

- Specification and actualisation of a performance assessment strategy in order to quantify the quality of the output and the system.
- Development of a reasonably user friendly interface with which to utilise the tools created.

In addition, the project obviously demanded an extensive amount of research in order to get a deep understanding of MT issues and techniques, which would bear upon the choices and approaches taken in all the subtasks concerned. This especially since MT is a research field with no well-established solutions or methodologies, in which systems tend to be optimised in an ad hoc manner for the environment in which they are used.

Apart from the intrinsic interest in exploring the field of MT, the motivation for this project arises from the fact that Malta, the country of origin of the author, is bilingual, having both Maltese and English as official languages. This leads to translation requirements in several contexts. For instance, governmental documents are required to have one version in each language. Besides, the Maltese language has a very limited number of speakers (approximately 300,000 – the population of the whole country), so a device for translating from Maltese into English or other languages would be quite useful in facilitation communications. This especially since Malta has an economy based on tourism.

1.2 Outline of Work Done

Since MT was not an area of study familiar to the author, the first prerequisite for the project was to carry out an extensive survey of the field, in order to get an understanding of the main issues and techniques. This included a research on its history, which had a direct impact on the directions that MT developers took.

The first major subtask of the project itself, however, involved selecting and creating an adequate and substantially large bilingual corpus for which to create and optimise the translation system. The domain that was chosen for this archive was that

of weather reports, which are translated daily in both English and Maltese and are sufficiently restricted in syntactic and semantic content to allow higher quality translations. This choice was inspired by the highly successful METEO system described previously. Developing the archive itself involved the manual task of obtaining and inputting six months worth of weather reports in both languages in machine-readable format.

The next step involved creating a natural language parser with which to parse the SL text. For this purpose, a chart-parser capable of extracting the constituent structure of input sentences was developed. Due to the use of non-optimal algorithms and data structures, the initial implementation proved space and time inefficient when used with lexicons and grammars of any substantial size. This necessitated the implementation of another version, based upon algorithms that are more efficient and utilising several optimisations.

Since the extraction of the constituent structure of the input is not sufficient for the purposes of translation, it was necessary to augment the parser with the capability of extracting deeper representations of the source. This was achieved using PATR-II style feature structures, which are information-bearing structures capable of storing semantic information of any arbitrary depth. With these, semantic values for individual words or semantic units can be specified in the lexicon. The grammar rules, then, can be used to specify how this semantic information can be propagated to phrases that are more complex, or used to disallow invalid semantic interpretations from being built.

After considering the text archive developed, a unidirectional, transfer-based approach was adopted. For the purposes of defining transfer rules, a formalism was then developed whereby *n*-ary trees are represented and manipulated as feature structures with an appropriate format. With this, translations for individual lexical units were specified as feature trees at the lexical level, and other features were used to provide semantic interpretations depending on context. Rules in the grammar were

consequently allowed to manipulate the translation trees of sub-parses to derive a translation tree for longer spanning parses.

Thus, the analysis and transfer processes were made to operate simultaneously during the parse. What was left for the translation part was the development of a generator module. This was achieved by a tree traversal algorithm that flattens the tree into the output text, performing certain syntactic manipulations required by the target language. Having completed the translation modules, a corresponding translation lexicon and grammar were developed for the SL, based on the text archive developed.

Some other failsafe strategies were also added to the system to handle cases where a word was not found in the lexicon or when a complete parse of the input could not be obtained. For ease of use, then, the whole system was wrapped up as a visual application, and an online version accessible through a Web browser was developed as well.

The choice of implementation language was the Java Platform 1.2. This was motivated by the familiarity that the author had with the language, as well as other valuable characteristics that it has. These include the use of an object-oriented paradigm, the provision of various useful data structures such as tables and maps, as well as standard facilities for providing user friendly visual interfaces.

Evaluation of the output of the system was carried out by translating a part of the text archive that had been reserved for that purpose. Appropriate evaluation forms were created and given to a third person to mark the translations according to their perceived quality. The grades obtained were tabulated and used to derive a profile of the system performance.

1.3 Structure of Report

This report is divided into two main parts. The first part, which comprises section 2 to section 6, provides the necessary background to the translation issues in

the MT context. These sections are not to be seen in isolation, but contain essential discussions of certain MT aspects that influenced decisions in the practical parts of the task. Section 2, "*Why Translation is Difficult*", presents the main difficulties in translating between languages, whether by human or by machine. This is followed by section 3, "*Overview of MT*", which describes the main categories of MT systems and the reasons that give rise to their importance. It ends with a brief history of MT, within which some other MT concepts, such as *controlled languages* and *sublanguages*, are introduced.

Section 4, "*MT Architectures*", discusses the MT approaches to translation, including the *transfer* and *interlingual* approaches, as well other designs, such as the *transformer* systems. Section 5, "*Linguistic Representation and Processing*", describes the available techniques for the representation and processing of linguistic information required in MT systems. Finally, section 6, "*Evaluation of MT Systems*", concludes the first part of the report with a description of the methods and criteria with which MT systems are evaluated.

The second part, from section 7 to section 16, is devoted to documenting the implementation of various parts of the system. Although the presentation is given in a linear order, as with any software development task, some tasks were developed in a parallel or in an iterative fashion. The first subtask is described in section 7, "*Creating A Bilingual Text Archive*", where the choice and the creation of the corpus used for translation are described. Section 8, "*Developing a Natural Language Chart Parser*", describes the algorithms utilised for building a *chart parser*. The latter was augmented with *feature structures*, which are thoroughly described in section 9, "*Augmenting the Parser with Features*".

Section 10, "*Selecting a Translation Approach*", describes a formalism developed for a transfer approach, based on what were termed *feature trees*. The utilisation of this formalism to create the MT system is then described in section 11, "*Implementing the Approach with the Available Corpus*". Details of the final

subtasks, such as the creation of the visual interface, as well as some other enhancements, are given in section 12, "*Enhancing the System*".

In section 13, "*Evaluating the System*", the final project deliverable, an assessment of the system quality, is described. This is followed by section 14, "*Implementation Details*", giving an outline of the implementation design and structure. To complement this, section 15, "*User Manual*", describes the utilisation of the implemented artefacts, including some screenshots of the application in use. Finally, section 16, "*Comments and Conclusions*", recapitulates the accomplishments of the project, accompanied by some hindsight comments with an eye to future improvements.

The report is ended with an extensive *Bibliography* of the literature and references covered for the purposes of the project, along with a set of appendices containing material that does not fit in the main text. Appendix A, "*Text Archive Samples*", gives some specimen entries from the corpus used for the translation system. Appendix B, "*Lexicon and Grammar BNF*" describes the format used for defining lexicons and grammars with which the language parser mentioned previously operates. Samples of the actual lexicon and the complete grammar used for translation purposes are respectively given in Appendix C, "*Bilingual Lexicon Samples*" and Appendix D, "*Transfer Grammar*". Appendix E, "*Parse Samples*", give some samples of analysis structures on which the system operates. Finally, Appendix F, "*Evaluation Form Sample*", shows the form used for evaluation purposes, which also includes examples of the system output.

2 Why Translation is Difficult

Translation, whether performed by human or by machine, is difficult for two main reasons. Firstly, it requires in-depth understanding of both the meaning and the context of the source text. Secondly, it requires the ability to relate all this with only the linguistic mechanisms provided by the target language. For MT, the first issue poses major difficulties as to the representation and capturing of meaning. The second is even worse in that in general it can be a genuinely creative act, which is quite beyond the current capabilities of computers.

This section discusses some particular problems, which fall in the categories above, that need to be considered when building an MT system. Many of these are relatively well known in Computational Linguistics and NLP, and several representation and processing techniques have already been developed in order to tackle them. The latter are discussed in section 4. The problems are here divided into four headings[2].

1. Problems of ambiguity.
2. Problems that arise from structural and lexical differences between languages.
3. Handling multiword units like idioms and collocations.
4. Preserving meaning and interpretation in translation.

[2] The first three headings of this classification, as well as some of the corresponding discussions, are taken from [2], where they are corroborated in considerable more detail. The fourth heading is based on a distinction found in several references.

These are not the only kinds of problem that make MT hard. One other is the sheer size of the undertaking, as indicated by the number of rules and dictionary entries that a realistic system will need. Besides, the grammar for many constructions is poorly understood, because how they should be represented or what rules should be used to describe them is not clear. This is the case even for English, which has been extensively studied, and is an even worse problem for other languages. Besides, even where there is a reasonable description of a phenomenon or construction, producing a formal description for use in MT raises non-trivial problems.

2.1 Ambiguity

Ambiguity is a pervasive phenomenon in human languages, and poses a central problem to all MT systems. It can occur both at the word and at the phrasal level. When a word has more than one meaning, it is said to be lexically ambiguous. When a phrase or sentence can have more than one structure, it is said to be structurally ambiguous.

Knowledge of the language syntax can be used to filter out ambiguity concerning the lexical category of words. However, words can have several meanings even within the same part of speech. For instance, the noun *pen* can refer to a writing device, an enclosure for cattle or sheep, a female swan and even a colloquial contracted form of *penitentiary*. To get the machine to pick out the right interpretation in a given context, it must be given information about meaning, and even world knowledge, as in the classic example, *The sheep are in the pen*.

The other form of ambiguity is structural ambiguity. One classic example of this is the following sentence.[3]

[3] This example and its discussion are adopted from “Machine Translation: A Problem of Linguistic Engineering or of Cognitive Modelling?”, [13], by Patrick Shann.

The tree in the garden which has cones is pretty.

The relative clause *which has cones* can be attached to either *tree* or *garden*. For semantic reasons it is clear this should be attached to *tree*, since it makes more sense to say that a tree, rather than a garden, has cones. However, simply changing the clause to *which has pansies* completely reverses the situation, since it is now more likely to be dependent on *garden*. This ambiguity becomes a translation problem as soon as one attempts to translates *has* in the sentence above from a language that has does not distinguish gender in verbs, such as English, to one which does, such as Maltese.

It is very hard to find words that are not ambiguous in at least two ways. Sentences that are, out of context, several ways ambiguous are the rule rather than the exception. This is not only problematic because some of the alternatives are unintended and thus represent wrong interpretations, but because ambiguities can multiply in a combinatorial way. One need only consider the way the prepositional phrases in the classic sentence *I saw the man on the hill with a telescope* contribute to an increasing number of possible interpretations. The number of analyses can be problematic in practice, since a system may have to consider all of them, rejecting all but one.

2.2 Lexical and Structural Mismatches

There are other difficult problems, besides ambiguity, which are more directly related to translation. Some of these have to do with lexical differences between languages. Languages seem to classify the world in different ways, resulting in differences as to which concepts are expressed by single words and which not. For instance, verbs conjugated from the infinitive *to know* in English can be used both for expressing knowledge of facts and for expressing knowledge of things or persons. By contrast, Italian differentiates between the two, using the verbs conjugated from the infinitive *sapere* for the former, and from *conoscere* for the latter.

Other problems arise due to structural mismatches, that is, where different languages use different structures for the same purpose and the same structure for different purposes. A simple example of the latter situation can be demonstrated by the use of nouns qualified by the pronoun *this* in both English and Maltese. In translating the phrase *this thesis* into Maltese, one needs to add the article to the noun, resulting in *din it-teżi*. If the same structure where used unmodified, the result would be *din teżi*, which actually means the completely different *this is a thesis*.

2.3 Translating Idioms and Collocations

Idioms can be described as expressions whose meaning cannot be completely understood from the meanings of the component parts. For instance, the idiomatic phrase *to kick the bucket* actually means *to die*. The problem with these in MT is that it is not usually possible to translate them using the normal rules. While there are some exceptions, applying the rules for normal words in order to translate idioms will generally result in nonsense. Instead, idioms need to be treated as single units in translation.

This is not so simple since idioms cannot simply be considered as special words that happen to have spaces in between. Apart from some non-representative collocations, such as *in fact* or *in view of*, idioms do not generally have a fixed form. Typically, they can take various forms of inflections and different tenses, and they may have adjectival modifiers embedded for stylistic purposes. One needs only consider all the possible conjugations of *to kick* in the phrase above. In addition, idioms are typically ambiguous between a literal and an idiomatic interpretation. For instance, *to kick the bucket* could actually refer to the action of giving a kick to a bucket. Thus, the recognition and interpretation of idioms require considerably sophisticated syntactic and semantic analysis to be handled properly.

2.4 Meaning and Interpretation

A major distinction, not always made, concerning translation is that between meaning and interpretation. A high quality translation, in fact, is not just one that preserves meaning, but one that preserves interpretation, besides, of course, being grammatically, and perhaps stylistically, correct.

This can be easily illustrated by considering the translation of the French sentence '*O'u voulez-vous que je me mette?*' This could be, more or less literally, translated as '*Where do you want me to put myself?*' However, a translator would probably put it into some more colloquial English form, such as '*Where do you want me to sit?*' or '*Where do you want me to stand?*', depending on the context in which it is uttered. Thus, information is added to the English rendering in order to make it sound natural, despite the fact that the sentence '*Where do you want me to put myself?*' has the same meaning as the French original.

It would also be possible to get the right effect by deleting information, as in '*Where do you want me?*', but this also changes the meaning. What remains invariant under translation is not the meaning, but the interpretation, that is, the response that the source text is intended to evoke in a reader. However, this preservation of interpretation requires considerable skill to achieve, even from a human translator, and it is not at all obvious as to how the intuition required can be explained, let alone formalised for MT purposes.

3 Overview of MT

3.1 Categories of MT Systems

There are three broad categories of computerised translation tools. The differences between them hinge on how ambitious the systems are intended to be. These categories can be described as *Machine Translation* (MT), *Machine Aided Translation* (MAT) and *Terminology Databanks* (TD). The relationship between these is shown in Figure 3.1.

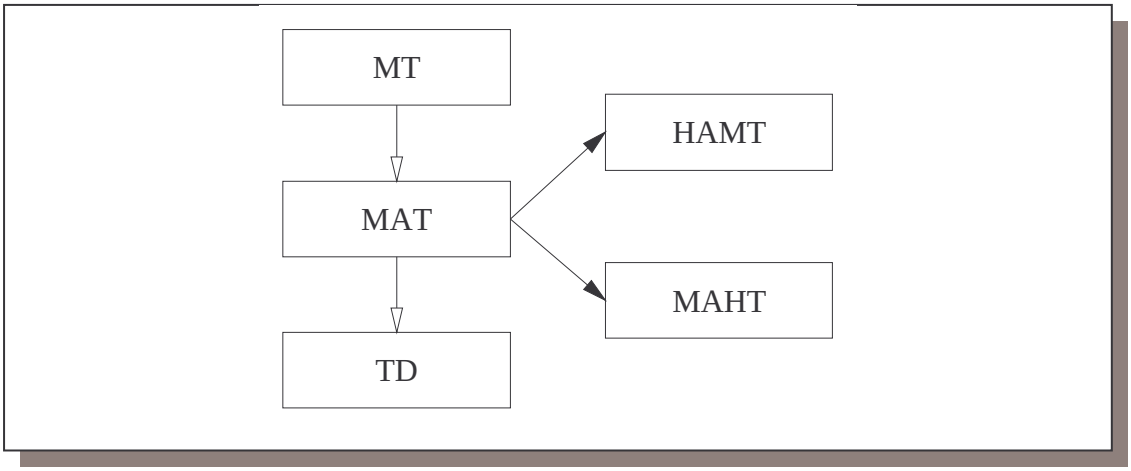


Figure 3.1 Categories of MT Systems

Fully automatic MT systems are intended to perform translation without human intervention. This does not rule out pre-processing, unless it involves marking phrase boundaries, resolving parts-of-speech ambiguities, and other things which help the analysis process. Nor does it rule out post-editing, since this is normally done for human translations anyway. However, an MT system is solely responsible for the complete translation process from input of the source text to output of the target text without human assistance. Obviously, MT occupies the top range on the scale of computer translation ambition.

MAT is what is usually meant by computer translation, since most systems require human intervention at some stage. MAT systems fall into two subgroups, *Human Assisted Machine Translation* (HAMT) and *Machine Assisted Human Translation* (MAHT). These occupy successively lower ranges on the scale of computer translation ambition. HAMT refers to a system wherein the computer is responsible for producing the translation itself, but may interact with a human monitor at many stages along the way in order to resolve ambiguities as they arise. For example, it might ask the human to disambiguate the part of speech or meaning of a word, or to indicate where to attach a phrase. It might also allow the choice of a translation for a word or phrase from among several candidates discovered in the dictionary of the system. The major problem in HAMT, aside from the difficulty of identifying the appropriate places for human interaction and the formalisation of the questions to put to the human being, is the inefficiency that it introduces. The translation process is slowed down to such an extent that it may not be considered a practical alternative for a general MT system.

MAHT refers to a system wherein the human is responsible for producing the translation, but may interact with the system in certain situations. For example, this can happen by requesting assistance in searching through a local dictionary or thesaurus, accessing a remote TD retrieving examples of the usage of a word or phrase, or performing word processing functions like formatting. With MAHT, the system does not need help, but is making help available instead.

Terminology Databanks are the least ambitious systems because access frequently is not made during a translation task but is usually performed before human translation. Indeed the data bank may not be accessible on-line at all, but may be limited to the production of printed subject-area glossaries. A TD offers access to technical terminology, but usually not to common words, since the user will already know these. The chief advantage of a TD is not the fact that it is automated, but that it is up to date. Technical terminology is constantly changing and published dictionaries are essentially obsolete by the time they are available. In this, a TD can take

advantage of the continual contribution of users to its contents, in the form of additions and updates.

Due to the objectives of the project concerned, this report will be mostly related to automatic MT systems, or possibly HAMT systems, the exact distinction being slightly blurred. These can be further subcategorised according to their directionality of translation and the number of languages between which they can translate. Systems capable of going only from the SL to the TL, but not vice-versa, are termed *unidirectional*, while those that can reverse the process are called *bidirectional*. As an example, the METEO system described in section 3.3.5 was initially developed as unidirectional system translating from English to French only, although it was later modified to translate from French to English as well. However, it is only capable of handling this language pair, and is thus referred to as a *bilingual* system. Systems capable of handling a larger number of language pairs are called *multilingual*. Examples of the latter include SYSTRAN (see section 3.3.4) for which several language pairs have been developed ([24]), as well as the EUROTRA project (section 3.3.6) that was meant to handle translation between seven languages. Bidirectional, multilingual systems are obviously desirable, but in practice, the design of a system, discussed in detail in section 4, strongly influences the ease with which a system can handle bidirectional translation and multiple language pairs.

3.2 Motivation and Aims for MT

MT has important social, political, commercial, scientific, intellectual and philosophical implications[4]. The social or political importance of translation can be seen in nations with more than one native tongue. In this case, the only alternative to widespread translation is the use of some ‘lingua franca’. Such use may cause other languages to be relegated to second class, with the danger of ultimately falling into

[4] This classification and some of the corresponding discussions are taken from [2].

disuse and fading out. Since a language is intrinsically related to the community that speaks it, this is not a problem to taken lightly. Such a situation can in fact be felt in the Maltese Islands, where, while both English and Maltese are the official languages of the country, there is an increasing tendency to use the English language. Some of the reasons for this are that it is more expressive, it is necessary for communication with foreigners and it has a somewhat higher status associated with it.

Related factors give rise to the commercial importance of MT. These are mostly evident in companies that sell products in the international market, and generally have to translate the accompanying documents, such as manuals, into the preferred language of their target customers. In such a case, translated texts must not only be right but also clear. Qualified human translators are hard to find, expensive, and slow, not averaging more than 4-6 pages per day on difficult material. Worse, in many technical fields there is a distinct shortage of qualified human translators.

Scientifically, MT is interesting because it is an obvious application and testing grounding in Computer Science, Artificial Intelligence and Linguistics. Some of the most important developments of these fields began in MT. For example, the origins of Prolog can be found in the Q-Systems language, which was originally developed for MT and used in the METEO system. Besides, MT also affects indirectly scientific research, and other forms of intelligence gathering, by alleviating the problem of information acquisition. Masses of data are available to sift through, but there is not enough time, money or incentive to translate carefully every document by normal means.

Finally, MT is interesting philosophically in that it attempts to automate an activity that requires the full range of human knowledge. Issues such as language learning and understanding, meaning, sense, reference and truth all play part of this field. In a sense, the automation of translation is an attempt to automate thinking, so that success in this area would have implications as to the extent in which a machine can perform as a human.

As regards the main objectives of the MT research field, obtaining '*Fully Automatic High Quality Machine Translation*', was the initial goal of MT and still remains the Holy Grail of the subject. However, this does not mean that limited systems are not of any use. In view of the practical importance of the information acquisition and dissemination problems described above, having a system capable of producing a rough, if not perfect, translation, is a valuable thing in itself. The reduction in time and costs obtained with MT may well justify crude or draft translations, which in certain cases, may be all that is required.

3.3 A Brief History of MT[5]

3.3.1 Birth and Early History

The concept of mechanical translation is quite – perhaps centuries – old. However, the birth of modern day MT can be traced down to a letter sent by Warren Weaver, director of the Natural Sciences Division of the Rockefeller Foundation, to Norbert Wiener, the founder of cybernetics, on March 4, 1947. Motivated by the communication problems within the recently founded UNESCO, and inspired by the code-breaking capabilities of the newly developed US computers, Weaver discussed the possibility of translating using computers, treating translation as a problem in cryptography:

"*When I look at an article in Russian, I say 'This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode.*"

[5] Some of the earliest historical facts presented here are taken from the detailed account given in "Fifty Years of the Computer and Translation", [3], by John Hutchins. Others have been adopted from "Early History of Machine Translation", [13], by B. Buchmann.

This letter, along with some other early MT related work, was published in a memorandum by Weaver called '*Translation*', in 1949, which effectively launched MT research in the US.

3.3.2 The Georgetown Project

One important MT project at this time was the one carried out at Georgetown University in 1954. Targeting various US government agencies for funding, a small group of programmers and linguists engaged in the creation of a system that could translate from Russian into English. The system was designed to translate fort-nine selected sentences with a vocabulary of two hundred and fifty words, using only six syntactic rules. The sentences were of the simple declarative type with a very restricted syntactic nature, limited verb tenses and very few prepositions.

The translation technique used was simple, based on local rearrangement of words depending on the word to the right or to the left of the one being considered for translation. The system was driven by its dictionary, where the rules for rearrangement were specified by means of codes.

Despite its feeble linguistic basis, the system could achieve the desired translation due to the highly closed domain of its input. The experiment was heralded as an unqualified success, and attracted funding to the growing field of MT.

3.3.3 The ALPAC Report

However, in 1964, after having funded MT research for ten years, the US National Academy of Sciences set up the Automatic Language Processing Advisory Committee (ALPAC), to investigate the results obtained so far. The result of the investigation report was that the current state of MT was not useful, largely due to the lack of linguistic knowledge employed in current systems, and there was no immediate or predictable prospect of it being so. It also went on to recommend that

the funding should be channelled to more long-term fundamental research in theoretical and computational linguistics.

Consequently, MT funding and projects were practically stopped and the whole field was thrown into bad light for almost a decade. However, the newly available funds in the fields of theoretical and computational linguistics ultimately benefited MT in that the use of the latter would form the basis of a second generation of systems. Eventually, though, several factors, such as the growing demand for MT, contributed to a renewal of interest in the field.

3.3.4 Renewed Interest: SYSTRAN and Controlled Languages

In addition, practical results were starting to be obtained. One of the earliest successful systems, and one which is still marketed to date ([24]), was SYSTRAN, a product established in 1968 by Dr Peter Toma, a linguist researcher who had also been involved in the Georgetown University project. This first generation system was often criticised for being ad hoc and lacking linguistic and semantic knowledge[6]. In fact, it achieved a reasonable level of quality in translation using a very large database of rules meant to handle all special situations.

Despite this, the company that produced it obtained several contracts for developing different language pairs. Its customers to date include the US Air Force, the US National Air Intelligence Centre, NASA, the Ford Motor Company, as well as Xerox. The latter pioneered the concept of the usage of controlled languages in order to achieve higher quality translations. The company developed a restricted input language for this purpose, and had its manuals rewritten accordingly before translating with SYSTRAN. This allowed a higher output quality rate to be obtained, with the pleasant side effect that the original manuals became more readable.

[6] See “SYSTRAN”, [13], by Peter J. Wheeler.

3.3.5 A Real MT Success: METEO and Sublanguages

Another highly successful system was METEO, developed by the TAUM group at the University of Montreal in 1977, and still used to date. This system was carefully tailored to the translation of weather reports from English into French at the Canadian Meteorological Centre. The consistent style and limited vocabulary used in these reports allowed the system to be fine-tuned to the extent that it was capable of correctly translating ninety to ninety five percent of the input. In addition, it could recognise its incapability to handle the rest of the other inputs, and relay these to human translators instead of attempting a translation itself. Actually, most of these failures were due to noise in the input, misspellings or words missing from the dictionary, although some were also caused by its inability to handle certain linguistic constructs.

The spectacular success of METEO, which was based on the transfer technique, was due to the restricted nature of the texts on which it worked. This idea led to the notion of what might be termed *Sublanguage*[7] MT, where systems are developed for a suitably restricted linguistic domain. This is particularly attractive since it simplifies the system design for the developer and provides better quality translation to the user, without requiring the artificial modification of the input text, as in the case of controlled languages. The downside is that such systems are not easily extensible, although other suitably restricted domains may be found, such as the languages used in stockmarket reports, instruction manuals and other forms of informative documentation.

3.3.6 Increase in Use and Research: EUROTRA

Other systems, such as ALPS, LOGOS, WEIDNER, GETA and SUSY, continued to be developed into the 1980s, either for commercial or research purposes.

[7] The term is taken from [2].

In Japan, work in MT was taken up in various sectors, and was seen as a main component in the fifth-generation project. Such systems, whose capabilities went well beyond that of earlier ones, started to be used by governments, businesses and the industry in increasing numbers. In fact, in 1984, approximately half a million pages of text were translated by machine.

One such MT project was EUROTRA[8], started in late 1977 by various universities in Europe, with the aim to provide translation facilities that met the requirements of the European Community. One major requirement was that the system had to be multilingual, having to handle translation between at least seven languages. In addition, the management of the project had to be decentralised, taking into account that linguists and developers had to work in separate countries while still co-ordinating their efforts with each other. In view of this, an attempt was made to use as much as possible an interlingual approach to the design of the system, in order to minimise the number of modules required[9]. It would also facilitate the tasks of linguists, since their only concern would be how to map the language they know best to and from the interlingua, without requiring interaction with other linguists, possibly working in another country. Unfortunately, the project collapsed under its own administration weight, but it could be considered as exemplary of a unified approach to MT.

3.3.7 Modern Day MT: Verbmobil

This trend of increased use and research continued into the 1990s. In addition, the increase in computer capabilities and decrease in price allowed the introduction of professional or semi-professional MT systems and translation tools capable of running on desktop PCs. A recent review of such a set of programs can be found in [25].

[8] For further details, refer to "EUROTRA", [13], by M. King and S Perschke.

[9] See section 4.3.2 for an explanation of why the interlingual approach theoretically requires less system modules.

These often integrate with common word processors, or even with the windowing environment in which they are used. The impact of the Internet can also be felt, and some systems have special facilities for the web. Some are even available on line in order to translate Web pages ([24]).

More recently, research in MT has turned to the development of speech-to-speech systems, in which a device is spoken into in one language, and a spoken translation is obtained as output. Obviously, such a system involves many more difficulties than one operating on texts, having to handle the difference pronunciation inflections of different users, the problem of noise and other issues related to phonology.

One such research project is Verbmobil ([25]), whose aim is to develop a system capable of recognising spoken language and translating it into spoken English. The system processes spontaneous-speech irrespective of the speaker. However, it has a restricted domain, limited to appointment-negotiation conversations. The first prototype of the system, The Verbmobil Demonstrator, was unveiled in 1995. It recognises spoken German with a vocabulary of 1292 words, and can analyse and translate the input, and utter the English translation in real time. A second version, The Verbmobil Research Prototype, with a vocabulary of 2,500 words, was presented in 1997. The latter additionally recognises Japanese utterances with a vocabulary of 400 words.

4 MT Architectures

4.1 A Typical Software Architecture

At the most general level, an MT system can be seen as a black box with one entry and one exit. It receives the SL text to be translated as input, and outputs the TL equivalent text. As regards the internal components within that black box, however, there is no standard characterisation. One possibility is to view a system as comprising three parts[10]: a translation process or engine, a short-term memory and a long-term memory. Figure 4.1 shows this decomposition, where the arrows represent the flow of data inside the system.

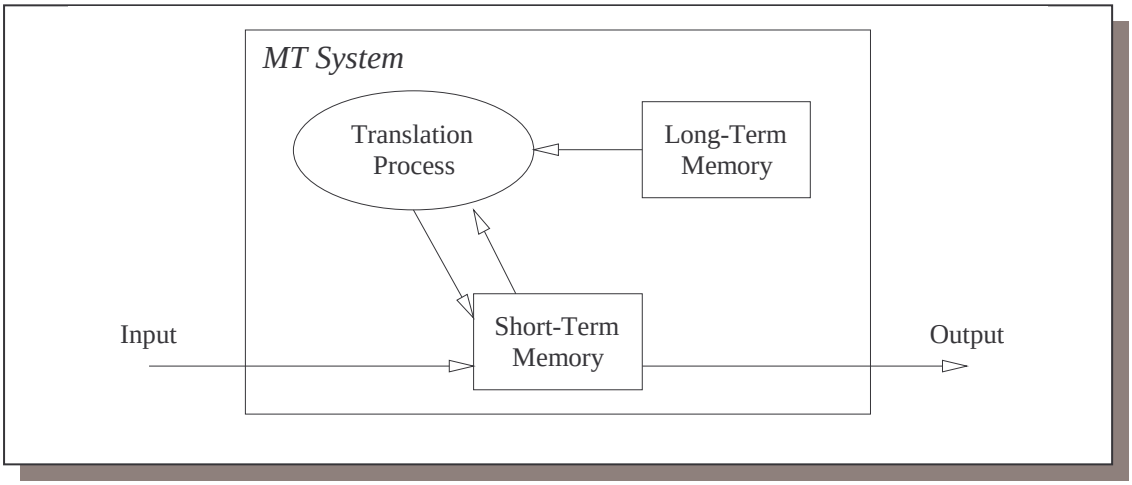


Figure 4.1 Typical Software Architecture

The translation process is the heart of the MT system. Its main functionality involves applying its own knowledge and knowledge saved in the long-term memory,

[10] This decomposition and following discussion is adopted from "A Brief Look at a Typical Software Architecture", [13], by J-L. Cochard.

to the input, in order to create the short-term memory. This short-term memory is then used to derive the translation of the given text. The actual processing operation and the kind of information stored in both types of memory are largely dependent on the design of the system.

4.1.1 The Long Term Memory

The long-term memory is used to store knowledge necessary for the translation, including linguistic information about the languages concerned, as well as information related to the mapping from one language to another. The exact contents will vary according to the way the translation process operates, the linguistic information embedded in the latter, as well as the overall linguistic depth of the system. In general, it will have a grammar and a dictionary (which is also called a lexicon) for one or both of languages involved, as well as a transfer grammar or some other mapping specification. The long-term memory is usually fixed by the system developer, but some parts of it may be reserved for information added by the user, such as new entries in the dictionary.

4.1.2 The Short Term Memory

The short-term memory is the working area of the translation process. It is here that structural descriptions of the source text and target language translation are held. Such structural descriptions will vary in complexity, again according the operational characteristics of the translation process. The type of structures used may be application specific, although some are typical of NLP applications. These include trees, features and charts, which are used to store the types of linguistic information described in section 4.

4.1.3 The Translation Process

The translation process largely determines the structure of all components. It is usual to classify MT systems according to this architecture, depending on the overall

processing organisation and the abstract arrangement of its various modules. The most common taxonomy is based on the technique used in going from the SL to the TL, whether *direct* or *indirect*. First generation MT systems were based on direct or *transformer* architectures, which are still found in many of the well-established commercial MT systems. Indirect or *Linguist Knowledge* architectures form the basis of second-generation systems, and have dominated MT research for several years. Both architectures are discussed in detail in the following sections[11].

4.2 Transformer Systems

The main idea behind transformer systems is that input sentences can be transformed into output sentences by carrying out the simplest possible parse, replacing source words with their target language equivalents as specified in a bilingual dictionary. The system might not even worry about getting a full and complete parse for the whole thing, and the output might just be a list of words with their parts of speech, rather than a complete representation.

The TL words are then rearranged to suit the rules of the target language, using a package of transformation rules working, where necessary, on information provided by the parsing process. Most of the translation competence of the engine lies in these rules. In a sense, a transformer system has some knowledge of the comparative grammar of the language, of what makes the one structurally different from the other. However, transformers generally do not have independent linguistic knowledge of the target language because they do not have an independent grammar for that language.

[11] Most of the following discussions, especially those on Transformer architectures, are taken from [2]. The classification terms *Transformer* and *Linguistic Knowledge*, as opposed to the more traditional *Direct* and *Indirect*, are taken from the same reference.

The architecture is highly robust in that it does not break down or stop when it encounters input that contains unknown words or unknown grammatical constructions. Typically, these are passed as output without any translation. However, the price for this robustness is paid in terms of output quality. If the system has no detailed knowledge of the target language grammar, there is no guarantee that the transformed input sentence is actually a grammatical sentence in the target language. In the worst case, the system can work rather badly, being prone to produce output that is simply unacceptable in the target language.

Transformer systems are also limited as regards extensibility. Firstly, the different translation rules interacting with each other may be hard to understand, and thus to extend or modify. Secondly, systems are usually designed to run in one direction only, say, from English to Maltese. To go in the other direction, the transformer rules have to be more or less completely rewritten. Since the latter include bilingual dictionary rules, this can mean that the system has to be supplied with two bilingual dictionaries in order to go in both directions. This is inefficient since apart from the differences in their directionality, the dictionaries contain much the same information.

Additionally, the architecture links single language pairs only, and is not conducive to the development of genuinely multilingual systems. If the system is made to translate into another target language, then most of the transformer rules will have to be rewritten again. Thus, grammatical knowledge of English and of Maltese that is built into an English-Maltese system cannot be transferred to an English-Italian or a Maltese-French system.

4.3 Linguistic Knowledge Architectures

4.3.1 The Transfer Approach

The main idea behind Linguistic Knowledge (LK) systems is that high quality MT requires linguistic knowledge of the source and the target languages as well as

knowledge on the differences between them. For these purposes, such a system requires the following components.

1. Substantial grammars and dictionaries of both the SL and the TL. These grammars are used by parsers to analyse sentences in each language into representations that show their underlying structure, and by generators to produce output sentences from such representations.

2. A comparative grammar, which is used to relate every source sentence representation to some corresponding TL representation. This representation forms the basis for generating a TL translation.

The generally agreed upon standard model for utilising this information is referred to as the *transfer* model, and is comprised of three sequential stages, described below.

1. **Analysis**: During this first phase, the SL dictionary and grammar are used in conjunction with a parser to analyse the input and construct an intermediate source structure representation.

2. **Transfer**: This second phase involves changing the source structure representing the input sentence into the corresponding target structure representation for the output sentence. The comparative grammar, or some other form of transfer rules, and possibly a linguistic knowledge base or bilingual dictionary[12], are used during this step.

3. **Generation**: This third and final step is the converse of analysis, and involves the transformation of the target structure representation into the TL translation using the TL grammar and dictionary.

[12] See section 5.5.

This whole process is described diagrammatically in Figure 4.2.

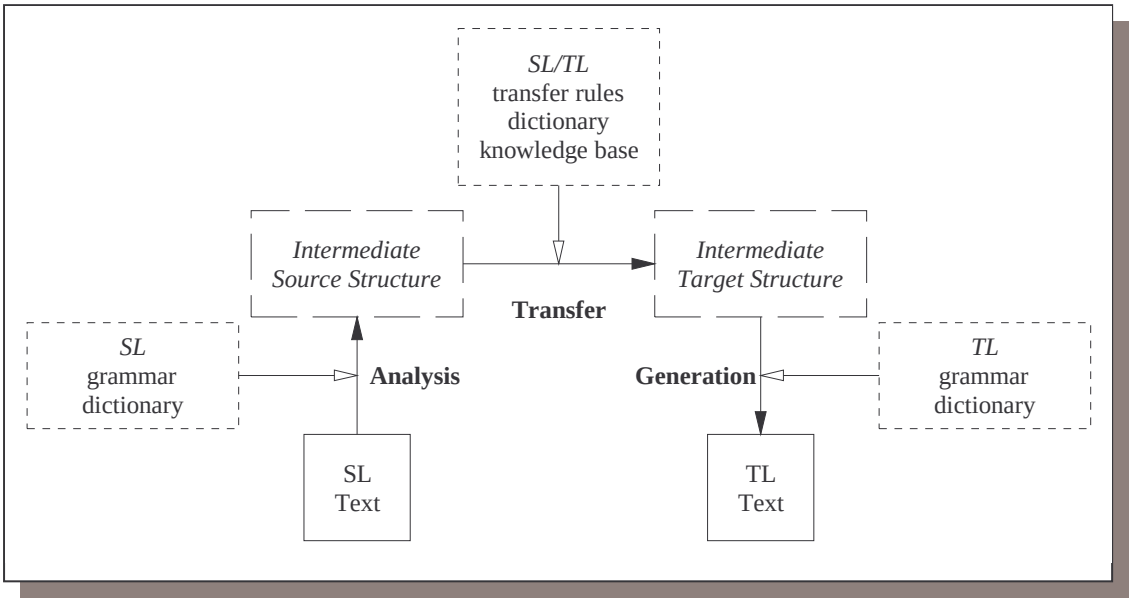


Figure 4.2 Organisation of a Transfer System

The three stages may be further subdivided into steps treating things such as morphology, surface and deeper syntactic relations as well as semantic relations, and may include different devices to account for pragmatics and stylistics. The basic text for translation is usually no more than one sentence.

In principle, the analysis and generation phases are only concerned with problems of a specific language, while the transfer component addresses the bilingual problems specific to translation. However, this separation is not always possible or necessarily sufficient. For example, monolingual criteria applied during analysis may not be sufficient to perform word-sense disambiguation for the transfer phase. The verb *to know* may seem unambiguous in analysis but the lexical equivalent in Italian can be either *conoscere* or *sapere* and a choice must be made during transfer.

Thus, the transfer component, which is central the whole system, may also be one of the most complicated to develop. The complexity of this component is inversely proportional to two other characteristics.

1. The depth at which the SL and the TL structural descriptions are stated, that is, the degree to which they manage to abstract away from language-specific constraints.

2. The degree of relatedness between the SL and the TL. The more these are lexically and structurally similar, the less need there will be for complex transfer rules.

LK architectures have some very positive characteristics as compared to transformer systems. The fact that a proper TL grammar is being used means that the output of such a system is far more likely to be grammatically correct than that of a transformer system. In addition, since the comparative grammar completely specifies a relationship between representations of two languages, translation quality will tend to be more reliable. The symmetry of the design also means that the whole engine should be reversible, at least in theory. In practice, few translation engines are reversible, since some rules that are necessary for correct translation in one direction could cause problems if the process were reversed. For example, if a rule involving deletion of a structure is reversed, there may be no indication as to how that structure can actually be created in the first place.

In principle, the transfer model allows for any number of languages to be added to a system. In practice, the choice of what information is represented will limit the possible target languages. The extensibility of a system will depend, in part, on the relative independence of the transfer phase and the extent to which it influences and interacts with analysis and generation. Many such systems are developed based on only one language pair, and the degree to which they are open to the introduction of other languages remains an open question.

One downside of the approach is due to the well-known fact that computational grammars invariably fail to capture completely natural language structure. Hence, cases arise where the system fails to recognise some complicated grammatical input. Especially for commercial purposes, pure LK engines have to be

supplemented with various coping strategies. For example, if they cannot parse a particular sentence completely, then they at least should be able to use information on those parts of the sentence parsed, and perhaps guess how those well-parsed bits might be fitted together.

LK systems are clearly superior in principle to transformers. However, such systems require considerable development effort while some commercial transformer systems that have undergone extensive revision, refinement and updating over the years can achieve a good overall performance. An example of the latter is the SYSTRAN system, described in section 3.3.4.

4.3.2 The Interlingual Approach

Comparative grammars, and hence the transfer component, become simpler as linguistic analysis goes deeper and as the representations become more abstract. In fact, a major object of MT research is to define a level of analysis that is so deep that the comparative grammar component disappears completely. Given such a level of representation, the output of analysis could be the direct input to the target synthesis component. Representations at such a level would have to capture only the underlying meaning of the input, and be completely independent of natural language specific surface realisation. For these reasons, such a representation is called an *interlingua* and a system that uses it is called *interlingual*. Figure 4.3 illustrates this idea, and clearly shows how the difference between the transfer approach and the interlingual approach is a matter of degree, rather than of absolute distinction.

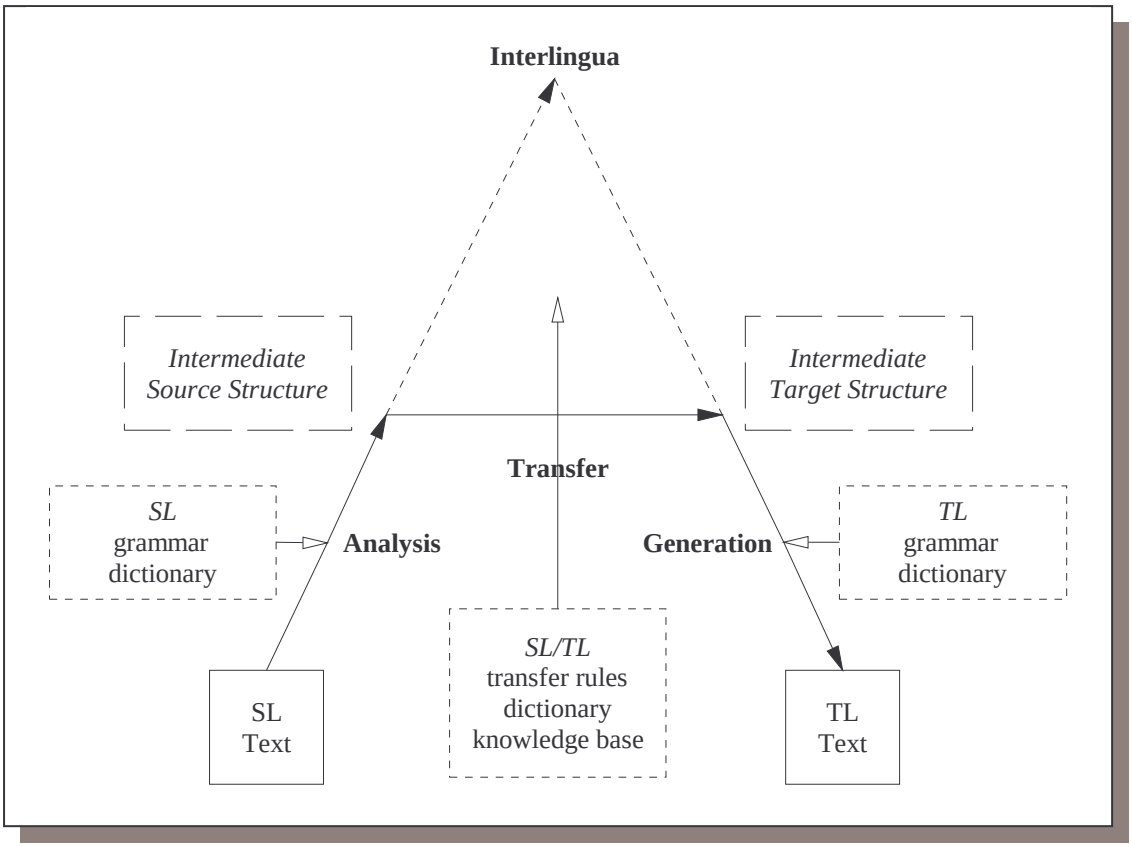


Figure 4.3 Transfer versus Interlingua

This interlingual architecture is conceptually, and deceptively, simple and has a number of attractions to it. Firstly, from a purely intellectual or scientific point of view, the idea of an interlingua is interesting and exciting. Secondly, from a practical viewpoint, an interlingual system promises to be much easier to extend by adding new language pairs, than a transfer or a transformer system. Provided the interlingua is properly designed, it should be possible to add a new language to a system simply by adding analysis and generation components for it. If, for example, a multilingual system has to handle n languages, n components will be required to translate these into the interlingua, and n to translate from it, for a total of $2n$. By contrast, a transfer system needs not only analysis and generation modules, but also transfer components into all the other languages involved in the system. Providing the same facilities with the transfer approach, then, requires a separate transfer module to translate in each

direction for every pair of languages, for a total of $n(n-1)$ components. (The case for a transformer system is even worse, since the $n(n-1)$ figure applies for the whole of the system.) For example, in the case of EUROTRA (see section 3.3.6), which aimed to translate between seven languages, the interlingual approach promised to require only fourteen components ($2\times7=14$), while the transfer approach would have required forty-two ($7\times6=42$).

Unfortunately, this old idea of a universal semantic language suffers from deep philosophical and technical problems that have not been resolved to date. Firstly, defining a complete, language independent representation for words and structure is quite difficult. Choosing the terms that semantically represent the words in natural languages is itself an arduous task, considering that each language carves up the world in a different manner, depending on the cultural environment in which it was developed. For example, Eskimos have innumerable words to describe particular types of snow, but have no generic term for it[13]. Thus, choosing the vocabulary for the interlingua will entail either a bias towards the conceptualisation of some particular language, or a multiplication of all distinctions found in any language. The former limits the validity of the interlingua, while the latter may cause the number of terms to become unmanageable, besides introducing ambiguity of choice during the generation phase.

Another problem with the interlingua is that it may not be actually desirable to make do with all the structure of the input and retain only the meaning. This can be shown by considering the translation of the following sentences.

1. I finished my thesis yesterday.

2. It was my thesis that I finished yesterday.

[13] Taken from "Languages", *New Age Encyclopaedia*, Bay Books Pty Ltd, 1983.

3. It was yesterday that I finished my thesis.

Each of these expresses the same facts, namely, that *I finished my thesis*, and that *I finished it yesterday*. However, they are not equivalent, since each has specific contexts within which it is better suited, and others in which the alternatives would make more sense. Effectively, the differences are ones of structure rather than meaning, so an interlingua which neutralises these structural aspects complicates considerably the task of generation, discourse structures being among the least understood phenomena of natural languages.

Given these difficulties, interlinguas in the sense described above are more popular as a basis for theoretical research in MT rather than for full-scale commercial development, although they can be used successfully for sublanguage systems. MT applications of a realistic scale have to use relatively shallow, language-dependent structural descriptions of natural language texts. Nonetheless, given the promise of better quality translation, the search for and application of increasingly deep analyses is a major goal of MT research.

5 Linguistic Representation and Processing

5.1 Types of Linguistic Knowledge

One major issue in the development of an MT system is the recognition of the linguistic knowledge required, and the encoding of this so as to make it amenable to processing by machine. The field of NLP has gone a long in identifying the major issues, proposing representation forms and processing techniques. However, there are still quite a number of unsolved linguistic problems in tackling these points.

Exactly how human translators perform their task is not completely understood, but it is possible to identify some types of knowledge that they require for their task[14]. They obviously need to know the SL and the TL, as well as the correspondences between them. At the simplest level, this involves knowing the translation of the individual words. Knowledge of the subject matter, as well as general knowledge and common sense, is required to understand the intended meaning. In addition, competent translators require knowledge of the culture, customs, social conventions and expectations of speakers of the target language, in order to convey the interpretation of the source, as explained in section 2.4.

Unfortunately, no one really knows how to handle this last kind of knowledge. Considerable effort, however, has been devoted to formalise the other types. For the purpose of natural language understanding systems, and hence for MT, it is possible to distinguish the following forms of linguistic knowledge[15]:

[14] Adapted from [2].

[15] Adapted from [1].

- **Phonological Knowledge**: This concerns the sound system of a language, allowing, for example, guessing the pronunciation of new words. In the MT context, it is mostly relevant to speech-to-speech systems.
- **Morphological Knowledge**: This relates to how words can be constructed from more basic meaning units, such as the derivation of the noun '*failure*' from the verb '*fail*' through the application of a suffix.
- **Syntactic Knowledge**: This specifies how sentences and phrases in a language can be made up from individual words, and the structural role that words or phrases play in larger phrases or sentences.
- **Semantic Knowledge**: This dictates what words and phrases mean, and how the meaning of a phrase can be construed from the meaning of its component words, independently of the context in which they are used. By contrast the following three types of knowledge are all concerned with the influence of local or global context on meaning.
- **Pragmatic Knowledge**: This concerns how sentences are used in different situations and how use affects the interpretation of the sentences.
- **Discourse Knowledge**: This is knowledge on how preceding sentences affect the sentence that immediately follows. It is especially important for interpreting pronouns and temporal aspects.
- **World Knowledge**: This is general knowledge on the world, whether the real one or a circumscribed one, including common-sense expectations and beliefs which can be assumed.

Phonology is only of limited interest to MT, so it will not be further discussed here. Morphological processing, on the other hand, is quite important for general systems, since it drastically diminishes the number of entries required in the dictionary of a system. In the case of a sublanguage system, such as the project documented in this report, this would be unfeasible due the limited vocabulary involved. Thus, morphology is not discussed further here. However, representation and processing of the other types of knowledge are discussed below.

5.2 Syntax and Grammars

In general, syntax is concerned with two slightly different sorts of analysis of sentences. The first is constituent or phrase analysis, which is the division of sentences into their constituent parts and the categorisation of these parts as nominal, verbal and so on. The second has to do with grammatical relations, that is, the assignment of grammatical realisations such as subject, object, head and so on to various parts of the sentence.

5.2.1 Grammars and Constituent Structure

Sentences are made up of words, traditionally categorised into parts of speech or categories, including nouns, determiners, verbs, auxiliary verbs, adjectives, adverbs and prepositions. These are normally abbreviated using symbols such as N, DET, V, AUX, ADJ, ADV and P, which are called lexical symbols. A grammar for a language, then, is a set of rules, which specifies how these parts of speech can be put together to make grammatical sentences or other well-formed phrases. The latter are also represented using other grammatical symbols. Typically, a sentence is abbreviated to S, noun phrase to NP, verb phrase to VP, adjective and adverb phrases are represented by ADJP and ADVP respectively, while PP stands for a prepositional phrase.

In linguistics, grammars are usually given in a semiformal manner and typically complemented with exceptional circumstances to the main rules (cf. [4]). This, of course, is not directly amenable to codification, so in computational linguistics it common to formalise grammars using rewrite rules. Such rules determine the possible structure of valid sentences in the language by asserting how words can be combined to derive complex phrases, and how these phrases can be combined in turn, until a valid sentence is obtained.

1. S → NP VP
2. NP → N
3. NP → ADJ NP
4. VP → V
5. VP → V ADJ

Figure 5.1 A Simple Grammar

Context-Free Grammars (CFG), which are grammars consisting entirely of rewrite rules with a single symbol on the left-hand side, are of particular interest. The reasons are that this formalism is powerful enough to describe most of the structure in natural languages, yet it is restricted enough so that efficient parsers can be built to analyse sentences. An example of such a grammar is given Figure 5.1.

1. N → periods
2. V → becoming
3. ADJ → sunny
4. ADJ → cloudy

Figure 5.2 A Simple Lexicon

Rules concerning lexical categories, which can be technically considered part of the grammar in a formal sense, are usually specified separately and referred to as a lexicon or dictionary. Lexicons may also give *subcategorisation* information, specifying the syntactic environment in which a word can occur, such as whether a

verb is intransitive, transitive or ditransitive, as well as some other semantic information discussed in section 5.3. The rules in Figure 5.2 specify a very simple lexicon.

With these, it is then possible to extract the syntactic structure of a sentence or phrase. This information is usually visualised by means of syntactic trees, represented as a labelled bracketing of a string of words or using graph notation. For example, the analysis of the phrase *sunny periods becoming cloudy* with the grammar and lexicon above would result in the structures shown in Figure 5.3.

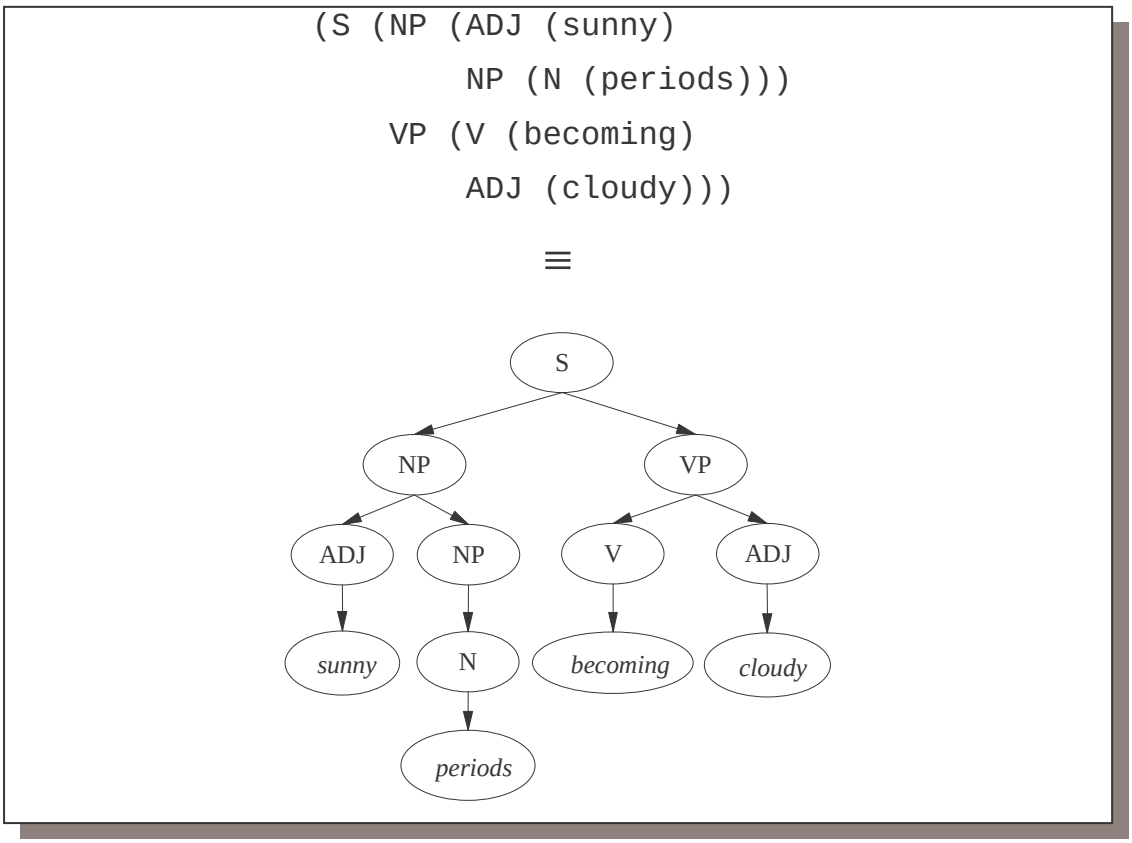


Figure 5.3 Representing Syntactic Trees

Parsing, the process that utilises the grammar and lexicon to extract such a representation is covered in section 5.6.1. It may be noted that there are alternative formalisms, such as transition networks[16], for representing grammars.

In constructing a grammar for a natural language, the interest is in the range of sentences the grammar analyses correctly, the range of non-sentences it identifies as problematic, and the simplicity of the grammar itself. Ideally, a grammar should be both complete in its coverage, and correct in disallowing invalid structures. While this is possible for formal languages such as those used in programming, it a matter of debate whether a natural language can be thus characterised with a CFG, even in theoretical terms. In addition, the complexity of the grammar increases exponentially as the number of structures recognised is increased.

5.2.2 Grammatical Relations

It is possible to describe syntactic structure at a level higher than by simply describing constituent relationships. In particular, it is possible to extract information about which phrases fulfil which grammatical relations or grammatical functions, such as subject, object and sentential complement. In English, the subject is normally the noun phrase that comes before the verb, while objects normally occur immediately after the verb. In other languages, these relations may, or may not, be realised differently with respect to the verb. Maltese, for instance, follows the same pattern, but in Japanese[17], the normal word order is subject-object-verb, and in Irish and Welsh, it is verb-subject-object. In many languages, such as Russian, the verb, subject and object can appear in essentially any order. This suggests that while constituent structures of languages differ greatly, the relationships may appear more similar when described in terms of grammatical relations.

[16] Details concerning these other representation forms can be found in [1].

[17] Information obtained from [2].

5.3 Semantics

Working out the meaning of sentences is an important part of the translation process for human translators, and the ability to work out the meaning of the source text would allow an MT system to produce much better translations. Semantics is concerned with the meaning of words and with how these combine to form sentence meanings. It is useful to distinguish between *lexical semantics* and *structural semantics*. The former has to do with the meanings of words, the latter to do with the meanings of phrases, including sentences.

There are many ways of thinking about and representing word and sentence meanings. For computational purpose, it is useful to assume that these can be represented in some logical or quasi-logical form, such as first-order predicate logic[18]. With this, objects can be represented as terms and relationships with predicates, while quantifiers can be used to define scope.

As for giving a sense to individual words, one mechanism that has proved useful in the field of MT involves the association of words with semantic features that correspond to their sense components. Features can be thought of attribute-value pairs, where, in fact, values can themselves be features as well. As regards semantics, features can be very useful for performing word-sense disambiguation, such as for example, when handling homonyms, that is, words with the same spelling but different meaning. For instance, the English word *period* can mean either an interval of time, or a full stop, besides other things. In order to differentiate between them, a lexicon might use features as shown in Figure 5.4.

[18] Such a formalism is discussed in detail in [1].

$$\begin{bmatrix} lex: & period \\ cat: & N \\ sem: & PERIOD1 \\ TIME & + \end{bmatrix}$$

$$\begin{bmatrix} lex: & period \\ cat: & N \\ sem: & PERIOD2 \\ PUNCTUATION: & + \end{bmatrix}$$

Figure 5.4 Features for Semantic Information

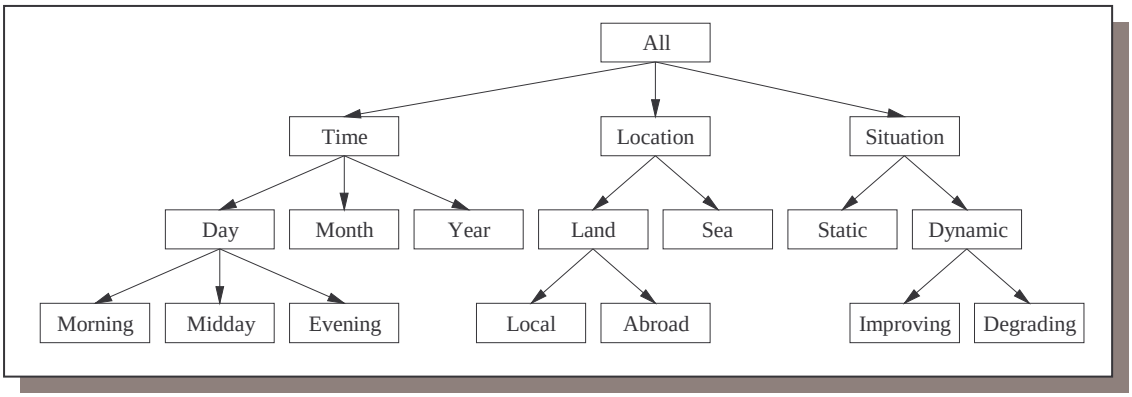


Figure 5.5 A Semantic Type Hierarchy

Thus, in extracting the meaning of the phrase *sunny periods* a system uses that information, as well as information on *sunny* to determine that the correct, or at least preferable, interpretation. In this case, the system might deem that the more likely meaning of *period* (ignoring other possibilities) is that of a time interval, since punctuation words are rarely modified by adjectives. When used in this way, features are usually referred to as *selectional restrictions*. In order to be consistent in their use, it is also typical to use a *semantic type hierarchy*. In the context of weather reports, this could be something like that shown in Figure 5.5.

One of the principal assumptions often made about semantic interpretation is that it is a compositional process. This means that the meaning of a constituent is derived solely from the meanings of its subconstituents. A useful formalism for this purpose is the lambda calculus, which can be used to represent semantic structures that can be manipulated in a compositional way.[19].

When building a system for a restricted domain, one can take advantage of the predetermined context of the application to improve the efficiency of parsing and semantic interpretation. One such technique is the use of a semantic grammar, which is a grammar that is cast in terms of the major semantic categories of the domain. For instance, when defining constituents for the grammar to parse phrases concerning weather reports, one could define a constituent TIME-N for a time-related noun, such as *period*, or LOCATION-N for a location related noun, such as *sea*. This has a number of advantages compared to using generic constituents. Firstly, semantic specification for words will be simpler, since some of it will be implicitly associated with the constituent category. Secondly, the grammar rules will provide more efficient parsing, since selectional restrictions are again implicitly performed by the rules themselves. However, such grammars tend to be large and unfeasible for broad domains, and they are usually not portable to new domains.

5.4 Pragmatics and World Knowledge

While semantics is concerned with context-independent meaning, pragmatics deals with the context-dependent meaning of sentences. The term context is used ambiguously to refer to the rest of the text in which a sentence occurs, sometimes referred to as the discourse. It also refers to circumstances external to the text itself, such as who the author of the text is, and the social setting in which it occurs, which also contribute to its interpretation.

[19] The use of the lambda calculus for semantic representation is further discussed in [1].

For instance, out of context, the translation of the phrase *mainly fine* is ambiguous in that it does not specify what is being considered fine. This causes problems in translating into a language that differentiates adjectives according to gender, and in giving a correct interpretation of the vague term *fine*. However, if it is known, or assumed, that the context is one of weather reports, then it would be deemed likely that this refers to the state of the weather, and translation can proceed accordingly.

However, the knowledge needed to extract the meaning from texts and translate them may not be discernible from the texts and their contexts only. A classic example involves determining to what the pronoun *they* refers to in *The council refused the women a permit because they advocated violence*. One would assume *they* refers to *the women* since advocating violence is a good reason for refusing a permit. The knowledge that is utilised here is non-linguistic knowledge, and the reasoning is more or less common sense. These are both quite difficult to represent and automate.

Some methods of representation for real-world knowledge have actually been developed. One involves specifying the knowledge as a set of facts and rules, in a fashion similar to Prolog programming. Inference rules can then be applied to achieve the required conclusions. Another form of representation is the use of a semantic network, that is, a network that represents relationships, such as '*is a*' or '*consists of*' relationships, between concepts, depicted as nodes. An example of this is given in Figure 5.6.

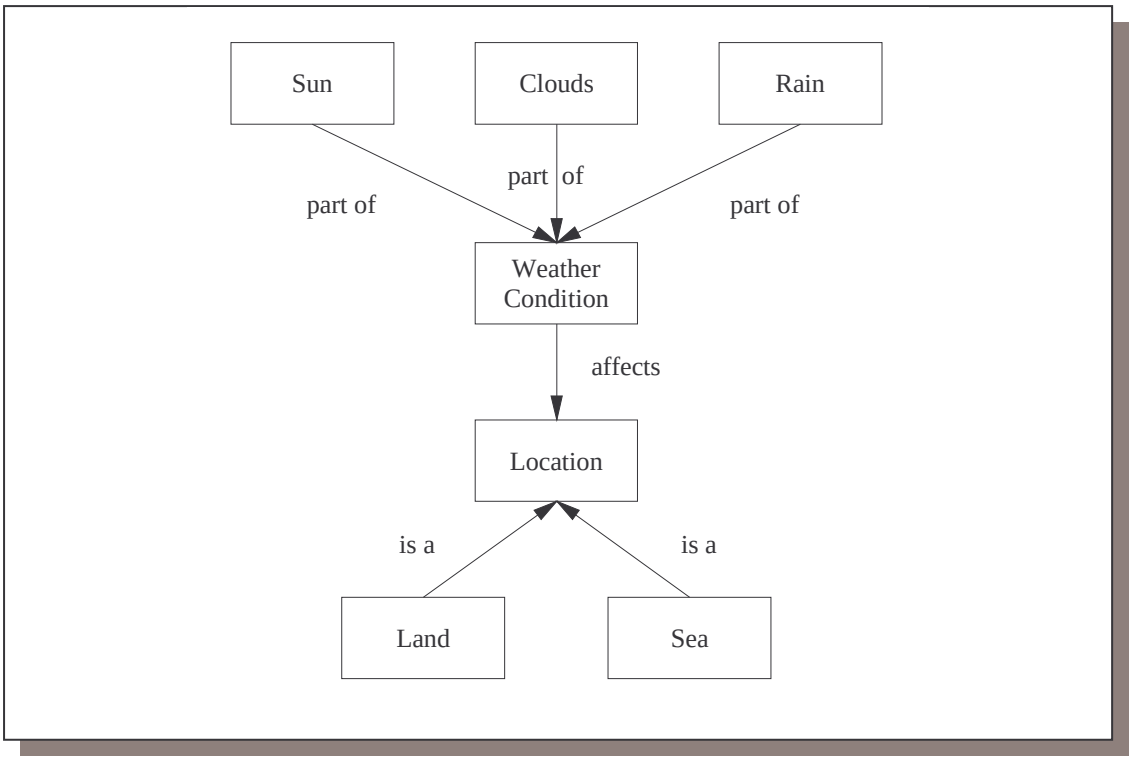


Figure 5.6 A Semantic Network

Another technique is to encode information in structures called *frames*, which are a cluster of facts and objects that describe some typical object or situation. The principal objects in a frame are assigned names, called *slots*. Thus, a frame representing a weather condition may have slots for the location that is affected, the time, the amount of clouds and rain, and so on, which can be instantiated with actual values. The frame then allows stereotypical reasoning on slots and other objects represented in it. Thus, for instance, if the amount of rain in the weather condition slot is high, it can be inferred that the location is wet.

5.5 Linguistic Information in MT Systems

In MT systems, linguistic information at the lexical level is specified in the system dictionaries. These can be diverse in terms of formats, coverage, level of detail and precise formalism for lexical description. Different theories of linguistic

representation can give rise to different views of the dictionary, and different implementation strategies can make even fundamentally similar views of the dictionary look very different in detail.

Moreover, the different kinds of MT engine obviously put quite different requirements on the contents of the dictionary. For example, dictionaries in an interlingual system need not contain any translation information. All that is necessary is to associate words with the appropriate interlingual concepts. By contrast, transformer systems will typically give information about SL items, and their translations, including perhaps information that is really about the TL, and which is necessary to trigger certain transformations.

It is usual to distinguish between two types of dictionary, *monolingual* and *bilingual*[20]. Roughly speaking, these give to the MT system the same kind of information that monolingual and bilingual paper dictionaries give to a human. Monolingual dictionaries contain syntactic and semantic information corresponding to words in a single language, and they are used to derive information for the various levels of representation involved in analysis and generation. By contrast, bilingual dictionaries normally relate the source and the target lexical items, and contain information only about the levels of representation that are involved in transfer.

Dictionaries are the largest components of an MT system in terms of the amount of information they hold. More than any other component, the size and quality of the dictionary limits the scope and coverage of a system, and the quality of translation that can be expected. Their management is thus an integral part of MT system maintenance. For such purpose, it is usual to include in them documentation on design decisions and terminology, use of dictionary terms and other

[20] This must not be confused with the term *bilingual* as applied to a system. As described in section 3.1, a bilingual MT system is one that handles only a single language pair, irrespective of the type of dictionaries used.

lexicographer's comments on each entry. In general, the quality and quantity of such documentation has no effect on the actual performance of the dictionary, but it is critical if a dictionary is to be modified or extended.

The second main repository for linguistic information in an MT system lies in the grammars used. The same kind of distinction used for dictionaries applies here, in that grammars can be both monolingual and bilingual, or *comparative*. Monolingual grammars are used to specify the valid constructions for a particular language, using subcategorisation information and selectional restrictions specified at the lexical level to filter unwanted interpretations. The comparative grammars, on the other hand, specify how linguistic structures for the SL can be mapped to those for the TL, based on information from the bilingual dictionaries.

5.6 Processing

The issues involved in transferring linguistic structure from one language to another were expounded upon in section 4, so will not be further discussed. Issues on analysis and generation are described here. Analysis or parsing, is the process of taking an input string of expressions and producing some form of representation. Generation, or synthesis, is the reverse process taking a representation and producing the corresponding sentence.

The system grammars are responsible for directing both processes. A grammar that supports both analysis and generation is called a bidirectional grammar. Such kind of grammar is obviously desirable, since it would not require any change when either using it for parsing or for generation. In practice, however, grammars are often tailored for the parsing task or the generation task, because different issues are important for each task.

Another desirable property for MT purposes is that the grammar used for the SL and that used for the TL be isomorphic. That is, rules or subsets of rules one grammar have a one-to-one correspondence with other rules or subsets of rules of the

other. This is important, because it means the generation grammar will be able to synthesise any representation generated by the analysis grammar. Such an approach is at the heart of the ROSETTA[21] MT system, based on Montague Grammars[22], a type of isomorphic grammars. In practice, this is again difficult to achieve, and entails that the grammars for the SL and TL have to be developed in parallel.

5.6.1 Parsing

Parsing is the process of taking a formal grammar and a sentence, and applying the grammar to the sentence in order to check that it is indeed grammatical. Given that it is grammatical, its constituent structure is derived in some format as described in section 5.2. There are many ways to apply the rules to the input to produce such structures. It is usual to distinguish between two main approaches. The first, top-down, approach uses the grammar rules to try to match the structure of the input, expanding topmost level rules first. It is also possible to differentiate this approach between depth first and breadth first, depending on which constituent in the partial syntactic tree being built is chosen next for expansion. With the bottom-up approach, the input is used to guide the parsing, and rules are matched to this structure until the topmost grammar symbol is reached. Both approaches have their positive and negative aspects, and in practice, efficient parsers use some characteristics of both.

This description applies only to building the surface, constituent structure tree. As regards other levels of representation, such as representations of grammatical relations and semantic representations, there are two basic approaches. If information about other levels of representation is represented as annotations on the constituent structure rules, then it should be possible to construct these other representations at

[21] The ROSETTA system is described in “Isomorphic Grammars and their use in the ROSETTA Translation System”, [13], by Jan Landsbergen.

[22] Montague Grammars, and M-Grammars, a derivative of the former, are described in [14].

the same time as the constituent structure representation. This is slightly harder if the relationships between levels are stated in a separate collection of rules. In this case, the constituent structure representation may be built first, and the rules are applied to that representation.

5.6.2 Generation

The difficulty in the generation process depends on the kind of structure used. Generating a string from a constituent structure representation, such as a syntactic tree, is almost trivial. At worst, the words will have to be modified to get the correct form, such as by performing morphological processing, before extracting them from the structure. Starting from a representation of grammatical relations or a semantic representation poses much greater difficulties.

When using deep representation structures, generation is harder than parsing because the analysis tree, being an abstraction of the original sentence, contains less information. This is particularly the case in MT, where the analysis tree is the result of transfer operations that in no sense guarantee to produce a complete structure. Handling the situation described in section 2.2 provides an example, in that the structure representing the phrase *this thesis* might not indicate that the article needs to be inserted for a proper translation into Maltese.

If the relations between syntactic, grammatical relation structures, and semantic structures are described by means of explicit rules, then one approach is to use those rules in the same way as described for parsing but in reverse. That is, with the part of the rule written after the arrow interpreted as the left-hand side. Things are not quite so straightforward when information about grammatical relations or semantics is packed into the constituent structure rules, since this might not be directly reversible.

One possibility is to have a completely separate set of procedures for producing sentences from semantic or grammatical relation structures, without going

through the constituent structure stage. For example, one would need a rule that puts the head, subject and object of a sentence into the normal word order for English, depending on whether the sentence was active or passive, interrogative or declarative. The disadvantage, however, is that the developer will end up describing again most, if not all, of the knowledge that is contained in the grammar which is used for parsing.

6 Evaluation of MT Systems

6.1 Types of Evaluation

MT being a research area of practical interest, it is obviously desirable to quantify the quality of the systems created. The most intuitive measure one can come up with is the relatedness between the output of the system and the needs of the user. For example, a consumer magazine might compare different systems based on such criteria as speed, quality and usability. An example of this can be found in [25].

As in other areas of NLP, three types of evaluation are usually recognised, each appropriate to three different goals. The distinction between criterion, measure and method used in each is not hard and fast and the three are interdependent. The evaluation methods are the following[23]:

1. **Adequacy Evaluation**: This determines the fitness of a system for a particular purpose, that is, whether it does what is required, how well, at what cost and other similar considerations. It may or may not use comparative measurements, and usually requires considerable effort to identify the user's actual needs. In the case of MT, adequacy evaluation is used to determine the fitness of systems within a specified operational context. It is typically performed by potential users or purchasers of systems, whether individuals, companies, or agencies.

2. **Diagnostic Evaluation**: This is the production of a system performance profile with respect to some taxonomy on the space of possible inputs. It is

[23] This classification is adapted from “Evaluation”, [6], by various authors, with Joseph Mariani as chapter editor.

typically used by system developers, but sometimes offered to end-users as well. It usually requires the construction of a large and hopefully representative test suite. Diagnostic evaluation is used to identify limitations, errors and deficiencies in MT systems, which may be corrected or improved by the research team or by the developers, to whom it is mostly relevant.

3. **Performance Evaluation**: This is the measurement of system performance in one or more specific areas. It usually takes the form of a comparison between similar systems, whether alternative implementations of the same technology, or successive generations of the same implementation. This implies that this kind of evaluation is largely application specific, and different methods will be required for different systems. Typically, it is created for system developers or R&D programme managers. Performance evaluation is used to assess the stages of development of an MT system or to compare different versions of the same system. It is usually undertaken by researchers or developers or by potential users.

All the usual evaluation issues that concern the engineering of software applications, such as black box and glass box evaluation, user friendliness and acceptability, and marketability, can be associated with the production of an MT system. However, at least from a theoretical point of view, the major evaluation issue is that related to the quality of output produced by the system. This is not an easy task to perform because it is hard to quantify the notion of good quality objectively in this context.

The main problem is that the information methods required in order to make judicious comparisons are not well developed. While there is general agreement about the basic features of MT evaluation, there are no universally accepted and reliable methods and measures, and evaluation methodology has been the subject of much discussion in recent years.

6.2 Evaluating Output Quality[24]

Productivity and cost-effectiveness obtained by using MT systems are variable, and depend on the context within which they are used. Typically, a system may perform very well on a certain forms of input, but badly or erratically on other forms. Thus, the first requirement in evaluating output quality is to establish clearly a context in which this will be done.

Whichever context is chosen, however, the problem of defining an evaluation method remains. Assessing translation quality is not just an MT problem, but a problem that even human translators have to face. There are typically many valid translations for a particular text, some of them being faithful to the original in content, while some may preserve other aspects such as style and emotion. The main methodologies that have been recognised assess the output in terms of the intelligibility, accuracy with respect to the original or in terms of the number of errors discerned. One main problem with all three is that the measurements obtained are difficult to explain. In addition, all three will require the development of a representative test suite.

6.2.1 Intelligibility

The most straightforward method for evaluating a system is by grading its output on some scale according to intelligibility. Top marks are given to those sentences that are completely comprehensible in the target language. Bottom marks are given to output that is so incomprehensible that the translator or evaluator has no clue as to what the original source might have been. Output will probably be graded somewhere in between, depending on the severity and quantity of errors. This measure directly reflects the quality assessment of a user. The more one understands

[24] This section draws heavily from the same considerations given in [2].

the higher the perceived quality. During evaluation with this method, no reference should be available to the original source text.

6.2.2 Accuracy

Accuracy testing is also based on a grading system like that for intelligibility. However, this time, the ranking is based on how close the output is to the original text. In this case, of course, reference to the translation input is necessary. It is important to realise that intelligibility and accuracy are directly related. An intelligible output from a sensible system will also tend to be accurate. Conversely, an unintelligible output will result in low accuracy grading, since it is difficult to impose any meaning to it. In fact, accuracy testing usually follows intelligibility testing in practice. However, due to the properties described, accuracy is not as interesting to MT evaluation, since the intelligibility measure will usually be indicative of the accuracy anyhow.

6.2.3 Error Analysis

Another scoring mechanism for MT is based on the number and types of errors that can be perceived in the output. With this method, a set of possible errors, such as invalid number agreement or preposition attachment, is identified. Each is given a weighting factor based on the severity of the error. The total error in the output is then calculated as the weighted-sum of all the errors that can be identified.

Unfortunately, there are several pitfalls with applying this approach. Firstly, analysing the output carefully for the different types of errors is a time-consuming and error-prone task. Evaluators will have to be well trained to recognise all the different types of errors in all their possible forms. Secondly, coming up with a sensible weighting scheme may not be that easy. Possible approaches could be to base it on the cost required for post-editing, or on the effect of intelligibility, but there are no straightforward ways to quantify these, either.

Thirdly, and perhaps most significantly, the output of a system may be so unintelligible that it is largely impossible to determine the actual errors in midst of the ungrammaticality of the output. Errors in grammaticality affect each other, and it would be difficult, in a highly corrupted phrase, to determine when one error starts and another begins. Thus, the assignment of weights can become very tricky.

Finally, there is an issue as to the amount of linguistic knowledge required by users in order to evaluate properly errors in the output. Typically, knowledge of syntactic categories and phrase structures is a prerequisite, and evaluators without this knowledge may have to be trained for the task. In contrast, the other two methods may be performed on informal terms by users without such specialised knowledge.

6.2.4 Using the Grading Derived

Deriving an evaluation for a system in terms of the grading methods described above is straightforward. There is a problem, however, as to how the derived grading is used in describing the output. One could plot a performance profile plotting the grading derived, as shown in Figure 6.1. However, such a plot does not given any indication of how the system will satisfactorily translate new input, or whether it will be cost effective. Even plotting profile curves for different systems, as shown in the graph, does not aid in discriminating between their operational qualities.

6.2.5 Selecting a Text Suite

When performing evaluation, the text used for translation has to be carefully selected. The choice may depend on the context in which the system is to be used. In a closed environment, this may mean using representative samples of the actual future input to the system. However, in general MT, one does not have such a simple choice. One possibility is to use ever-larger corpuses of text. The problem with this is that such corpuses may not have the actual types of input that are problematic to the system.

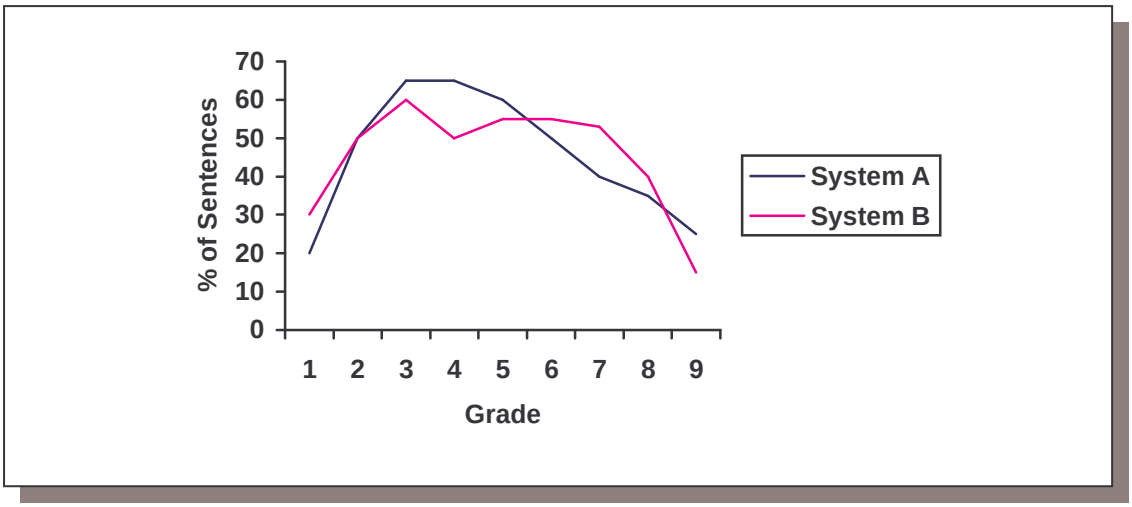


Figure 6.1 An Output Quality Profile

The trend is instead to derive a specially designed text suite that contains all the grammatical and ungrammatical forms of input, and evaluate using these. This is a complex task since the text suite needs to be complete with respect to the phenomena in the present and future input to the MT system. There is also the danger of being biased with respect to the system when creating the sample phrases. Research is going on into what form of phenomena should be included in these text suites. It is obvious that to be complete, such a text suite would have to be considerably large, and its development would be quite costly.

7 Creating A Bilingual Text Archive

7.1 Obtaining the Text Data

At the outset, the project required the development of a substantially large bilingual text archive from which to develop the required grammars and dictionaries. The source chosen for the archive also had to display certain characteristics required for building a restricted domain translation system. In particular, the data used had to have:

- Translations in both languages to allow comparison between human and MT output
- A limited syntactic and semantic domain, permitting optimisation of the translations by permitting assumptions on the input
- Enough samples for both development and evaluation purposes

After perusing available data sources for suitability, it was decided to use a form of weather reports issued daily in local newspapers. The Meteorological Office of the Malta International Airport, Luqa, originally generates these reports in the form shown in Figure 7.1[25]. Local newspapers receive a copy of such a form and then print a subset of the entries depending on whether the journal is written in English or in Maltese.

[25] The form reproduced here was kindly made available by officials at the Meteorological Office, Luqa, Malta.

Date: 04 Nov 98

WEATHER FORECAST FOR MALTA AND 10 MILES RADIUS

IT-TEMP SA 10 MILI MADWAR MALTA

GENERAL STATEMENT:	A TROUGH OF LOW PRESSURE EXTENDS FROM THE BALTIC TO THE BALEARICS AS A HIGH PRESSURE AREA PERSISTS OVER THE EAST MEDITERRANEAN		
FORECAST DAWN TO DUSK		TBASSIR TAT-TEMP SA NŻUL IX-XEMX	
WEATHER:	Mainly fine becoming rather cloudy at times	IT-TEMP:	Ġeneralment sabiħ li jsir xi ftit imsaħħab kultant
VISIBILITY:	Good	VIŻIBILTA:	Tajba
WIND:	Moderate to rather strong Southerly (over sea areas force 4 to 5)	IR-RIĦ:	Moderat għal ftit qawwi min-Nofsinhar (fuq il-baħar forza 4 għal 5)
SEA:	Moderate to rather rough	BAĦAR:	Moderat għal ftit qawwi
SWELL:	Low Southerly becoming low to moderate	IMBATT:	Baxx min-Nofsinhar li jsir baxx għal moderat
FORECAST MAXIMUM TEMP. L-OGĦLA TEMPERATURA: 24 C			

FORECASTER_____PJD

LUQA OBSERVATIONS AT NOON ON 3-Nov-98		
Mean sea level pressure 1017.6 hPa	Wind S 11 kt	Cloud 1/8 CU
Visibility 20km	Relative Humidity 71%	Hours of bright sunshine 9.5
Max. Shade Temp (day) 24 C	Min air temp. (night) 16 C	Average Sea Temp 22C
RAINFALL – Average during the past 24 hours up to noon yesterday		
Average MALTA nil mm	Average GOZO nil mm	TOTAL since 1st Sept. 77.9 mm
SUNRISE at 0627	SUNSET at: 1704	PHASE OF MOON: FULL

Figure 7.1 A Weather Report Form

Unfortunately, while one copy of such a report was kindly made available from the officials responsible, it was not possible to obtain these in sufficient volume as required for a suitable text archive. This was because these forms are regularly disposed of at the end of the month. Instead, it was necessary to refer to old issues of local newspapers from the central Maltese public library for a contiguous list of such reports.

While it has been remarked that newspapers only publish a subset of the fields from the original report, the fields of interest for the task are those that have both a translation as well as relevant syntactic content. It can also be easily seen that these are the *Weather*, and to some extent, the *Visibility*, *Wind*, *Sea* and *Swell* entries. These fields, and some others, are printed in each local newspaper, but only in the journal's own language. Hence, it was necessary to lookup a consistent set of both Maltese and English newspapers, and manually input this data in machine-readable form. Consequently, six months worth of sequential reports were copied from a local library, and tabulated in Microsoft Excel format. Samples taken from the text archive are shown in Appendix A.

7.2 A Note on the Maltese Alphabet

The Maltese alphabet uses a subset of the Roman alphabet, with the addition of some special characters. It has 24 consonants and 5 vowels, which are:

A a, B b, Ċ ċ, D d, E e, F f, Ġ ġ, G g, GĦ Għ, Ħ ħ, H h, I i, J j, K k, L l, M m, N n, O o, P p, Q q, R r, S s, T t, U u, V v, W w, X x, Ż ż, Z z

In addition, there is also a special conjunction of vowels, ç. As can be seen, this latter, and some other of the above, are not characters represented in the ASCII character set and special fonts have to be used to render them properly. This leads to difficulties in processing both the input and the output of Maltese texts.

In fact, this is a problem with all MT systems required to handle languages with special characters or diacritics. Although Unicode is becoming increasingly popular, there is no genuinely accepted standard for character representation beyond basic ASCII. Thus, it was decided to use the latter for processing input and output texts, but to use special sequences of ASCII characters to represent the special characters described above.

For this purpose, a convention used by Maltilex – an ongoing project ([20]) for the development of a computational lexicon for the Maltese language – was adopted. With this, the special characters in the Maltese text are replaced by a special character sequence in their ASCII representation, as shown in Table 7.1. This approach was utilised when storing Maltese text in machine-readable format.

Maltese	ASCII
Ċ	_C
ċ	_c
Ġ	_G
ġ	_g
Għ	_Y
għ	_y
Ħ	_H
ħ	_h
Ż	_Z
ż	_z
Ie	_I
ie	_i

Table 7.1 The Maltilex Character Convention

8 Developing a Natural Language Chart Parser

8.1 Developing a Parser for Analysis

Most MT systems use a parser in order to extract the syntactic and, possibly, semantic structures corresponding to the input text. Since this was deemed an appropriate initial transformation of the input, the second phase of the project involved the development of a general parser, for which an SL lexicon and grammar would then be created. This would form the basis of the analysis processing stage.

8.2 Chart Parsing

As explained in section 5.6.1, there are two main parsing approaches, top-down and bottom-up, both of which try to match the input sentence with a possible derivation from the topmost or start symbol, usually denoted by S.

In practice, naïve implementations of both approaches would result in inefficient parsers. The reasons are that these do not store information on parses of sub-segments of the text, resulting in unnecessary duplication of effort when the same sub-parses are found repeatedly. In addition, they may try search paths that could not possibly lead to valid parses.

To avoid these problems, a data structure called a chart is often used for parsing. This allows a parser to store the partial results of the matching it has done so far so that the work is not duplicated. There exists a whole family of dynamic programming algorithms that use this structure to provide efficient parsing (cf. [1], [8], [11], [21]). These are usually called *Chart Parsers* or *Tabular Parsers*.

In the abstract, all chart parsers work in essentially the same way. The input is first split into a sequence of tokens according to the lexicon, and conceptually numbered as follows:

$$0\ w_1\ 1\ w_2\ 2\ w_3\ 3 \ldots n-1\ w_n\ n$$

The chart for an input sentence with n tokens is a directed graph containing $n+1$ vertices and a number of edges that connect these vertices. An edge from vertex i to vertex j is annotated with information of the parse available within that sub-segment of the text. A complete chart for the phrase *sunny periods*, obtained with the grammar and lexicon given in section 5.2.1, is shown in Figure 8.1.

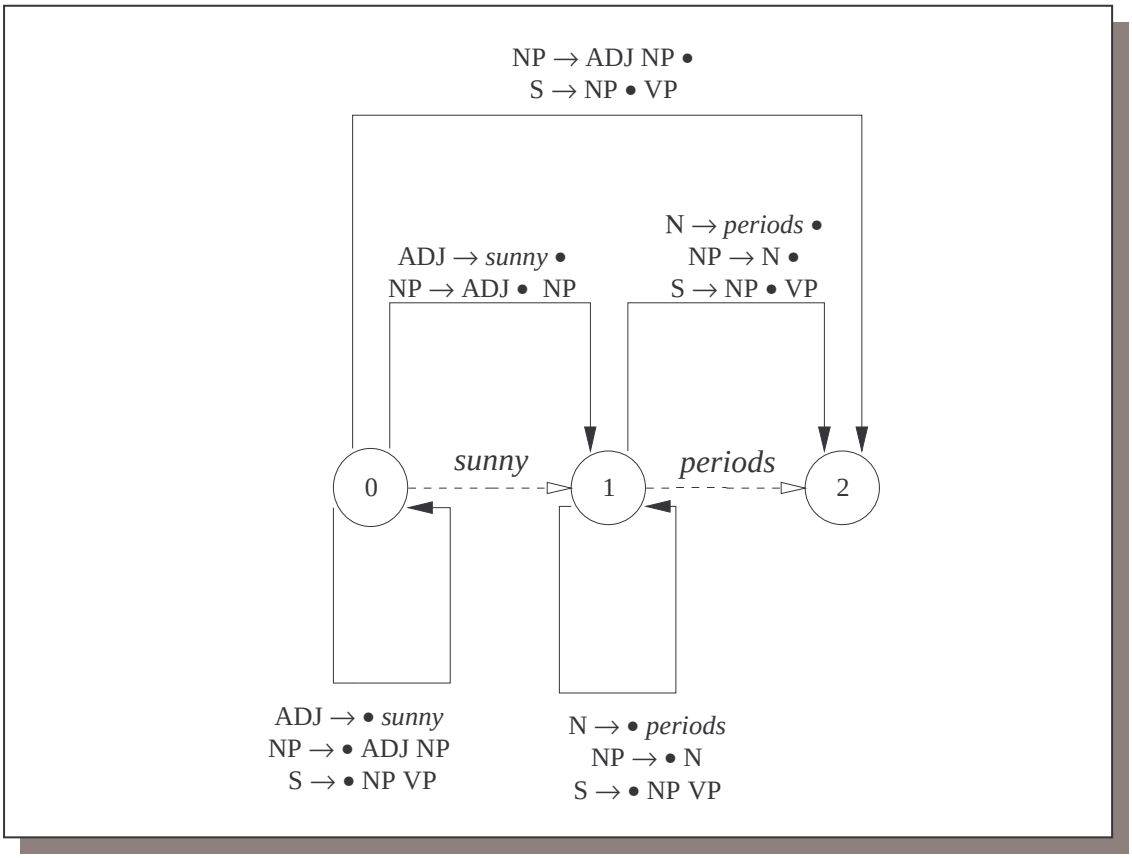


Figure 8.1 A Complete Chart

The information contained on each edge consists of a dotted rule such as, for example, $S \rightarrow NP \bullet VP$. The dot separates the structure that has been found so far from what remains to be found. Thus, for example, a word at position i having the category X in the lexicon will generate an edge from vertex $i-1$ to vertex i, having the dotted rule $X \rightarrow w_i \bullet$.

According to the associated dotted rule, an edge can be *active* or *complete*. A dotted rule of the form $X \rightarrow X_1 X_2 \ldots X_i \bullet X_{i+1} \ldots X_n$ implies that the edge is active, meaning that a partial parse has been found up to the dot, but not enough has been found to satisfy the rule. On the other hand, a complete edge from vertex i to vertex j has a dotted rule of the form $X \rightarrow X_1 X_2 \ldots X_n \bullet$, meaning that a parse for X exists in the segment spanned.

The basic idea for covering larger and larger segments of the text is that having an edge from vertex i to vertex j with the dotted rule $X \rightarrow X_1 X_2 \ldots X_i \bullet C \ldots X_n$ and an edge from vertex j to vertex k with the dotted rule $C \rightarrow X'_1 X'_2 \ldots X'_n \bullet$, then an edge from i to k can be added to the chart with the dotted rule $X \rightarrow X_1 X_2 \ldots X_i C \bullet \ldots X_n$. The intuition behind this should be obvious, meaning that if there is a parse for C from j to k and the parser is looking for C at position j, then C can be considered found, and the following constituent, if any, is expected at position k.

Creation of the chart is started with dotted rules derived from the lexicon as explained above, and the grammar rules are used for specifying which dotted rules can be created next. Hence, the main task of a chart parser is to combine all possible edges in the manner above. Full parses of the whole text are indicated by edges from vertex 0 to vertex n whose dotted rules have the start symbol of the grammar on the left-hand side.

In practice, certain techniques can drastically improve the efficiency of a chart parser. One such is the use of *prediction*, or *top-down filtering*, which avoids the insertion of edges that will never be used in the parse of the whole text, such as the $S \rightarrow \bullet\ NP\ VP$ in the edge attached to vertex 2 in the chart above. With this, when the parse has proceeded up to word i, edges are inserted from vertex i to i, but only if these edges can possibly be combined with edges ending at that same vertex. As a special case, when starting the parse a top-level grammar rule $S \rightarrow NP\ VP$ would generate an edge from vertex 0 to itself having the dotted rule $S \rightarrow \bullet\ NP\ VP$. Members of the closure of the rules that could be used to derive a parse using that rule would also be used to generate dotted rules. Thus, a rule $NP \rightarrow N$ would also generate a cyclic edge at vertex 0 with the dotted rule $NP \rightarrow \bullet\ N$.

The second efficiency technique is the use of *lookahead*, which also tries to reduce the amount of useless edges generated. However, in contrast to prediction, which considers the left context of the current parse, lookahead takes the context on the right in order to achieve its goal. The idea is to look ahead the categories of the next n words from the current position in the parse, and only build edges that could be possibly used to combine with edges created for the following input. In practice, n is usually taken to be 1.

As regards the actual algorithm used for performing the processes above, more than one was found in the literature. A first implementation was based upon definitions and algorithms given in [11]. However, it turned out that this implementation was not efficient enough, taking a very long time to run or running out of memory on even modest input, especially when features were added (see section 9). Thus, the first implementation was modified to utilise the structures and algorithms specified in [21], extended to use lookahead and certain indexing techniques that also increase efficiency. Both algorithms are described below.

8.3 A CKY Chart Parser

The first algorithm used for building a chart parser was based on that given in [11]. This was developed independently by J. Cocke, T. Kasami and D. H. Younger in the early mid 1960s, and is thus often referred to as a CKY parser, but was later improved upon by J. Earley. In the reference, the algorithm is presented incrementally, starting with a simple version that does not even use dotted rules and incrementally adding functions and facilities. There is no space to do that here, so instead, a final version, similar to the final one in the reference with prediction and lookahead but lacking grammar rules with regular expressions and syntactic tree extraction, is presented here.

With the CKY algorithm, the chart is treated as a triangular matrix, with the entry (i, j) containing the set of dotted rules of all the edges from vertex i to vertex j. The matrix is built incrementally using a suite of functions returning sets of categories or sets of dotted rules

The first function is called $Lookup$, and it uses the lexicon L to create a set of complete dotted rules corresponding to a word and its categories. Its definition is given by:

$$Lookup(word) = \{A \rightarrow word \bullet \mid A \rightarrow word \in L\}$$

The second function, $First$, is directly related to prediction. Effectively, $First(K)$ returns the set of all categories that can start a constituent of category K, and is defined as follows:

$$First(K) = \{A \mid (A = K) \vee ((B \in First(K)) \wedge (B \rightarrow A\,X \in G))\}$$

With $First$ it is possible to define $Pending$, another prediction related function. When applied to j, $Pending$ returns the set of useful starting categories for position j in the parse:

$$Pending(j) = \left\{ A \,\middle|\, \begin{array}{l} ([B \to Y \bullet A Z] \in chart(i, j)\ \ 0 \le i < j) \vee \\ ((A \in First(B)) \wedge (B \in Pending(j))) \end{array} \right\}$$

Switching to a lookahead related function, $Next(i)$ delivers a set of all the categories of which word $i+1$ can be a prefix. Obviously, for an input of length n, $Next(n)$ returns the empty set.

$$\begin{array}{l} Next(i) = (0 \le i < n) \to \{A \,|\, (A \to word_{i+1} \in L) \vee ((A \to B\,X \in G) \wedge (B \in Next(i)))\} \\ \qquad otherwise \to \{\ \} \end{array}$$

Prediction and lookahead are then assimilated into one function, actually called *Predict*. It takes two arguments, both of which are sets of categories. The first argument is the set of categories predicted from the left. The second is the set of categories that are compatible with the next word on the right. It then returns only the useful dotted rules considering these two arguments.

$$Predict(S, K) = \{[A \to \bullet\, B\,X] \,|\, (A \in S) \wedge (A \to B\,X \in G) \wedge (B \in K)\}$$

The remaining two functions deal with the actual combination of edges in the chart. One of them is called *Star*, and is used to combine edges. It takes three arguments. The first is a set of dotted rules of active edges ending at a vertex i. The second argument is also a set of dotted rules, containing those associated with complete edges starting at vertex i. The third argument is a set of lookahead categories, used to filter unnecessary edges. As a result, *Star* returns the set of dotted rules resulting from the combination of edges:

$$Star(R, S, K) = \left\{ [A \to X\ B \bullet Y] \,\middle|\, \begin{array}{l} ([A \to X \bullet B Y] \in R) \wedge \\ \qquad ([B \to Z \bullet] \in S) \wedge \\ \qquad ((Y \text{ is empty}) \vee ((Y \equiv C V) \wedge (C \in K))) \end{array} \right\}$$

Finally, there is the function *Closure*, which takes the same arguments as *Star*, and returns the closure of the application of the latter over combinable edges:

$$Closure(R,S,K) = \{X \mid (X \in S) \vee (X \in Star(R, Closure(R,S,K), K))\}$$

The CKY parsing algorithm that completes the chart utilising the functions above is then given in Figure 8.2.

The algorithm starts by putting cyclic edges on vertex 0 whose dotted rules are those that can be created by the start symbol of the grammar, filtered by lookahead on the first word. Then the main loop iterating over the vertices from 1 to n is started. This loop begins by applying *Lookup* and *Closure* on the current word, using prediction and lookahead filtering. Then, another inner loop performs the *Star* operation over all possible sub-segments from 0 to k. Before finishing, the loop performs prediction and lookahead for the current vertex, based on the categories that are pending there.

```
chart(0, 0) := Predict(First(S), Next(0))
for k := 1 to n do {
    chart(k-1, k) := Closure(chart(k-1, k-1),
                             Lookup(word_k),
                             Next(k))
    for i := k-2 downto 0 do {
        chart(i, k) := {}
        for j := k-1 downto i+1 do {
            chart(i, k) := chart(i, k) U Star(chart(i, j),
                                              chart(j, k),
                                              Next(k))
        }
        chart(i, k) := Closure(chart(i, i), chart(i, k), Next(k))
    }
    chart(k, k) := Predict (Pending(k), Next(k))
}
```

Figure 8.2 The CKY Chart Parser Algorithm

Upon termination of the main loop, any top-level parses of the whole text will be found in $chart(0, n)$.

8.4 An Earley Style Chart Parser

The CKY chart parser implementation proved very inefficient in practice. Even on modest input, having some 100 different parses, the parser either took hours to finish the parse or ran out of memory. Instead of trying to implement low-level efficiency techniques, it was thought more feasible to seek a more efficient algorithm. Several were found, but the one adopted was the Earley style chart parser given in [21]. This was mostly implemented directly from the pseudo-code given in the reference, but lookahead and certain indexing techniques were added, resulting in an order of magnitude in increased efficiency.

This algorithm is more directly related the notion of the chart as a graph, rather than as a triangular matrix. In fact, edges are represented using the notion:

$$[start\ vertex, end\ vertex, dotted\ rule]$$

Edges in the chart are accessed based on their end vertex. That is, $chart(j)$ represents the set of all edges of the form $[i, j, X \rightarrow X_1 \ldots \bullet \ldots X_n]$ in the chart. Closure of the chart under all combinable edges is performed using the following main functions and procedures.

The *lookahead* function behaves exactly like the one in the previous algorithm. Applied to the vertex j, it returns the set of useful categories that can start at that point in the text. It uses another function, $categories-of$, which returns a set of categories that a word has in the lexicon. In the implementation of the lexicon, indexing on the words was performed, in order to execute this function more efficiently, rather than traversing the whole lexicon, which can be quite large, each time.

```
function lookahead(int j) returns CategorySet {
    if (0 <= j < n) {
        CategorySet la = categories-of(string[j+1], lexicon)
        for each B in la {
            for each (A → B β) in grammar {
                la = la U {A}
            }
        }
        return la
    } else {
        return {}
    }
}
```

Figure 8.3 The *lookahead* Function

The *predictor* performs top-down filtering by taking an incomplete edge that is looking for an X, and adding a new incomplete edge that, if completed, would build an X in the right place. It uses a function $rewrites-for$ that returns the rules in the grammar that rewrite a specific category. In the implementation, another index was used on the categories on the left-hand side of grammar rules in order to improve the execution speed of this function.

```
procedure predictor([i, j, A → α . B β]) {
    for each (B → γ) in rewrites-for(B, grammar) {
        add-edge([j, j, B → . γ])
    }
}
```

Figure 8.4 The *predictor* Procedure

The *completer* procedure performs the combination of edges. It takes an incomplete edge ending at vertex j and looking for an X, and combines it with a complete edge that begins at j and has X on the left-hand side of the rule. Once more, indexing was used on incomplete edges ending at vertex j by the category X they are looking for.

```
procedure completer([j, k, B  → γ .]) {
    for each [i, j, A → α . B β] in chart(j) {
        add-edge([i, k, A → α B . β])
    }
}
```

Figure 8.5 The *completer* Procedure

The *scanner* is similar to the *completer* except that it uses the input words, rather than existing complete edges, to combine with incomplete ones.

```
procedure scanner(j, word) {
    for each [i, j, A → α . B β in chart(j) {
        for each (B → word) in lexicon {
            add-edge([i, j+1, A → α B . β])
        }
    }
}
```

Figure 8.6 The *scanner* Procedure

The *add-edge* procedure adds an edge to the chart if it is not already there, and passes it the *completer* or *predictor* depending on whether it is complete or active respectively. However, in the latter case, the edge is only added if the category that the edge is looking for is in the lookahead list for that vertex. In the implementation,

complete and active edges for each end vertex were stored separately to improve access time in functions that require to only access one type or the other.

```
procedure add-edge(edge) {
    j = end-vertex(edge)
    if (!chart(j).contains(edge)) {
        if is-complete(edge) {
             chart(j).add(edge)
             completer(edge)
         } else if lookahead(j).contains(required-cat(edge)) {
             chart(j).add(edge)
             predictor(edge)
         }
    }
}
```

Figure 8.7 The *add-edge* Procedure

```
function chart-parse(string, lexicon, grammar) returns chart {
    for each (S → γ) in rewrites-for(S, grammar) {
        add-edge([0, 0, S → . γ])
    }

    for v = 1 to length(string) {
        scanner(v, string[v])
    }

    return chart
}
```

Figure 8.8 The *chart-parse* Function

Finally, *chart-parse*, the main algorithm shown in Figure 8.8, performs the closure of all edges from vertex 0 to itself, before starting the parsing proper by calling the *scanner* function for all words.

8.5 Tokenisation of the Input

As implicit in the descriptions above, the very first step before parsing involves the application of a tokeniser to the input text. This is a device that segments an input stream into an ordered sequence of tokens, each token corresponding to an inflected word form, a number, a punctuation mark, or other kind of unit to be passed on to subsequent processing. When applied to natural languages, the tokenisation process may not be trivial. For instance, Japanese does not have spaces between word, and even in English, things like hyphens may be difficult to split up in the correct manner. Besides, if the text contains formatting, this must be taken into account by this process as well.

For the project, a tokeniser was required to operate on an ASCII character stream containing the source text, extracting the individual words upon which the parser is to work. Instead of simply splitting words at whitespace boundaries, it was desired to allow the tokeniser to handle multiword tokens. Such tokens can be of several types:

- adverbial expressions, such as “all of a sudden”
- prepositions, such as “in the evening”
- dates, such as “15 June”
- time expressions, such as “12:00”
- proper names, such as “Maltese Islands”

Such a tokeniser can be used to handle those types of multiword expressions or even idioms, although it is not the ideal tool for such a purpose, as explained in section 2.3. A problem with a multiword tokens is that they introduce multiple ways of splitting the input if there are no constraints on the selection of the multiword

tokens. To illustrate this, consider a tokeniser that recognises the multiword tokens '*a b*', '*b a*' and '*a b a*'[26]. The input '*a b a*' can be split up in four different ways, namely '*a - b - a*', '*a b - a*', '*a - b a*' and '*a b a*', depending on where the multiword token is made to start.

One option was to allow the tokeniser to produce all alternative segmentations and postpone the decision of choosing a correct interpretation to a later processing stage. For simplicity, however, it was decided to make the tokeniser deterministic, and not allow the output to contain alternative segmentations for any part of the input. To this purpose, a left to right, longest match first, tokenisation strategy was employed[27]. This guarantees a unique factorisation on every input string. So, for the example above, the input would be tokenised uniquely as '*a b a*'.

A problem with this technique is that multiword expressions are consistently treated as a single token, even if the component words should be separated in a given context. For example, if '*in general*' is included in the multiword lexicon, then it will be tokenised as a unit in both of the examples '*very hot in general*' and '*in general situations*'. If an unambiguous tokeniser makes a wrong choice, it may lead to a parse failure or incorrect semantic interpretation. Therefore, such multiword expressions must be carefully and conservatively designed.

[26] This example is adopted from [12].

[27] This is similar in principle to the use of the @→ operator defined by L. Karttunen and explained in [12].

9 Augmenting the Parser with Features

9.1 Feature Structures

Context-free grammars by themselves are not very convenient for capturing natural languages. One common extension is the use of *features*, which allows some aspects of natural language, such as agreement and subcategorisation, to be handled in an intuitive and concise way. In addition, it is possible to use features to provide semantic information and enforce selectional restrictions.

In order to increase the capabilities of the parser developed, it was decided to add the facility of specifying features in the lexicons and grammars. In particular, the PATR-II formalism described in [22] was adopted, the algorithm for unifying feature structures as directed graphs was taken from [1], while others were purposely developed.

9.2 Basic Concepts

Feature structures, f-structures or simply features are information-bearing structures used in unification-based NLP formalisms, such as PATR-II, LFG, GPSG and DCG. They are also called feature bundles, feature matrices or categories, functional structures, terms, or directed acyclic graphs, depending on the context in which they are being used.

Feature structures can be described as partial functions from features to values, the latter being feature structures as well. The simplest feature structure is the *atomic* feature, represented by a string *constant*. The second form is the *complex* feature structure, which maps atomic features to their atomic or complex values. The notation used for describing such a feature takes the form of a matrix

$$\begin{bmatrix} feature_1 : value_1 \\ feature_2 : value_2 \\ \cdots \\ feature_n : value_n \end{bmatrix}$$

where $feature_i$ is an atomic feature, while $value_i$ is an atomic or complex feature. An example of the latter is

$$\begin{bmatrix} feature_1 : value_1 \\ feature_2 : \begin{bmatrix} subfeature_1 : subvalue_1 \\ subfeature_2 : subvalue_1 \end{bmatrix} \end{bmatrix}$$

Since feature structures can be viewed as partial functions, the notation $D(f)$ can be used to denote the value associated with the feature f in the feature structure D. So for the example above, $D(feature_1) = value_1$, while

$$D(feature_2) = \begin{bmatrix} subfeature_1 : subvalue_1 \\ subfeature_2 : subvalue_1 \end{bmatrix}$$

As for functions, one can refer to the domain of a feature structure D as $dom(D)$, giving $dom(D) = \{feature_1, feature_2\}$ for the example above. A feature structure with an empty domain is called an empty feature structure, or a *variable*, and is denoted by $[\ ]$.

When handling feature structures, it is useful to define the notion of a *path*. This is a sequence of features, denoted by $\langle feature_1\ \ feature_2 \ldots\ feature_n \rangle$, which can be used to pick out a particular subpart of a feature by repeated application of the function D in the obvious way.

$$D(\langle\ \rangle) = D$$
$$D(\langle feature \rangle) = D(feature)$$
$$D(\langle feature_1\ \ feature_2 \ldots\ featue_n \rangle) = D(feature)(\langle feature_2 \ldots\ featue_n \rangle)$$

The result is undefined for features not in the domain of D. In particular, note that atomic features and variables have an empty domain and the only defined application on them is thus $D(\langle\ \rangle)$. Continuing the example above, then,

$$D(\langle\ feature_2\ \ subfeature_1\ \rangle) = subvalue_1$$

A final property of feature structures is that these can be *re-entrant*. A re-entrant feature is one in which two features in the structure share a common value. Thus, it is important to distinguish between a feature structure where features have distinct but similar values, as in

$$\begin{bmatrix} feature_1 : [subfeature : value] \\ feature_2 : [subfeature : value] \end{bmatrix}$$

from one where the features actually share the same, single value, as in

$$\begin{bmatrix} feature_1 :^1 [subfeature : value] \\ feature_2 :^1 \end{bmatrix}$$

Feature structures with shared values are usually notated as shown above, with shared features indexed the first time they appear, and further references to the same feature replaced by the index only. Alternative notations, such as using arrows to reference the shared features, are sometimes used.

9.3 Subsumption

There is a natural lattice structure for feature structures that is based on *subsumption*, an ordering on feature structures that roughly corresponds to the compatibility and relative specificity of information contained in them. Intuitively, a feature structure D subsumes a feature structure D', notated as $D \subseteq D'$, if D contains a subset of the information in D'.

Formally, a complex feature structure D subsumes a complex feature structure D' if and only if $D(l) \subseteq D'(l)$ for all $l \in dom(D)$ and $D'(p) \equiv D'(q)$ for all paths p and q such that $D(p) \equiv D(q)$. That is, paths must share the same value in D' if they do so in D. An atomic feature neither subsumes, nor is it subsumed, by any different atomic feature. Variables subsume all other feature structures, whether atomic or complex, because they contain no information at all.

9.4 Unification

Subsumption is only a partial order in that not every two structures are in a subsumption relation with each other. This can be because the feature structures have differing but compatible information, as in the following examples[28],

$$\begin{bmatrix} cat: & NP \\ agreement: & [number: sing] \end{bmatrix} \text{ and } \begin{bmatrix} cat: & NP \\ agreement: & [gender: masc] \end{bmatrix}$$

or because they have conflicting information, as in

$$\begin{bmatrix} cat: & NP \\ agreement: & [number: sing] \end{bmatrix} \text{ and } \begin{bmatrix} cat: & NP \\ agreement: & [number: plur] \end{bmatrix}$$

For the first pair of feature structures, however, there exists a more specific feature structure,

$$\begin{bmatrix} cat: & NP \\ agreement: & \begin{bmatrix} number: & sing \\ gender: & masc \end{bmatrix} \end{bmatrix}$$

that is subsumed by both, whereas no such feature exists for the second pair. The notion of combining the information from two feature structures to obtain a feature

[28] Adapted from [22], but with features as used in project.

structure that includes all the information of both is central to unification-based formalisms, and is called *unification*.

Note that there are many feature structures that may be subsumed by the same two feature structures. For instance

$$\begin{bmatrix} cat: & NP \\ agreement: & \begin{bmatrix} number: & sing \\ gender: & masc \\ person: & third \end{bmatrix} \end{bmatrix}$$

is subsumed by the above pair as well. In general, however, one is interested in the most general feature structure of this type, the one that contains all the information from the unified feature structures but no additional information. Formally, the feature structure D unifying the two feature structures D' and D'' is defined as the most general feature structure such that $D' \subseteq D$ and $D'' \subseteq D$. This is notated by $D = D' \cup D''$.

As shown above, not all pairs of feature structures can be unified in this way, because they contain conflicting information. In such cases, unification is said to *fail*.

9.5 Feature Structure Specification

Feature structures are used in PATR-II either to provide information about a lexical item, or to build information when combining constituents. In the former case, a feature structure is specified by declaring a set of constraints, in the form of equations, for the feature structure, along with some other lexically related information. One side of the equation is a feature path, while the other side may be either a feature path or a constant. These constraints are interpreted as follows.

An equation of the form $\langle f_1\ f_2 \ldots f_n \rangle = constant$, or equivalently $constant = \langle f_1\ f_2 \ldots f_n \rangle$, requires that the specified feature path be mapped to an atomic feature with the specified constant as value within the feature structure being

defined. The other form, $\langle f_1\ f_2 \ldots f_n \rangle = \langle f_1' f_2' \ldots f_m' \rangle$, requires that the two feature paths specified share the same value. Essentially, this means that any values mapped to those paths must unify successfully for the constraint to be satisfied.

When specifying feature structures for combining constituents, these equations are associated with a specific grammar rule, and the first feature in any feature paths specified refers to the particular category for which the constraint applies. Thus, for a grammar rule $X_0 \rightarrow X_1 X_2 \ldots X_l$, constraints take the form $\langle X_i\ f_1\ f_2 \ldots f_n \rangle = constant$ or $\langle X_i\ f_1\ f_2 \ldots f_n \rangle = \langle X_j\ f_1' f_2' \ldots f_m' \rangle$. The interpretation is similar to the above, but feature paths actually refer to paths in the feature structure associated with the corresponding X_i. With this notation, it is possible to specify constraints between constituents, as well as share information between them.

9.6 Lexical Templates

Formalisms that use feature structures are typically lexically oriented, meaning that they have simple combinatory rules operating on lexical items with quite complex associated information structures. In such cases, it is important to organise the lexicon in such a way as to remove redundancies and encode generalisations in an efficient manner.

One such technique is the use of templates, which are name bearing feature structures that can be used in specifying other feature structures. These are defined using the same format as for the lexical entries. For example:

Let *Sing* be $\langle agreement\ number \rangle = sing$.

In addition, templates can refer to any number of other templates, inheriting the structure of the latter. Thus,

Let *SingMasc* be *Sing*
$\langle agreement\ gender \rangle = masc.$

is equivalent to

Let *SingMasc* be $\langle agreement\ number \rangle = sing$
$\langle agreement\ gender \rangle = masc.$

Feature structures may thence be specified using the templates whose structures they are to inherit, along with any further particular constraints required for the particular structure.

Defining the templates in a hierarchical manner leads to a structure-sharing organisation that promotes reuse. However, this form of multiple inheritance can result in conflicting definitions. A specification for a template can cause a contradiction in constraints, due to either contradictions between the specification for the template and constraints in an ancestor template, or a conflict between two ancestor templates for the template being defined. The following definitions give an example of the latter situation.

Let $Parent_1$ be $\langle feature \rangle = value_1$.

Let $Parent_2$ be $\langle feature \rangle = value_2$.

Let *Child* be $Parent_1$
$Parent_2$.

The simplest approach to handling this problem is to disallow such an entry. Otherwise, a disambiguation technique is required to determine which definition is going to hold. One way to do this is to allow default inheritance, in which case a conflicting entry is resolved by allowing information lower in the template hierarchy to have a higher precedence over information specified in higher nodes. In the above

case, however, this does not suffice, since both parents of the child template would have the same precedence. Another technique, used in PATR-II, is to allow *overwriting* of features, specified using the operator $\Rightarrow$. For example, consider the definition

Let $Parent_1$ be $\langle feature \rangle = value_1$.

Let $Child$ be $Parent_1$
$\langle feature \rangle \Rightarrow value_2$.

In this case, no conflict arises, and $value_2$ overwrites $value_1$. Note that overwriting, unlike unification, never fails.

9.7 Data Structures for Representing Features

Feature structures can be viewed as rooted, directed graphs (but some formalisms require them be acyclic as well), whose arcs are labelled with feature labels. Each arc points to either another such graph or to a node with no outgoing edges, called a sink, that represents an atomic feature. For example, the graph for

$$\begin{bmatrix} feature_1 : v_1 \\ feature_2 : \begin{bmatrix} subfeature_1 : sv_1 \\ subfeature_2 : sv_1 \end{bmatrix} \end{bmatrix}$$

is shown in Figure 9.1.

When features share values, as in

$$\begin{bmatrix} feature_1 : {}^{1}\begin{bmatrix} subfeature_1 : sv_1 \\ subfeature_2 : sv_2 \end{bmatrix} \\ feature_2 : {}^{1} \end{bmatrix}$$

the corresponding arcs point to the same subgraph, as shown in Figure 9.2.

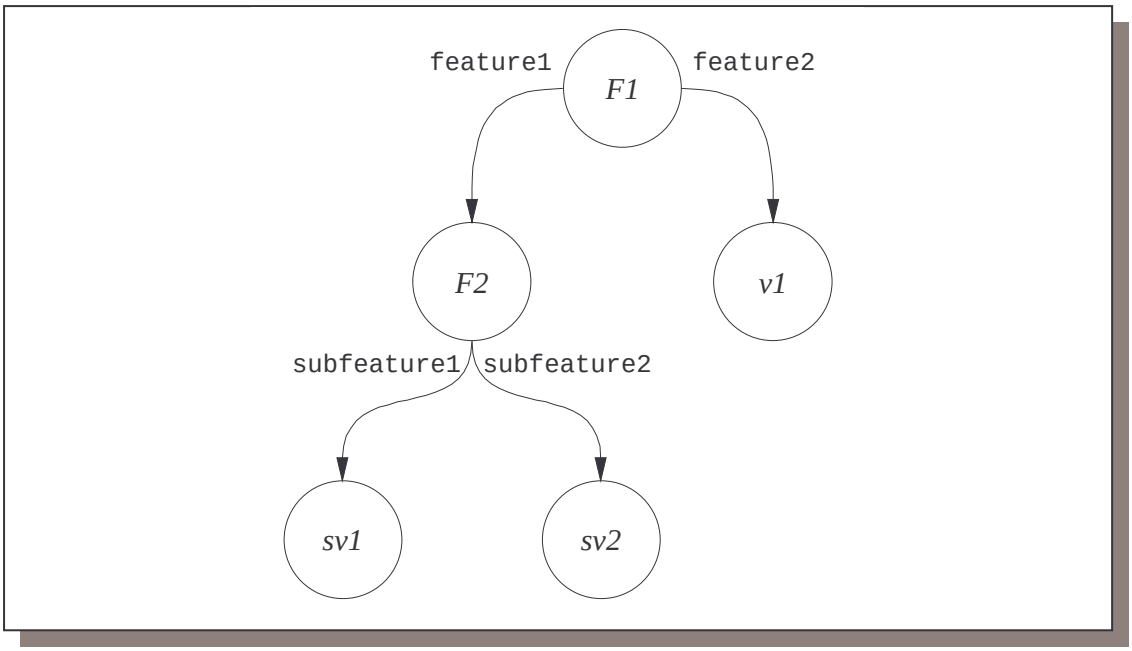


Figure 9.1 A Complex Feature Structure

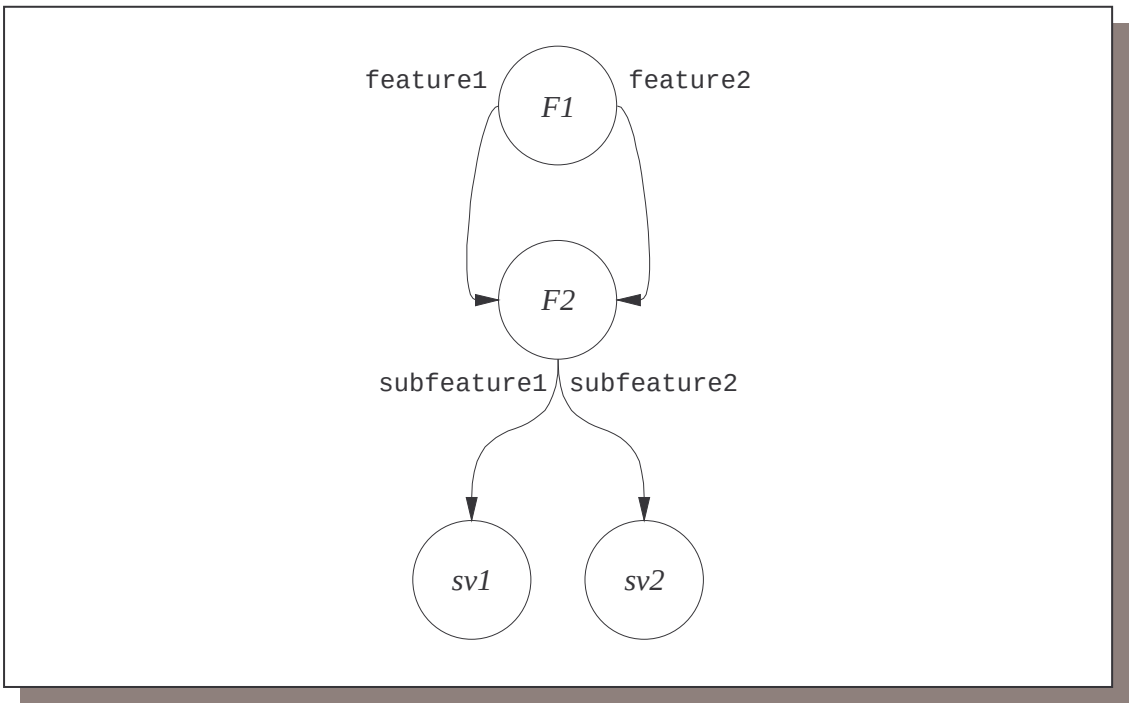


Figure 9.2 A Complex Feature Structure with Shared Values

Although not of theoretical importance, it is also useful to add to this representation special nodes, called forward pointers[29], which have a single, unlabelled, outgoing arc. The subgraph represented by a forward pointer is semantically equivalent to that obtained by replacing the node and its outgoing arc with the node to which this arc points, but maintaining any incoming arcs. Recursively, it is possible to remove all forward pointers from a graph, obtaining the equivalent, forward pointer free, graph. An example of a graph with a forward pointer and its collapsed version is shown in Figure 9.3.

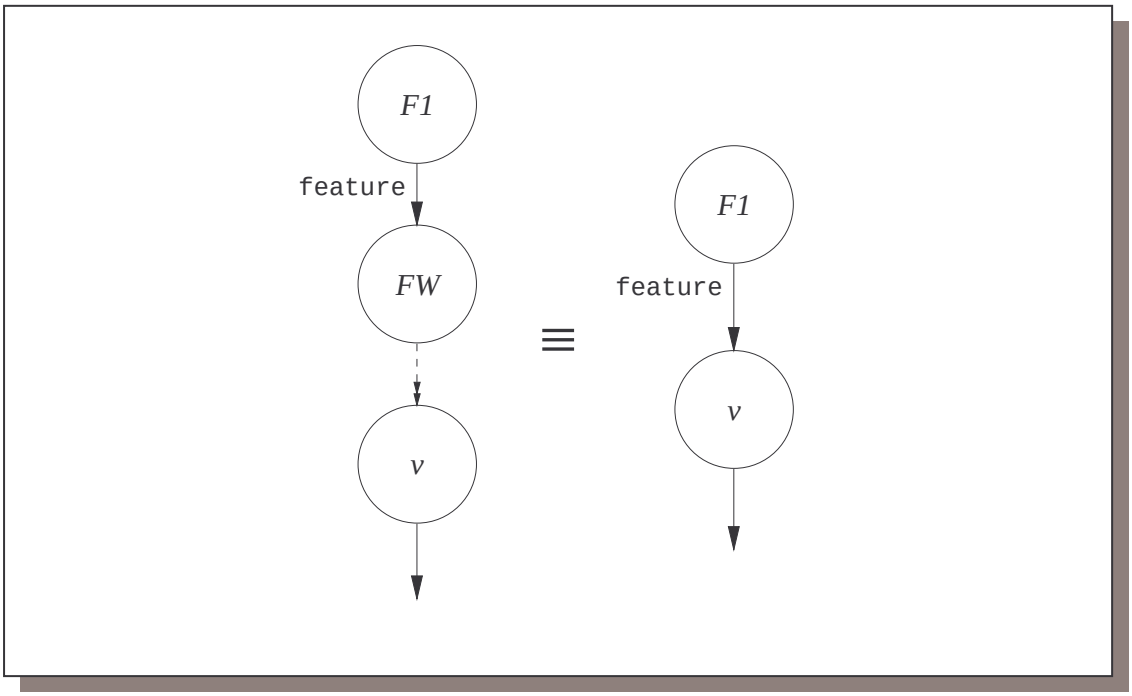


Figure 9.3 Forward Pointers

The practical relevance of this notion is that it facilitates the implementation of certain algorithms on feature structures which require that upon updating a node, all nodes having an arc to it are updated to point to the new node. With this approach, an

[29] This idea was adopted after analysing the implementation details of PC-PATR, an implementation of

implementation can simply turn the old node into a forward pointer to the new node. An alternative approach, which was also attempted, would be to store for each node information about which nodes have which arcs pointing to it, and then using this information to update each parent node to the new node. This, however, proved to be space and time inefficient, due to the extra overhead required for each node and the time spent updating each parent.

Note, that in literature, such as in [1] and [22], it is required that the graphs are acyclic. This requirement is not enforced here, and the algorithms developed to operate on feature structures take into consideration the possibility that there are cycles in the graph.

9.8 Modification Operations

The most basic operations developed for operating on feature structures are those to map and retrieve values, which were respectively called *put* and *get* operations. In line with the definition of feature structures, each of these operations can be operated upon with either a single feature or a feature path. These form the basis for the other algorithms.

The *put* operation attempts to modify the feature structure such that the specified path is mapped to the specified value. It fails if another value is already mapped to the same path, or if there is some prefix of the path that has an atomic feature mapped to it. The algorithm for this operation is very simple and can be easily deduced from the definition of feature structures, so it is not presented here. Note that this operation is meant to be destructive, in the sense that it does not attempt to unify the new value with the old.

PATR-II for PCs. For further details, refer to [15].

Conversely, the *get* operation is akin to applying the feature structure as a partial function. The algorithm can be easily derived from the recursive definition given above for the application of a feature structure to a feature path.

To unify a graph rooted at node N_i with a graph rooted at node N_j:

- If N_i is the same node as N_j, then return N_i and succeed.
- Else, if both N_i and N_j are sink nodes (that is, they do not have outgoing arcs), then if their labels are the same return N_i. Otherwise, the graphs do not unify.
- Else, if both N_i and N_j are not sinks, then create a new node N which will be the result of the recursive call. For each arc labelled F leaving N_i to node NF_i:

  1. If there is an arc labelled F leaving N_j to node NF_j, then unify recursively NF_i and NF_j. If these do not unify, then unification fails. Otherwise, build an arc labelled F from N to the result of the recursive call.

  2. If there is no arc labelled F from N_j, build an arc labelled F from N to NF_i.

  3. For each arc labelled F from N_j to node NF_j where there is no F arc leaving N_i, create a new arc labelled F from N to NF_j.

- Else, the graphs do not unify.

Figure 9.4 The Unification Algorithm

9.9 Implementing Unification

The unification algorithm for feature structures is based on their definition as directed graphs. The main algorithm was derived from that given in [1] for directed acyclic graphs, suitably modified to handle the structure given above. The algorithm is shown in Figure 9.4. In this and all the following algorithms, the issue that some nodes may be forward pointers is omitted. This is handled in unification, and all other algorithms, by simply following any forward pointer that is to be operated upon until a node that is not a forward pointer is found. Obviously, this means that algorithms must not allow a cycle containing only forward pointers to be created, as otherwise the procedure above would end up in an infinite loop.

Since this is a fundamental operation on feature structures, an example of its application will be given on the structures discussed in section 9.4, whose graphs are shown in Figure 9.5.

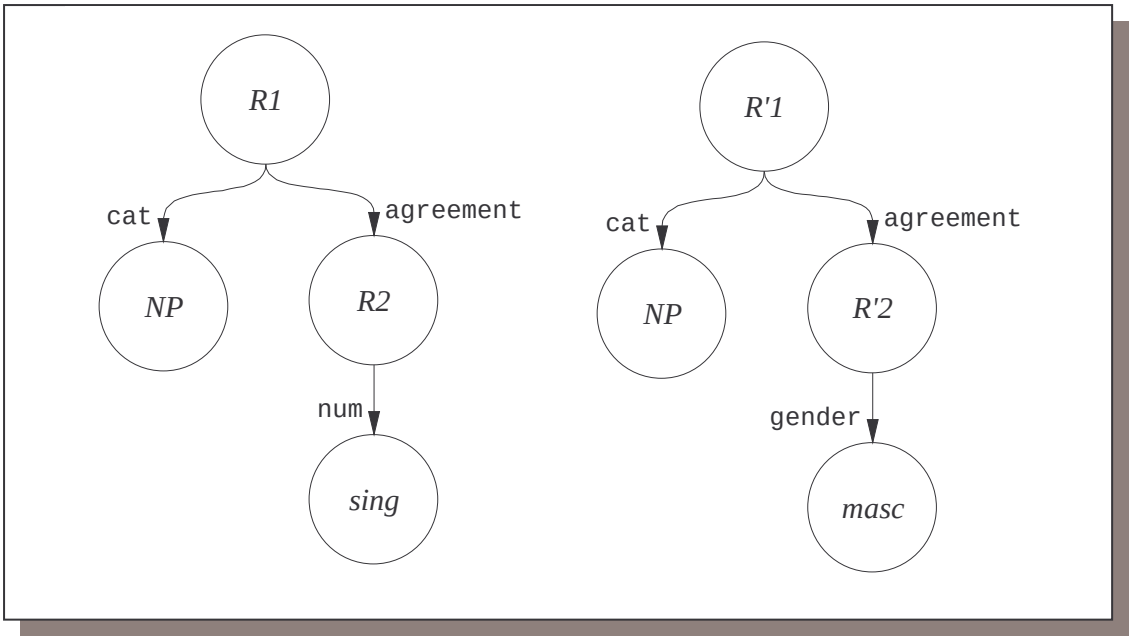


Figure 9.5 Unifying Directed Graphs

Starting with R_1 and R_1', these are both not sink nodes, so a new node R_1'' is created, and the algorithm proceeds to unify the subgraphs of the two root nodes. The first arc found is labelled *cat*, and, since both R_1 and R_1' have such an outgoing arc, the nodes at the end of this arc have to be unified by a recursive call to the algorithm. The latter are both sink nodes with the value *NP*, so unification succeeds and one of them is returned and linked to R_1'' with an arc labelled *cat*.

The algorithm then tries to unify the nodes pointed to by the arc labelled *agreement*, and finds that it has to unify R_2 and R_2' into the new node R_2''. This time, however, the nodes to be unified do not have arcs with common labels. Thus, from R_2'', an arc labelled *num* to the sink node *sing* and another labelled *gender* to the sink node *masc*, are built. The two levels of recursion of the algorithm are unfolded, resulting in the structure shown in Figure 9.6.

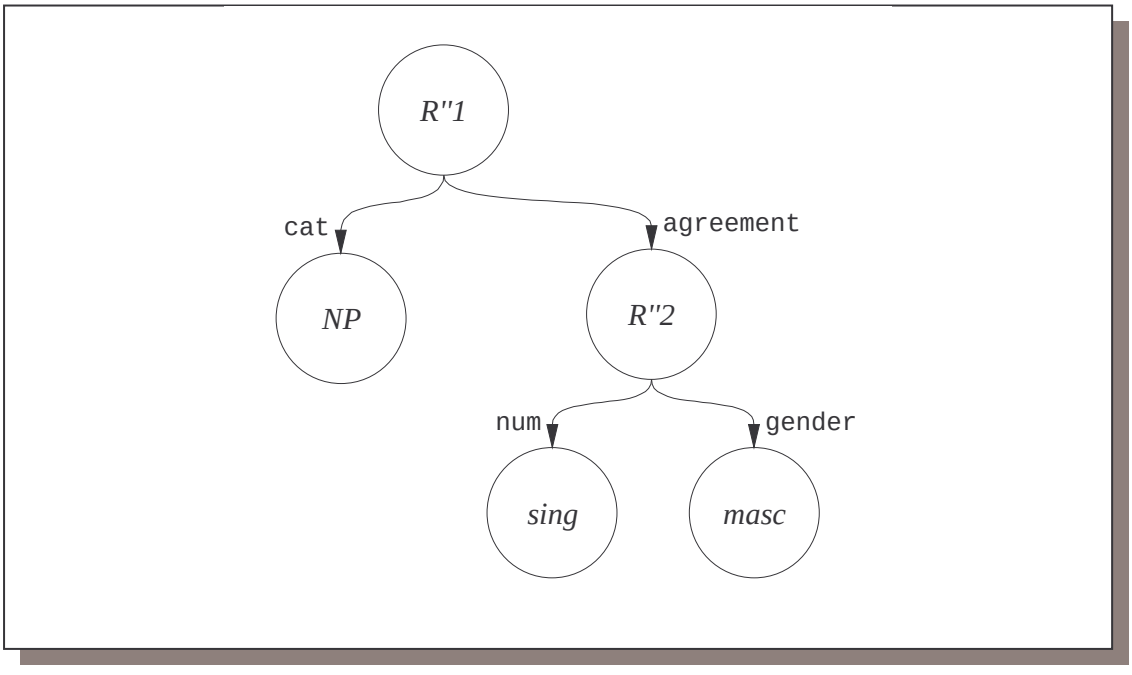


Figure 9.6 Result of Unification

9.10 Feature Equation Satisfaction

With the operations defined above, it is possible to determine the satisfiability or non-satisfiability of feature structure constraints and build feature structures corresponding to valid constraints.

The first form of constraint, $\langle f_1\ f_2 \ldots f_n \rangle = constant$, is trivially satisfied, or otherwise, by means of a *put* operation with the left and right hand side of the constraint as operands.

The other form of constraint, $\langle f_1\ f_2 \ldots f_n \rangle = \langle f_1'\ f_2' \ldots f_m' \rangle$, can be seen as a requirement for any values mapped to both paths to be unifiable to a single value, which is to be the value which both paths will be mapped to. The algorithm for satisfying this form of constraint is shown in Figure 9.7.

Retrieve the values for the first and second path. Then:

- If both values are null, map both paths to a new variable feature structure.
- Else, if only one value is not null, map the path pointing to the null value to this value.
- Else, try to unify both values. If successful, modify both paths to point to the unified value. If the values do not unify, the constraint cannot be satisfied.

Figure 9.7 Satisfying a $\langle f_1\ f_2 \ldots f_n \rangle = \langle f_1'\ f_2' \ldots f_m' \rangle$ Constraint

9.11 Duplicating a Feature Structure

Since feature structures may be modified when applying the above operations, it is necessary to duplicate a structure if the original features are to be retained. The simple recursive algorithm shown in Figure 9.8 does this, utilising a map from feature

structures to feature structures. Note that when duplicating a feature structure it is possible to make ado with any forward pointers by following them and only duplicating nodes that are not forward pointers. For simplicity, this is omitted from the algorithm.

Start with a new map from nodes to nodes.

To return the duplicate of node N :

- Lookup the N in the map. If there is a corresponding entry, return this entry.
- Else, if the N is an atomic feature, create a new node atomic feature N' with the same value, map N to N', and return N'.
- Else, the current node N is complex, so create a new complex node N', and map N to N'. For each arc labelled F from N to a node NF, recursively duplicate NF as NF', and make an arc labelled F from N' to NF'. Finally, return N'.

Figure 9.8 Duplicating a Feature Structure

9.12 Extracting a Textual Representation

Both for debugging and for illustration purposes, it was necessary to define an algorithm for extracting a string representation for a feature structure, in the format shown in previous sections. This requires knowing which values are shared in a particular feature structure, in order to assign indexes properly. This representation is thus derived by going over the structure twice, first to build a table of information on the usage of values, and secondly to derive the actual string representation.

Create a new map from features to information structures. Then:

1. Create a new stack and push the root node of the feature structure on it.
2. While the stack is not empty:
    - Pop the first element on the stack and retrieve information on it from the map.
    - If no information is available, create a new information structure and put it in the information map. If the node is a complex node, push all nodes that are reachable from it onto the stack. Go to step 2.
    - If information was available, see if this node has been seen multiple times. If not, mark it so, and create a new index for it. Go to step 2.
3. Return the information map created.

Figure 9.9 Building a Map of Information on Shared Features

The information required for each value in a feature structure, including the structure itself, is whether the structure is referenced multiple times and, if it is, the index assigned to it. A flag, initially set to false, is also required to determine whether, during the second phase of the process, a shared value has already been output, so that in that case only its index is turned into a string. The algorithms for the two phases are shown in Figure 9.9. and Figure 9.10, respectively. Note that the latter omits details concerning the proper alignment of the values and features when pretty-printing the string representation.

Build the information map for the feature structure, and create a string to hold the textual representation. Then:

- Retrieve information for the root node
- If the node is marked as having already been printed, append the index to the string and return the string.
- Else, mark the node as having been printed, and append an index if the node is used multiple times. Then:
    - If the node is atomic, append the its value and return the string.
    - Else, append an opening brace '[' and for each outgoing arc labelled F to node N, append F followed by ':' to the string and recursively append the textual representation of N. Finally, append a closing brace ']' and return the string.

Figure 9.10 Extracting a Textual Representation of a Feature Structure

9.13 Specifying Features in the Lexicon and Grammar

Having implemented feature structures and their corresponding algorithms, the next step involved modifying the lexicon and the grammar, which so far consisted only of context-free rules, to allow the utilisation of features as specified in section 9.5. Firstly, the format of the lexicon was modified so that it would be possible to specify feature templates using exactly the same syntax as that described in section 9.6. Following the templates, lexical entries were allowed to be specified using special delimiters to mark the start of an entry and its corresponding information, in the form shown in Figure 9.11[30].

[30] This format was adopted from [15].

\w $(word)+$
\c *category*
\f $(TemplateName \mid \langle f_1, f_2, \ldots, f_n \rangle = constant \mid \langle f_1, f_2, \ldots, f_n \rangle = \langle f_1', f_2', \ldots, f_m' \rangle)+$

Figure 9.11 Lexicon Entry Format

In effect, the word and category delimiters can be simply considered as another form of feature specification, and are actually treated as the constraints $\langle lex \rangle = word$ and $\langle cat \rangle = category$, respectively. Thus, the net result is that with each lexical entry is associated a feature structure satisfying the latter constraints, as well as any other specified after the feature delimiter, possibly using templates.

The grammar required a slightly different form of feature specification. The format for a grammar rule was defined as shown in Figure 9.12.

Rule $X_0 \rightarrow X_1 X_2 \ldots X_l$:
$(\langle X_i\ f_1\ f_2 \ldots f_n \rangle = constant)*$
$(\langle X_i\ f_1\ f_2 \ldots f_n \rangle = \langle X_j\ f_1' f_2' \ldots f_m' \rangle)*.$

Figure 9.12 Grammar Rule Format

With each rule of the form $X_0 \rightarrow X_1 X_2 \ldots X_l$, constraints of the form $\langle X_i\ f_1\ f_2 \ldots f_n \rangle = constant$ or $\langle X_i\ f_1\ f_2 \ldots f_n \rangle = \langle X_j\ f_1' f_2' \ldots f_m' \rangle$ were allowed. These also associate a feature structure with the rule, but paths of the form $\langle X_0\ f_1\ f_2 \ldots f_n \rangle$ are interpreted simply as $\langle f_1\ f_2 \ldots f_n \rangle$. The intuition is that such paths correspond to information for the head of the rule itself, with which the feature structure is actually associated. Thus, the corresponding feature structure of the rule would have the form

$$\begin{bmatrix} \dots & \\ X_1: & [\dots] \\ X_2: & [\dots] \\ \vdots & [\dots] \\ X_l: & [\dots] \end{bmatrix}$$

The following section will explain how this form is used to constrain the grammar rules that can apply during parsing. Note however, that the structure above requires that each X_i be unique. Otherwise, it would be ambiguous as to which constituent a particular rule applies. Thus, it was required that duplicate constituents be differentiated using indexes. Of course, during the parsing process, such indexes would be neglected for the purposes of combining edges.

9.14 Using Features During Parsing

After defining features in the lexicon and the grammar, augmenting the parser to utilise features was rather straightforward. With the notation of the Earley style chart parser, an edge now became a four-tuple:

$$[\textit{start vertex}, \textit{end vertex}, \textit{dotted rule}, \textit{feature structure}]$$

Where an edge is directly created from a lexical entry with the *scanner* procedure or from a grammar rule with the *predictor*, its corresponding feature structure is taken from that of the entry used to generate it. However, when combining an edge $[i, j, X \rightarrow X_1 \dots \bullet C \dots X_n, F_1]$ with $[j, k, C \rightarrow C_1 \dots C_n \bullet, F_2]$, it is now necessary to unify F_1 with $[C: \quad F_2]$, to obtain the feature structure for the resulting edge. If unification fails, then the new edge cannot be created. It is important to note that while the indexes used in the grammar specification do not affect the combination of the dotted rules, they do affect the unification test. Thus, if the active edge actually has the form $[i, j, X \rightarrow X_1 \dots \bullet \underline{C}\_i \dots X_n, F_1]$, then F_1 must actually unify with $[\underline{C}\_i: \quad F_2]$. This follows from the way the feature structures associated with

grammar rules were defined in the previous section. Note that this also means that the indices must be preserved in the dotted rules when the latter are created from grammar rules.

9.15 Using JavaCC for Lexicon and Grammar Readers

Due to the increased complexity in the formats of the lexicons and grammars, it was deemed an error prone task to develop readers[31] that take a lexicon and a grammar and construct the required objects for chart parsing.

Consequently, it was deemed feasible to learn the utilisation of *JavaCC* ([10]), a Java parser generator, for creating such readers. Having obtained knowledge of its functionality, reader specifications[32] for both lexicons and grammars were derived. Such specifications were processed by JavaCC to create readers capable of reading the lexicon and grammar specifications and create corresponding lexicon and grammar objects to be utilised by the chart parser. For documentation purposes, these formats were allowed to have single-line comments, whereby anything following the '*;*' character up to the end of the same line is ignored.

All this above is better illustrated diagrammatically in Figure 9.13. In addition, the resulting BNF for lexicons and grammars, created using the JavaCC related utility, *jjDoc*, can be found in Appendix B.

[31] *Parsers* would be a more appropriate term, but quite confusing in this context.

[32] *JavaCC grammars* is the intended term, but again this would lead to confusion with the MT system grammars.

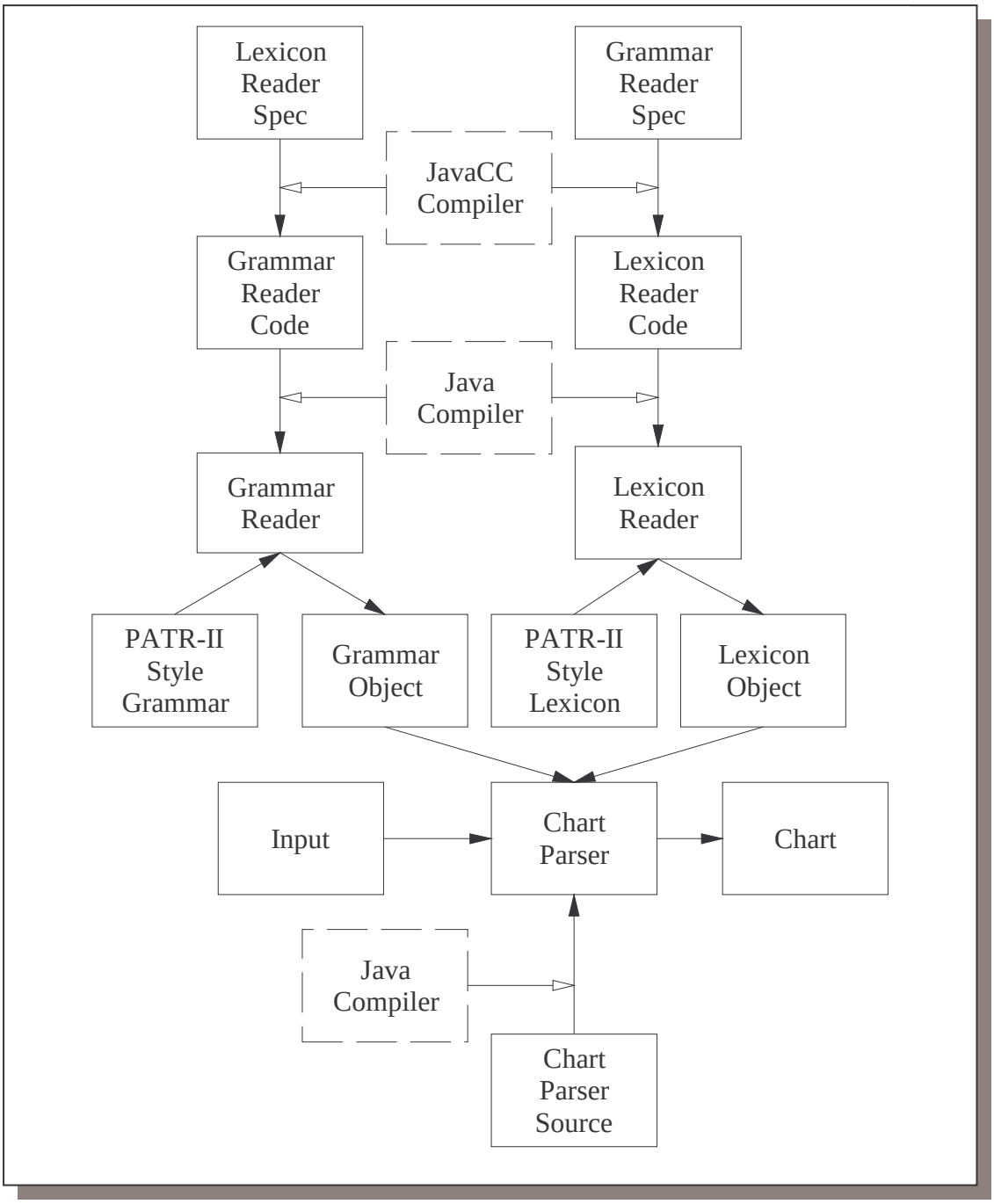


Figure 9.13 JavaCC for Lexicon and Grammar Readers

10 Selecting a Translation Approach

After examining the available corpus and translation patterns that emerged, a translation approach had to be selected. An initial attempt was made to define an interlingua for the available corpus. One possible initial approach was to build a hierarchy on the object properties and events in the weather reports, and build such a structure from a parse of the SL. This could have been something on the lines outlined in sections 5.3 and 5.4. However, this approached was dropped, and instead a transfer approach was adopted on the following accounts:

1. It was difficult to define a general hierarchy that would capture all the possible properties and events in all cases.
2. Such an approach would make the system very difficult to extend if it were desired to make the translation domain more general.
3. The closed domain of the corpus and the structural similarity of the SL and TL phrases, made it possible to identify easily recurrent transformation patterns, which in the limited context would make plausible translations.

Along with the adoption of the transfer approach, it was decided to make the system unidirectional from English to Maltese. This in part because the syntactic structure of the English language is much better known, so that it would present fewer problems for parsing.

10.1 Representing Trees with Feature Structures

Following the conceptualisation of feature structures as directed, cyclic graphs, it was realised that, as a special case, it is possible to represent any *n*-ary tree as a feature structure in the following manner. A leaf of the tree can be simply represented by an atomic feature with the same name as the value of the leaf node. A tree node with *n* children, on the other hand, can be denoted by a complex feature

structure with *n* unique, ordered features. For simplicity, these features can just be the numeric values $1, 2, \ldots, n$, although any other set of *n* feature names with an imposed ordering would do. The values mapped to these features represent in turn the subtrees of the root node in some specified left to right ordering. Figure 10.1 gives an example of such a feature tree.

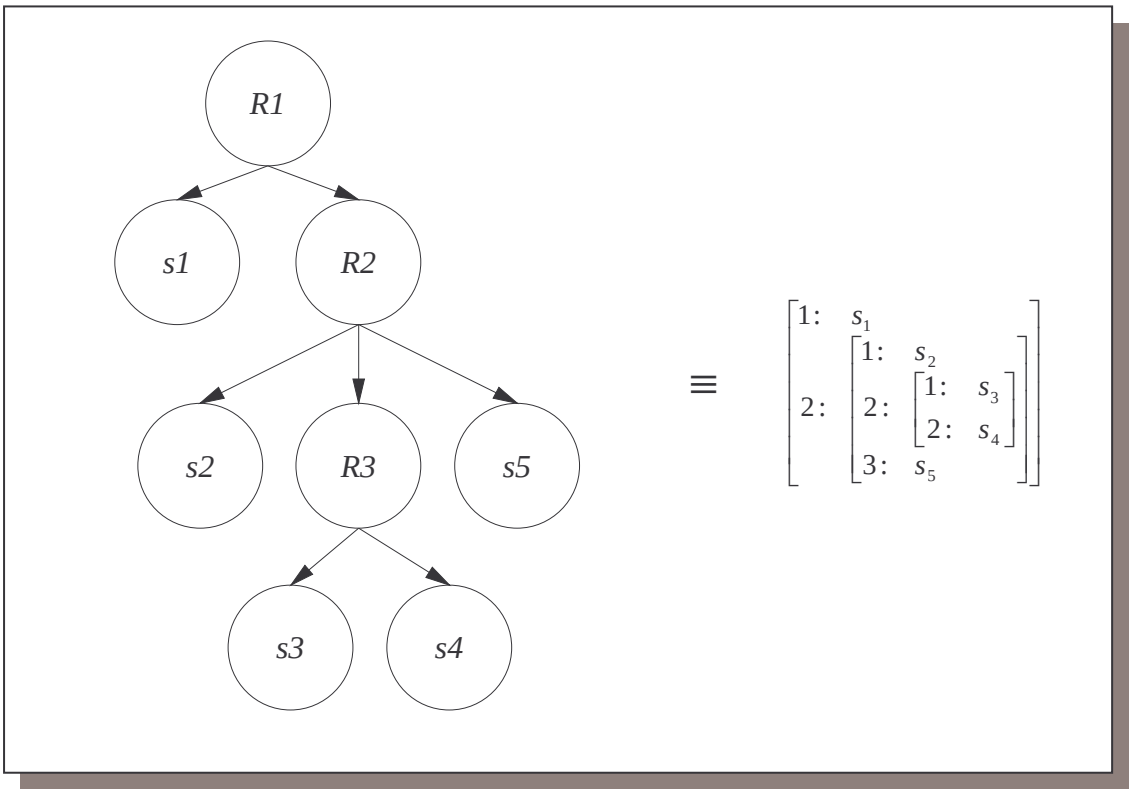


Figure 10.1 Representing *n*-ary Trees as Feature Structures

Of course, such trees can be of arbitrary depth and complexity, and feature paths can be used to refer to any subtree. In addition, it is possible to have other features and values within the feature structure without affecting the feature tree.

10.2 Defining a Tree-Combining Formalism

Being simply feature structures, feature trees can be constructed very simply using the equation constraints described in section 9.5. Of special interest here,

however, is how a feature tree can be construed out of any number of other feature trees. In order to simplify the presentation, the relevant operations will be concerned with how a feature structure T, with a feature tree mapped to the feature t, can be constructed from corresponding structures T_1 and T_2. However, the argument can be extended to any number of trees.

The basic idea is that the tree in T can be constructed either by explicitly specifying nodes and leafs, or by sharing subtrees from T_1 and T_2 through unification. The former technique is rather simple, one possible example being:

$$\langle T\, t\, 111 \rangle = s_1$$

With the latter, the tree can be built by simply referring to the trees of the other feature structures. For example,

$$\langle T\, t\, 1 \rangle = \langle T_1\, t \rangle$$
$$\langle T\, t\, 2 \rangle = \langle T_2\, t \rangle$$

results, through unification, in the following structure for T:

$$\begin{bmatrix} t: & \begin{bmatrix} 1: & {}^{1}[\ldots] \\ 2: & {}^{2}[\ldots] \end{bmatrix} \\ T_1: & {}^{1} \\ T_2: & {}^{2} \end{bmatrix}$$

That is, the tree for T is built using the tree of T_1 as a left subtree and that of T_2 as a right subtree. Swapping the order is a simple matter of specifying:

$$\langle T\, t\, 1 \rangle = \langle T_2\, t \rangle$$
$$\langle T\, t\, 2 \rangle = \langle T_1\, t \rangle$$

resulting in

$$\begin{bmatrix} t: & \begin{bmatrix} 1: & {}^{1}[\ldots] \\ 2: & {}^{2}[\ldots] \end{bmatrix} \\ T_1: & {}^{2} \\ T_2: & {}^{1} \end{bmatrix}$$

Sharing of subtrees need not be limited to the root node of T_1 or T_2. For instance

$$\langle T\, t \rangle = \langle T_1\, t1 \rangle$$

results in T having only the leftmost subtree of T_1. Conversely, a subtree from T_1 could be placed at a lower depth in the tree for T, as in

$$\langle T\, t12 \rangle = \langle T_1\, t \rangle$$

In addition, it is possible to specify constraints on whether the tree can be built or not, either by requiring a subtree to have a specific structure, or by requiring that subtrees have compatible information. For instance,

$$\langle T_1\, t1 \rangle = s_1$$
$$\langle T_1\, t1 \rangle = \langle T_2\, t \rangle$$

require that if the tree in T_1 has a left subtree, then it must be a leaf with value s_1, and that this must also be compatible with the tree in T_2, which in this case means that the latter must also be a leaf with the same value.

With this formalism, there is no notion of deletion, since the process is essentially one of construction. However, it is possible to achieve it indirectly, as shown previously, by simply sharing the subtrees of interest, and ignoring the others.

10.3 Feature Trees for a Transfer Formalism

As shown above, with the formalism developed, it is possible to create a tree from another set of trees using arbitrary movement, rearrangement, insertion and selection (as opposed to deletion) of subtrees. It is also possible to constrain the construction rules that can be utilised based on the structure of subtrees.

Consequently, this was seen as a valid formalism for the basis of a transfer technique. By specifying a representation tree structure in the lexicon for each lexical entry, it would then be possible to manipulate these at the phrase level, with the techniques shown above, through feature constraints in the grammar acting as transfer rules. These would create the output representation for the source simultaneously with the parsing stage. A final generation component would then be used to extract the target text from this output representation.

In practice, the choice of representation structure was taken to be directly the translation of the particular word or semantic unit. The leaf nodes of the translation trees were specified at the lexical level, mapped to a feature *trans*. Rules in the grammar were then made to operate on the translation trees of the subconstituents in order to create the translation tree for the head constituent.

10.4 Building a Generator Module

Owing to the simplicity of the target representation, defining a generator for the system proved rather trivial. Essentially, this involved creating an in-order traversal algorithm that flattened the translation tree by extracting the leaf nodes into a linear string. Having obtained a successful parse of the input, this algorithm was made to operate upon the translation tree of feature structures associated with the top-level parses in the chart.

In practice, it is possible to obtain several different parses for the input, depending on the grammar. In such cases, the corresponding translations may or may

not be unique. The generator was made to output only the unique translations as possible TL interpretations of the input, irrespective of the structure of the parse or the grammar rules involved.

11 Implementing the Approach with the Available Corpus

11.1 Creating the Lexicon

The part of the corpus in English was split up into the constituent words and was used as the basis of development for the system lexicon. After manually examining the corpus, entries were tagged with the corresponding category and other related information, such as number and gender. Lexical templates were used in order to capture common information for lexical entries, as shown in Figure 11.1. Use was also made of WordNet, a freely available lexical database for the English language ([16]) during this phase. For the purpose of extracting the words from the text archive, and looking them up in WordNet, some throwaway applications were also written. Due to the limited scope of the latter, they will not be discussed here.

```
Let Sing be <agreement number> = sing.
Let Plur be <agreement number> = plur.
Let Masc be <agreement gender> = masc.
Let Fem  be <agreement gender> = fem.
```

Figure 11.1 Use of Lexical Templates in the Lexicon

The lexicon created is full form, in that it does not allow any morphological descriptions on either the source or target language words. Thus, a separate entry is required for each inflectional or derivational form of a word stem. This makes the lexicon larger, but simplifies the task of parsing. However, for any word form found in the corpus, both its stem form and a subset of its relevant forms were added to the lexicon, even if these did not actually occur in the texts. This was done to make the lexicon more complete, allowing the system to handle related input whose exact

words had not occurred in the development archive. These were limited to third person, singular or plural forms, because other forms did not actually occur in the corpus. Since words can have multiple semantic values, it was also allowed to specify the same word multiple times with different feature specifications and selectional restrictions.

For the purposes of translation, a new field delimiter, '\t', was introduced in the specification of the lexicon. With this, it is possible to specify the particular translation of a word, or simply omit it if the corresponding TL construction requires so, such as is the case with the indefinite article in Maltese. The specified translation word or phrase is stored as the value of the feature *trans*. Thus, essentially the lexicon acts as a bilingual dictionary.

```
\w sunny
\c ADJ
\t xemxi
\f Sing Masc

\w sunny
\c ADJ
\t xemxija
\f Sing Fem

\w sunny
\c ADJ
\t xemxin
\f Plur
```

Figure 11.2 Entries for *sunny*

This required several translations for each word form, due to morphological differences between the source and target language. For example, the English adjective *sunny*, can have three different Maltese translations depending on the gender

and number of the phrase it modifies – *xemxi* (masculine, singular), *xemxija* (masculine, plural) and *xemxin* (plural). In such cases, different entries were listed in the lexicon and features were used to distinguish between the different interpretations. For example, the entries for the adjective *sunny* are shown in Figure 11.2, while their corresponding feature structures, are shown in Figure 11.3.

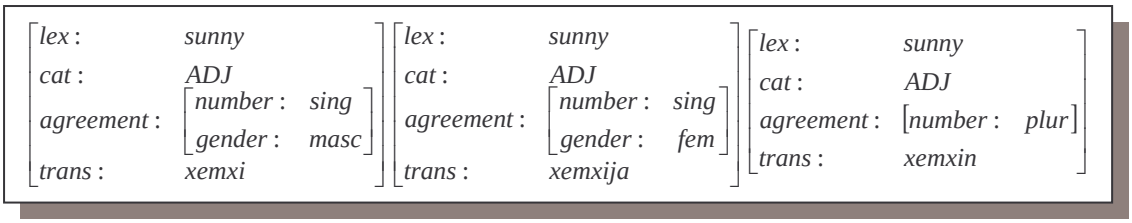


Figure 11.3 Corresponding Feature Structures for *sunny*

```
\w at times
\c PP
\t xi kultant

\w in the morning
\c PP
\t fil-_yodu

\w by evening
\c PP
\t sa fil-_yaxija

\w outbreaks of rain
\c NP
\t xita minn _hin _yall-_i_hor
\f Phen Plur
```

Figure 11.4 Collocations and Idioms in the Lexicon

In addition, certain collocations and idioms in the source archive were recognised as such and were inserted in the lexicon as single semantic units. Some examples are shown in Figure 11.4.

The count of the entries in each recognised category is shown in Table 11.1. A representative sample of the lexicon created can be found in Appendix C.

Category	**Count**
Nouns – *Generic*	68
Nouns – *Direction*	16
Nouns – *Month Names*	12
Nouns – *Place Names*	49
Verbs	83
Adjectives – *Generic*	125
Adjectives – *Quantitative*	3
Adverbs – *Generic*	19
Adverbs – *Direction*	16
Propositions	19
Articles	3
Auxiliaries	4
Conjunctions	3
Pronouns	2
Total	**422**

Table 11.1 Lexicon Category Counts

11.2 Creating the Grammar

As described previously, the parser developed allows the specification of a grammar with feature restrictions. In order to parse the source language, such a grammar was developed after analysing the syntactic structure of the available SL text. The grammar was loosely based on the syntactic structure of the English language (cf. [4]), but was empirically developed for the phrases in the archive, keeping part of it aside for evaluation purposes. In particular, attention was restricted to the entries of the *Weather Forecast* field.

Using the formalism described in section 10.2 rules were specified so that the translation tree for the head of a rule is created from the translation trees of the subconstituents in the rule. Other feature constraints were used to make restrictions on the possible parses, such as ensuring subject-verb agreement of the TL phrase. The transfer rules identified were rather simple, due to the simple correspondence between the SL samples and the TL versions. The most relevant ones were the omission of the indefinite article, which was left unspecified in the lexicon rather than in a transfer rule, and the rule shown in Figure 11.5, handling noun-adjective reordering from English to Maltese.

```
Rule NP → ADJP NP_1:
 <ADJP agreement> = <NP_1 agreement>
 <NP   agreement> = <NP_1 agreement>
 <NP   trans 1>   = <NP_1 trans>
 <NP   trans 2>   = <ADJP trans>.
```

Figure 11.5 A Transfer Rule in a Grammar Entry

The twenty-five rules making up the transfer grammar can be found in Appendix D. In addition, a sample of the structures that can be obtained by parsing with this grammar can be found in Appendix E.

12 Enhancing the System

12.1 Adding Syntactic Processing to the Generator

It was realised, both from personal analysis and from comments given by a third person, that certain Maltese translations were too rigid because of the lack of certain standard syntactic processing that is performed in the Maltese language. Thus, a first improvement to the system was to write a post-processor that would modify the base translations using a set of simple syntactic transformation rules, and couple this with the generator. These rules were derived from [7], a book on the Maltese language, from which the relevant translated excerpts follow.

12.1.1 Rules for the Maltese Article

In Maltese, the definite article is the letter *l* and is the same for any number and gender.

In front of nouns starting with a consonant, the article takes an *i* and becomes *il*. For example, we write *l-aħwa*, *l-art*, *l-ommijiet*, *l-oqbra*, *l-iżgħar* but *il-ġrieden*, *il-qattus*, *il-ħanfus*.

When applying the article to nouns that start with *m*, *n* or *s*, and have a following consonant, the nouns take an *i* in front. For example, *mqass*, *nsara* and *skola* become *l-imqass*, *l-insara* and *l-iskola*.

When the article is applied to a noun starting with one of *ċ*, *d*, *n*, *r*, *s*, *t*, *x*, *z* and *ż*, the article *l* is changed to the corresponding letter. For example, *iċ-ċirċ*, *id-dnub*, *in-nar* and *ir-raġel*. These letters are called *xemxin* (‘*sunny*’). The others are called *qamrin* (literally, ‘*of the moon*’).

The article *il* loses the *i* if the word before ends with a vowel. For example *marru l-ġnien*, *waslu x-xatt* and *hu u t-tifel*.

Maltese does not have an indefinite article, as the English *a*. It is possible to write instead *wieħed* or *waħda* instead, but this is usually left out.

12.1.2 Particles which join with the Article

Certain particles (Maltese *partiċelli*, similar to prepositions) that appear before an article may be joined together with the article, as shown in Table 12.1.

Particle	Joined Form
ma	mal-
sa	sal-
bi	bil-
fi	fil-
ta	tal-
dan	dal-
din	dil-
ġo	ġol-

Table 12.1 Particles that can join with the Article

For example, we write *mal-knisja*, *sal-wied*, *tal-ħaddiema*, *dax-xagħar*, *bil-fomm*, *bis-skiet*, *fil-ħajt*.

When the word starts with a vowel, however, it is possible not to join the article with the particle. For example, *ma' l-omm*, *ma' l-istat*. However, this does not hold for the particles in Table 12.2, which are never separated from the article.

Particle	Joined Form
minn	mill-
għal	għall-
bħal	bħall-
lil	lill-

Table 12.2 Particles that must be joined with the Article

Thus, we write *għall-omm, bħall-oqbra* and not *għal l-omm, bħal l-oqbra.*

12.1.3 The Particles '*bi*' and '*fi*'

The particles *bi* (*with*) and *fi* (*in*), when followed by a noun without an article, are usually abbreviated to *b'* and *f'* respectively, especially if the latter starts with a vowel. However, they retain their full form if the noun concerned starts with two consonants.

12.2 Handling Unknown Words

In traditional parsing, an unknown word makes a parser fail completely, since obviously no lexical information is available for that word. In fact, in the initial implementation, an unknown word made the parsing stage halt and fail completely. However, most MT systems are design to recover from such a situation, and it was thus decided to augment the tokeniser accordingly.

There are at least five[33] types of automatic error recovery for unknown words. Firstly, morphological rules can be used to guess the syntactic class of a word. For

[33] Four of them, excluding the statistical technique, are taken from [21]. This fifth is in fact described in [1].

example, a word ending with *ing* is likely to be a verb, while one ending with *ly* is likely to be an adverb. A second possibility is to use statistical information, for example based on Hidden Markov Models as described in [1], obtained from some representative text suite to estimate the most likely category of a word. For instance, if the word before the unknown one was *the*, then statistics would indicate that it has a very high probability of being a noun. It is also possible to combine these two approaches, utilising a conditional probability measure based on the morphology of the word. For instance, it would be possible to derive the probability that a word is a verb, given that its ending is *ing*.

A third method is to use punctuation or capitalisation as clues for the category of a word or a sequence of words. Thus, a word with a first capital letter is likely to be a proper name, while one in full capital letters might be an acronym. Certain language characteristics may also be taken into consideration in this case, such as the fact that in standard German, all nouns are written with an initial capital.

Fourthly, other specialised but regular formats can easily be marked as denoting dates, times, social security numbers and other such things written in a stylised manner.

Finally spelling correction routines can be used to find a word in the dictionary that is close to the input word. There are two popular models for evaluating closeness between words. In the letter-based model, an error consists of inserting or deleting a single letter, transposing two adjacent letters, or replacing one letter with another. There are some well-established techniques for evaluating the severity of error, such as the Levensthein string-edit distance algorithm, which dynamically computes the distance between two strings based on weighted string operations.

In the sound-based model, words are translated into a canonical form that preserves most of the information needed to pronounce the word, but abstracts away

some of the details. For example, the word '*attention*'[34] might be translated into the sequence [a, T, a, N, SH, a, N] where '*a*' stands for any vowel. The idea is that words such as '*attension*' and '*attenshun*' translate to the same sequence. If no other word in the dictionary translates to the same sequence, then one can unambiguously correct the spelling error.

Another alternative is to allow the user to provide information interactively on the new word. However, the system was meant to be automatic and was not designed for interactive updating, and besides, the user might be required to know the internals of the system intimately in order to provide suitable information. It would also add a burden on the user, who might not be interested in updating the lexicon during translation, but only concerned with obtaining an output.

The morphologically based method was finally adopted for the system, and the tokeniser was accordingly modified to handle unknown words before starting the parsing stage. When an unknown word is found, a simple guess on its category is made based on its suffix, and information is created for it in the lexicon. In this case, the translation of the word was made to be the same word, with an appropriate prefix[35] denoting that it was not originally found in the lexicon.

12.3 Partial Parsing

Another problem that caused the system to halt without a translation was when the input specified could not be parsed completely with the grammar available, although sub-parts of the text may have actually been parsed successfully. To handle this problem, the generator process was modified so that, if a top-level parse is not found, it reverts to translating any partial parses found.

---

[34] Example taken from [21].

[35] The prefix '@@' was used, which is also used for the same purpose in [18].

In general, the number of partial parses for a given text is exponential in the length of the input. Having no sound linguistic basis on which to select the correct combination of partial parses, a linear, deterministic approach was instead used when selecting the partial parses to translate, selecting the longest parses found moving from left to right. Since each partial parse can also have more than one possible translation, it was decided to output all the possible translations in the cross product of the partial translations.

Unfortunately, while obtaining a full parse gives a high chance of obtaining a reasonable translation, when the system reverts to utilising such fail-safe techniques, the number of translations increases due the combinatorial approach taken. Besides, the quality of translations obtained in such a manner is generally low due to the lack of relationship between the partial translations.

12.4 Exhibiting Preferences in Translation

One of the major ambiguity problems in the source texts was concerned with determining the number and gender of phrases such as '*sunny becoming partly cloudy*'. Originally, the system translated such a phrase into all possible interpretations, that is, singular masculine, singular feminine and plural. However, the translations in the corpus, as well as general translations from English to Maltese, give a preference to a singular masculine interpretation, failing any other indication.

With partial parsing implemented in the system, it was possible to define such a preference by requiring the top-level parses to have a singular masculine interpretation. If such a parse exists, then it would be chosen. Otherwise, that is, if a singular feminine or plural interpretation was found, the system resorts to partial parsing. However, such a parse would be longest spanning in the input, so it would be chosen for translation anyhow.

12.5 Providing a Visual and On-Line Interface

In order to facilitate the utilisation of the translation system developed, an easy to use visual interface was developed. The main interface allows the user to specify the text to translate, either by editing it directly or by loading it from a text file. The user can then initiate translation and see the results in a different text area. It is also possible to browse the contents of the lexicon that the system uses for translation using a separate dialog.

In order to make the system and its output easily available, for example, to allow evaluation by remote users, it was decided to make the system, as well as the related data-files, accessible on-line through a Web browser. This is also in line with the current trend of Internet integration exhibited by modern MT systems (cf. [24] and [26]).

One way to provide this could have been to develop a server-thick, client-thin environment, whereby the translation application would reside on the remote server and the user specifies the source text through a Web form. Upon submission of the form, the text is translated on the server side and the result relayed back to the user in a dynamically created web page. This is efficient from the point of view of the client, but requires considerable effort in handling system dependent details, such as CGI setups and the launching of daemon servers, from the developer in order to set up properly.

A simpler solution, which was adopted in practice, was to wrap up the application in a Java applet that can be loaded directly in a browser. This is far simpler from the point of view of implementation, but adds a burden on the user who has to download the system in its entirety (approximately 95 Kilobytes) before being able to utilise it. It also requires the Java 1.2 appletviewer or a browser with Java Plug-In 1.2 (see [9] for details on these) in order to run properly, since current browser do not currently support Java 1.2 applets. Unfortunately, some of the features, such as the loading of source files from disk, are disabled due to the network

security features employed in Java applets. At the time of writing, the system can be accessed on-line at the address http://www.cis.um.edu.mt/~pfar/project.html.

13 Evaluating the System

13.1 Quality of Output

In order to assess the quality of the output produced by the system, an accuracy evaluation method was utilised. A set of phrases, which had been reserved for testing purposes, was translated and the output was used to create evaluation forms. In these, each SL phrase was matched with the corresponding TL translation or translations. Each single translation could then be marked according to a discrete scale with values '*Excellent*', '*Good*', '*Fair*' and '*Bad*'. A sample of such a form, including output from the system, can be found in Appendix E.

These evaluation forms were given for correction to a third person who was unaware that the translations had been created by machine. The person was then instructed to mark each translation according to how well he felt that a particular translation was valid with respect to the source phrase. While the person was fluent in both the source and the target language, he had no special translation or linguistic knowledge. Thus, the informal scale used was necessary, since one with more detailed linguistic specifications might not have been comprehensible.

The results were tabulated in order to derive a quality profile. For each phrase, the number of translations with a particular grade, as well as the total number of translations, were counted. These values were again summated, and used to create the graph shown in Figure 13.1.

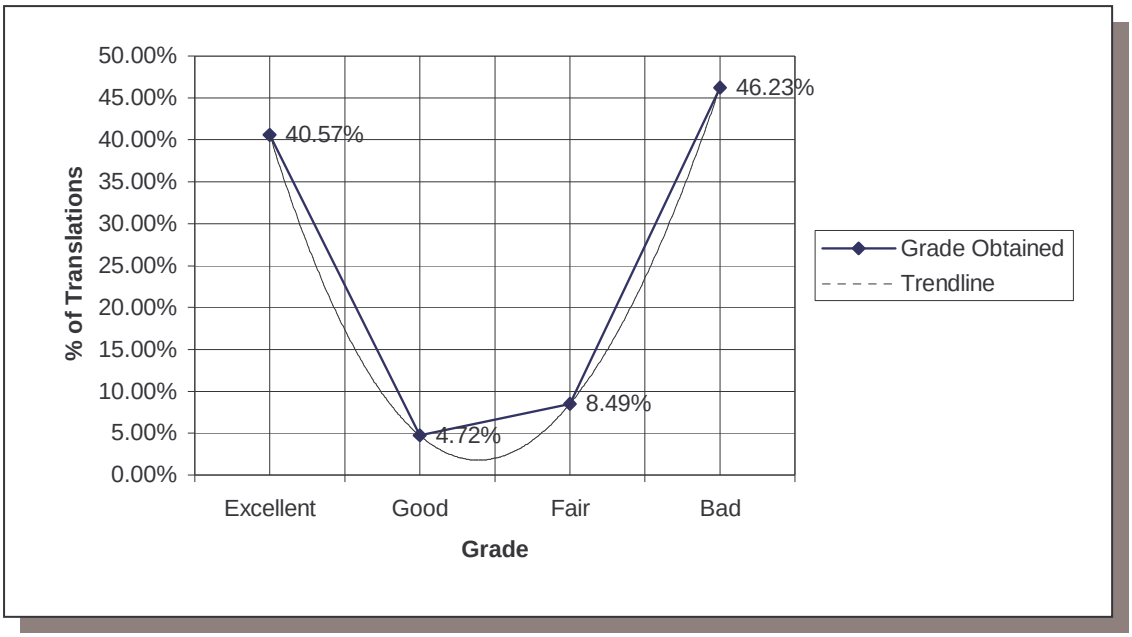


Figure 13.1 Evaluation Results

The evaluation was based on 52 unique phrases. On average, the system generated two translations for each phrase, although in practice 56% of the phrases had a single translation, while the others had two or more. When an incomplete parse was found, there could be as much as eight, due to the combinatorial partial translation approach described in section 12.3. As can be seen from the graph, about 41% of the translations were considered excellent, while 46% were considered completely bad. Only 5% were considered good, while 8% were considered fair.

As described in section 6.2.4, it is difficult to give an exact interpretation of this information. Based purely on the data, one could conclude that the system tends to generate two translations, one that is valid and another that is not. However, scrutinising the evaluation forms, it was found that it was usually the case that either a single valid translation was given, or a larger number was output, of which one was marked excellent and the others as bad. Most of the errors, in such a case, were due to structural ambiguity, where, for example, it was ambiguous whether an adjective modified the following noun or the larger noun phrase that started with that noun. For

instance, translating *partly cloudy with isolated showers at first* gives both the correct *xi ftit imsaħħab b'ħalbiet tax-xita iżolati għall-ewwel,* where *isolated* modifies *showers*, and *xi ftit imsaħħab b'ħalbiet tax-xita għall-ewwel iżolati, isolated* modifies *showers at first*. As a Maltese phrase, this latter is not linguistically happy, and could be intended as meaning that the showers will only be isolated at first, and may not be so later. This phenomenon occurs due to the generic form of rules such as that given in section 11.2.

13.2 Adequacy Evaluation

The aims of the project were to investigate the process of translation, rather than to automate some human translation process in particular. The archive and the system were derived without any reference to the context in which the translation of the weather reports actually takes place. Thus, it is not really appropriate to evaluate the fitness for purpose of the system. However, some comments may be given with respect to the theoretical replacement of the manual translation with this MT application.

In terms of execution speed, the system is certainly adequate, with a response for a typical phrase obtained nearly instantly on a typical PC. The input and editing capabilities are also flexible. No pre-editing is assumed, but post-editing is required for selecting the best translation, and possibly amending this as well. However, the translation task is very small and circumscribed, and for someone used to translate these highly stylised phrases, it might even be quicker to translate them by hand. This also depends on the format of the source phrase. If this is, say, obtained from a network, it could be possible to arrange for the system to receive it and translate it without human intervention. (This, in fact, was done with the METEO system.)

If source phrase has to be input manually in any case, then the system might seem more of a hassle, rather than an aid, to the user. It could be possible, however, to justify the system more if this were tailored better to the environment. For example,

facilities could be added to reproduce the exact format of the weather forms described in 7.1, and have these printed or sent as email to interested clients.

13.3 Architectural Considerations

The system designed clearly does not attempt to extract an interlingual representation of the input, and in fact it lies somewhere between a transfer and a transformer system, with all the limitations that the latter entails. Figure 13.2 shows an attempt to characterise the system as compared to the architectures described in section 4.

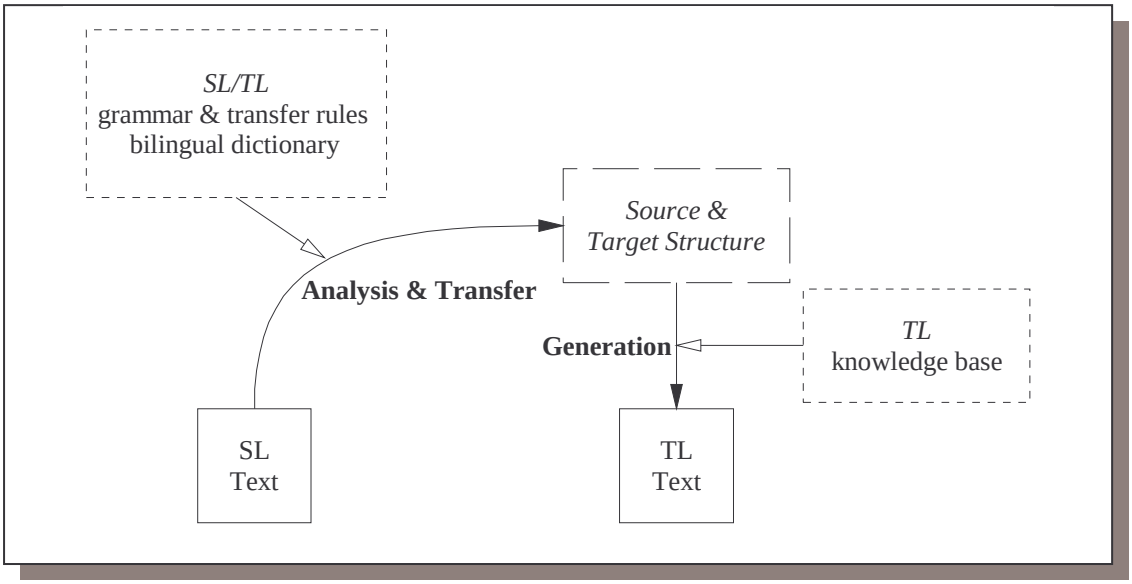


Figure 13.2 System Architecture

Starting with the dictionary, it is primarily an SL lexicon, but since information was added which is really about the TL, it can be ostentatiously called a bilingual dictionary. This is pretentious, since the correlation specified at the lexical level is obviously too shallow for an MT system that would like to claim knowledge of the correspondences of the two languages.

The grammar suffers from the same criticisms. It is essentially an SL grammar, with transfer rules added on to create the target structure during parsing. This structure is shown in the parse samples in Appendix E. Both the coupling of the grammar with the transfer rules and the embedding of the target structure in the source one were obviously not good design choices. In addition, the use of feature trees as a representation structure is too shallow, especially since at best it gives only the syntactic structure of the output.

As regards the generation process, it suffers greatly from the lack of an independent TL grammar, although it does have some linguistic knowledge embedded in the form of syntactic manipulation operations. The omission of a TL grammar has to be considered against the fact that no computational grammar exists for the Maltese language, and would probably be quite hard to define due to the informal character of this language. The relative success of the translation process relies quite heavily on the fact that the Maltese language has a very low threshold as to what accounts as a valid locution.

14 Implementation Details

14.1 Language Choice and Design Structure

The implementation was carried out using the Java Platform 1.2. The code was written in a highly object-oriented fashion, as appropriate to the Java language. The code was split into six Java packages. Each of these is concerned with a major part of the system.

Package	Purpose
mt.features	classes concerning the data structures and manipulation operations for feature structures
mt.lexicon	classes involving the reading and manipulation of lexicons
mt.grammar	classes involving the reading and manipulation of grammars
mt.parser	classes involving the implementation of a chart parser
mt.translation	classes involving the process of translation
mt.visual	classes providing a visual and on-line interface

Table 14.1 Code Package Structure

Complete details on the functionality of each class, as well as on the relevant fields and methods can be found by browsing the JavaDoc HTML documentation

provided with the program. An overview of the main classes in each package is however given in the sections below.

14.2 Package mt.features

This package contains the main class *FStructure*, which represents a feature structure, as well as corresponding classes for feature paths and constraint equations.

14.2.1 Class FStructure

This implementation of a feature structure follows closely the design given in section 9.7. Internally, a field of type *Object* is used to define the type of feature structure. An atomic feature structure simply stores a *String* object in this field, while a complex feature structure stores a map from *String* objects to other *FStructure* objects. If another *FStructure* is stored in this field, then the *FStructure* becomes a forward pointer. Utility methods are supplied to determine the type of the *FStructure* object, as well as for following forward pointers to reach a non-pointer *FStructure*.

This class also implements all the operations described for feature structures in section 9, with all the implementations following closely the algorithms described there. With regards to translation, it also provides the method *extractTranslation()*. This method traverses the translation tree built into the feature structure using in-order traversal and returns the string containing the lexical items in order. This method actually performs part of the generation process. Thus, it would have been more appropriate to define it in the package *mt.translation*. However, it was inserted here since it needs to access implementation dependent parts of an *FStructure*, which were hidden from other packages to provide data abstraction.

14.2.2 Class FStructureFormatException

This is an *Exception* class that is thrown from methods when a modification operation on a feature structure, such as unification, fails, or when an attempt is made to retrieve a value from an invalid path.

14.2.3 Class FeaturePath

Class representing a feature path, that is, a list of feature labels.

14.2.4 Class FeatureEquation

An abstract base class for *FeatureEquationConstant* and *FeatureEquationPath* classes, allowing some generic manipulation for object of these subclasses.

14.2.5 Class FeatureEquations

Class representing a set of *FeatureEquationConstant* and *FeatureEquationPath* objects. This class defines a method *toFStructure()*, which builds an *FStructure* object satisfying each of the constraints. This is done by first satisfying any $\langle f_1\ f_2 \ldots f_n \rangle = constant$ constraints on a variable feature structure using the *put* operation, and then satisfying the $\langle f_1\ f_2 \ldots f_n \rangle = \langle f_1'\ f_2' \ldots f_m' \rangle$ constraints, as described in section 9.10.

14.2.6 Class FeatureEquationConstant

Class representing an equation constraint of the form $\langle f_1\ f_2 \ldots f_n \rangle = constant$.

14.2.7 Class FeatureEquationConstantSet

Class representing a set of *FeatureEquationConstant* objects.

14.2.8 FeatureEquationPath

Class representing an equation constraint of the form $\langle f_1\ f_2 \ldots f_n \rangle = \langle f_1' f_2' \ldots f_m' \rangle$.

14.2.9 Class FeatureEquationPathSet

Class representing a set of *FeatureEquationPath* objects.

14.3 Package mt.lexicon

This package contains classes related to the reading and manipulation of lexicons. Some of the classes were generated using JavaCC from the specification given in *LexiconReader.jj*. The most relevant classes are briefly described below. Other classes that were automatically created for the lexicon reader are omitted.

14.3.1 Class Lexicon

Class representing a lexicon. It implements methods for adding entries, looking up words and loading a lexicon from a file. Entries are indexed by word, for the parsing purposes described in section 8.4.

14.3.2 Class LexiconEntry

Class representing an entry in a lexicon, specifying the category, translation and features corresponding to a specific word.

14.3.3 Class LexiconReader

Class capable of reading a lexicon specification from any arbitrary character stream, and create a corresponding *Lexicon* object. It is the main class created by JavaCC after reading the parser specification given in *LexiconReader.jj*. The BNF for the lexicon specification is given in Appendix B.1.

The method *Lexicon()* starts the reading procedure, which parses the associated character stream and returns a *Lexicon* object containing the *LexiconEntry* objects corresponding to the lexicon specification. If an error in the input format or a feature specification which cannot be satisfied is encountered, an appropriate exception is generated and the reading stops.

This reader class first reads any templates defined, making sure that any parent templates mentioned are defined previously. It does not check whether the templates themselves are validly specified. It then reads a list of one or more lexicon entry specifications, making sure that any templates mentioned are defined, and that the definition of the templates and the constraints for the particular features are consistent. This is done by deriving the set of equation constraints for an entry, both those specific to the entry and those defined through any specified templates, creating a *FeatureEquations* object and calling its *toFStructure()* method.

14.3.4 Class LexiconTemplate

Class representing a lexical template, described in section 9.6, which is used during the reading process.

14.3.5 Class LexiconTemplateList

Class representing a list of *LexiconTemplate* objects.

14.3.6 Class LexiconTemplateNameList

Class representing a list of template names, used to maintain the list of names of parent templates for each *LexiconTemplate* object, thus maintaining a template hierarchy.

14.4 Package mt.grammar

This package contains classes related to the reading and manipulation of grammars. Some of the classes were generated using JavaCC from the specification given in *GrammarReader.jj*. The most relevant classes are briefly described below. Other classes that were automatically created for the grammar parser are omitted.

14.4.1 Class Grammar

Class representing a grammar. Essentially, this is a list of *GrammarRule* objects, with facilities to add rules and load a grammar from a file. For efficiency during parsing, rules are indexed by the category on the left-hand side, as explained in section 8.4.

14.4.2 Class GrammarParameters

Class representing a set of parameters for a grammar. The only parameter that can be specified is the start symbol of the grammar.

14.4.3 Class GrammarRule

Class representing a grammar rule. This consists of a context free rewrite rule, and a corresponding set of restrictions specified as a feature structure, obtained from the grammar specification as described in section 9.13.

14.4.4 Class Grammar Reader

Class capable of reading a grammar specification from any arbitrary character stream, and create a corresponding *Grammar* object. It is the main class created by JavaCC after reading the parser specification given in *GrammarReader.jj*. The BNF for the grammar specification is given in Appendix B.2.

The method *Grammar()* starts the reading procedure, which parses the associated character stream and returns a *Grammar* object containing the

GrammarRule objects corresponding to the grammar specification. If an error in the input format or a feature specification which cannot be satisfied is encountered, an appropriate exception is generated and the reading stops.

14.5 Package mt.parser

14.5.1 Class Chart

Class representing the chart obtained by parsing an input text with a specified lexicon and grammar. It is instantiated using a *Lexicon*, a *Grammar* and an *Input* object. Upon instantiation, the input is parsed according to the lexicon and grammar specified. The implementation of the parse procedure follows closely the definitions given in section 8.4 for the Earley style chart parser. It is also possible to specify a *ChartListener* object, whose methods are called during updates of the parsing.

This class also provides methods to access the edges using both the $chart(j)$ and the $chart(i, j)$ notation. Their interpretation is exactly like that given in sections 8.2 and 8.3. Other utility methods are available for retrieving complete top-level parses of the input.

14.5.2 Interface ChartListener

Interface that can be implemented by classes that want to listen to events occurring during parsing. The events include the setting of the number of vertices, the processing of a new vertex and the conclusion of the parsing. An implementation of this interface is used in the visual interfaces in order to provide a cue to the user as to the state of the parse.

14.5.3 Class Category

Class representing a grammatical category.

14.5.4 Class CategoryList

Class representing a list of *Category* objects.

14.5.5 Class Edge

Abstract class representing an edge in a chart. It is the super class of the *LexicalEdge* and *RuleEdge* classes.

14.5.6 Class EdgeList

Class representing a list of *Edge* objects.

14.5.7 Class LexicalEdge

Class representing an edge in the chart created using a lexicon entry.

14.5.8 Class RuleEdge

Class representing an edge in the chart created using a grammar rule.

14.5.9 Class Input

Class representing the input text. It is instantiated using any arbitrary character stream and a lexicon, and performs the tokenisation of the input, as specified in section 8.5, before starting the parsing process. It is also responsible for handling unknown words, according to the method specified in section 12.2. This is done by inserting entries in the lexicon for any unknown words found according to a very simple morphological recogniser.

14.5.10 Class InputException

Exception class thrown when an error occurs during the parsing of the input. Originally, it was thrown when an unknown word was found or when the text could not be parsed with the available rules. After the introduction the fail-safe procedures

described in sections 12.2 and 12.3, that is, the handling of unknown words and partial parsing, this class fell into disuse.

14.6 Package mt.translation

14.6.1 Class Translator

Class responsible for performing the main translation process, based upon the facilities provided by classes in other packages. It is instantiated with specific lexicon and grammar files, '*english.lex*' and '*english.grm*', containing the appropriate features and rules for translation. Samples of the lexicon used by this class are given in Appendix C, while the grammar is shown in Appendix D.

Translation on input text is performed by creating a chart using the specified lexicon and grammar. If top-level parses are found for the input, the translations are extracted from the corresponding features structures by calling their *extractTranslation()* method, duplicates are removed, post-processing is performed as described in section 12.1, and the resulting output strings are returned. Otherwise, it resorts to translating the longest partial parses from left to right, returning the strings in the cross product of the partial translations, as discussed in section 12.3.

14.7 Package mt.visual

14.7.1 Class Translator

Class providing a visual interface to the translating system. It provides facilities for inputting and editing the SL text, including the facility to load or store this from or to a file. It is also possible to initiate parsing or browse the contents of the lexicon by selecting appropriate buttons or menu items. In addition, it specifies a dialog that shows the progress of the translation process, and another that displays lexicon entries.

14.7.2 Class TranslatorApplet

Class which wraps up the visual *Translator* class into an applet application.

14.7.3 Class TranslatorOptionPane

Class providing some utility methods for the visual interface.

15 User Manual

15.1 Running the Program as an Application

The program requires Java Platform 1.2 installed in order to run. Assuming the Java interpreter is in the executable path and the program jar file, *translator.jar*, is in the current directory, the main program can be run by issuing the command:

java –classpath translator.jar mt.visual.Translator

15.2 Running the Program as an Applet

The program can also be run as an applet using the same jar file. This requires an HTML file containing the applet tag shown in Figure 15.1, with the jar file residing in the same directory as the HTML document.

```
<APPLET
   CODE="mt.visual.TranslatorApplet.class"
   ARCHIVE="translator.jar"
   WIDTH=150
   HEIGHT=60>
```

Figure 15.1 Applet Tag for Running the Application On-Line

The applet can be viewed using the appletviewer utility supplied with the Java 1.2 Platform. Alternatively, the applet tag can be appropriately modified so that it can be viewed using any browser with the Java 1.2 Plug-In. Further details on how this can be done are given in [9].

15.3 Using the Program

Upon loading, the program displays the main window, shown in Figure 15.2.

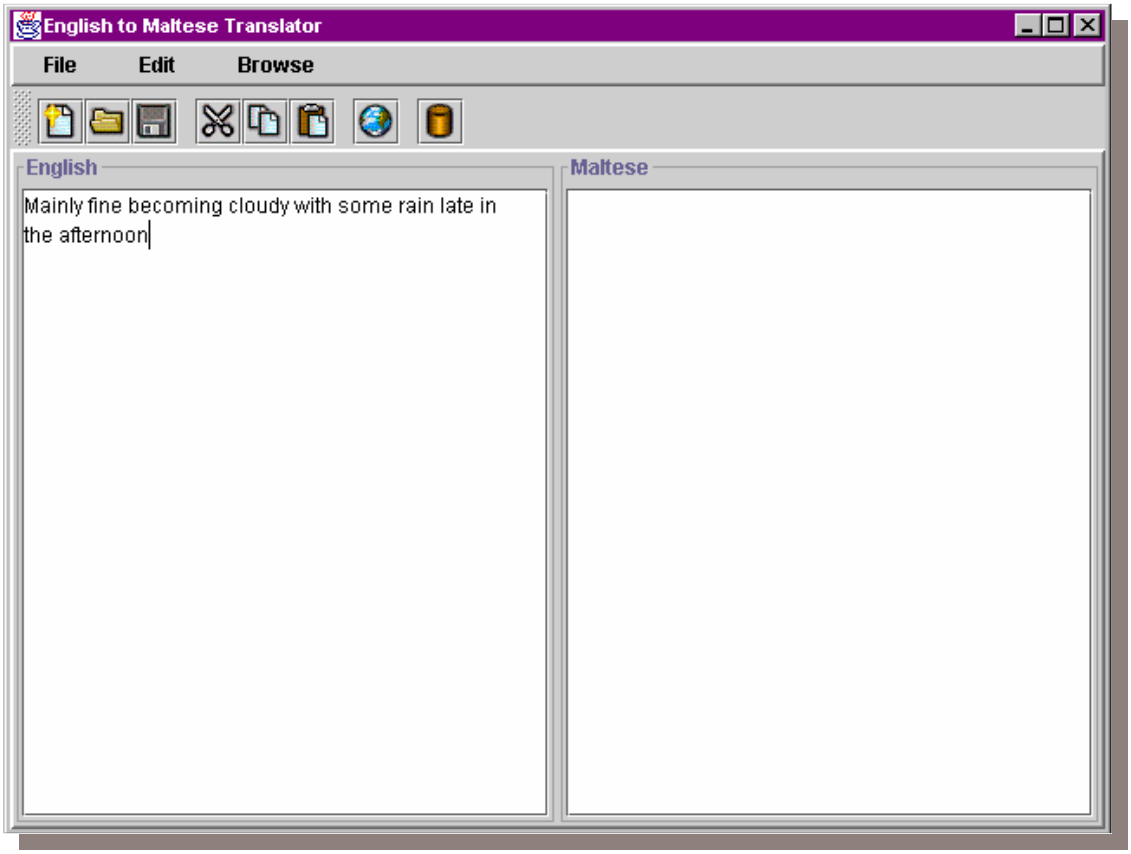


Figure 15.2 Main Program Window

The left text pane is used to edit the English source text. The right text pane is not editable, and is used to display the results of the translation. A menu and a detachable menu bar are available. The facilities provided by the system are described in Table 15.1.

Icon	Menu Option	Operation
	**File | New**	Clears both text panes, allowing the inputting of a new English source phrase.
	**File | Open**	Displays a dialog for loading the contents of the English text pane from a text file
	**File | Save**	Displays a dialog for saving the current contents of the English text pane to a text file
	**Edit | Cut**	Cuts the selected text in the English text pane and places it on the clipboard.
	**Edit | Copy**	Copies the selected text in the English text pane to the clipboard
	**Edit | Paste**	Pastes any text contained in the clipboard to the English text pane
	**Edit | Translate**	Translates the current contents of the English text pane, displaying the results in the Maltese text pane.
	**Browse | Lexicon**	Displays a dialog for browsing through the contents of the system lexicon.

Table 15.1 Program Facilities

Upon initiating the translation, a progress bar is displayed. This gives the translation progress both in terms of the number of words[36] being parsed, and as a percentage of the number of words, as shown in Figure 15.3.

[36] Actually, this is the number of tokens, which can be less than the number of words, due to multiword tokens specified in the lexicon.

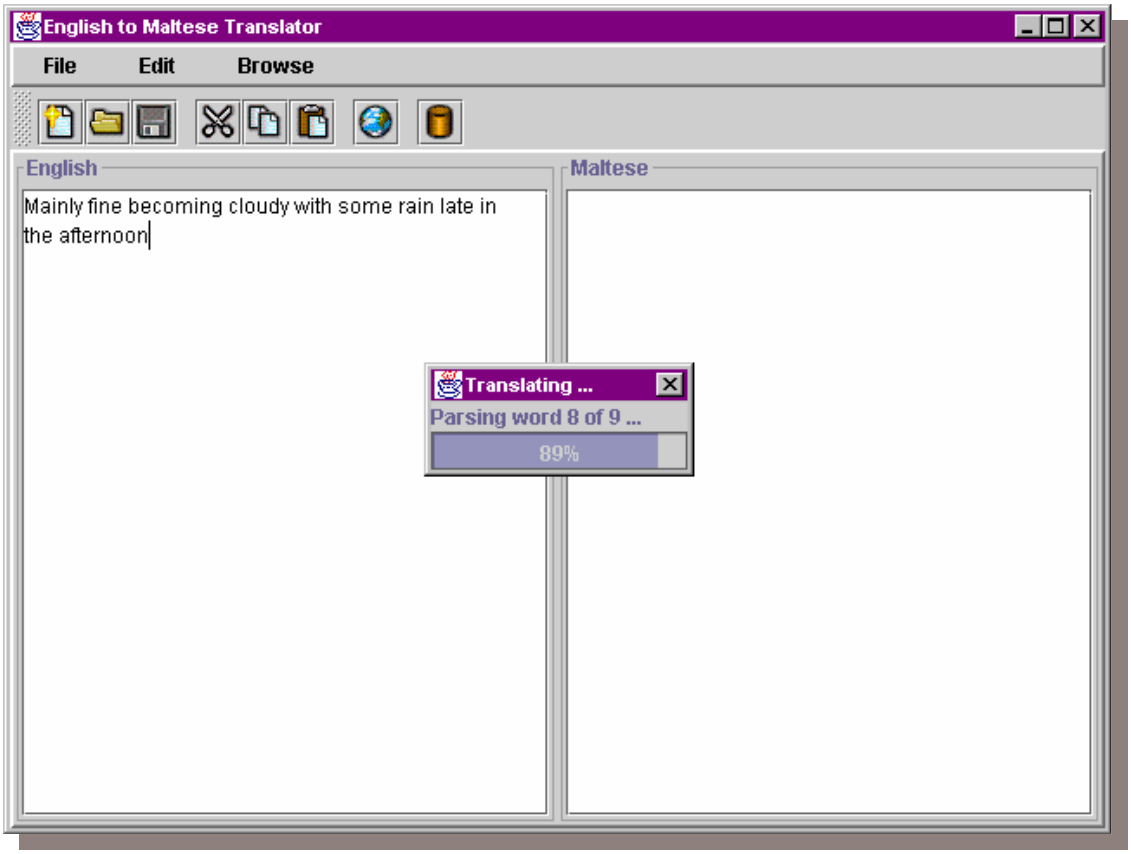


Figure 15.3 Displaying Parsing Progress

When the translation process terminates, the results are displayed in the Maltese text pane, as shown in Figure 15.4. As can be noted, the results may be slightly unreadable due to the use of the Maltilex character convention described in section 7.2.

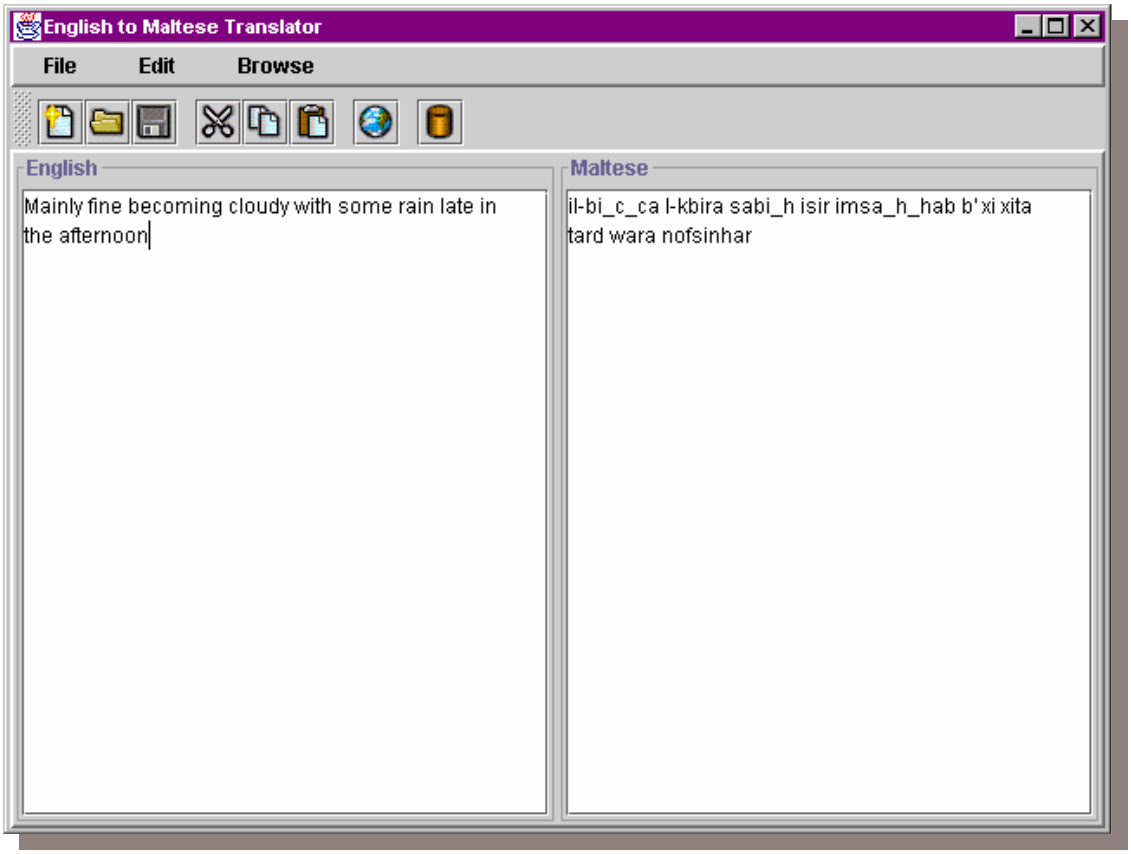


Figure 15.4 Displaying Parse Results

It is also possible to browse the contents of the lexicon. Upon selecting the appropriate option, the modal dialog shown in Figure 15.5 is displayed. By default, information is displayed on the first entry in the lexicon. By selecting the arrow buttons, it is possible to browse the previous and next entry, in the order given in the lexicon.

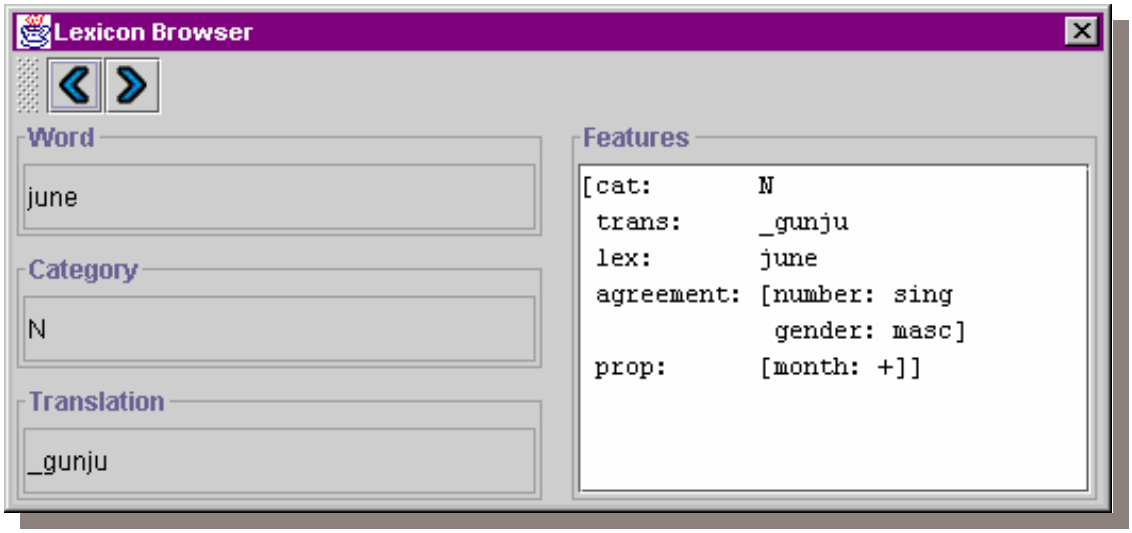


Figure 15.5 Browsing the Lexicon

16 Comments and Conclusions

This project set out to develop an automatic MT system for translating between English and Maltese. For this purpose, a wide range of literature was surveyed. In particular, the various problems a translator, whether human or machine, has to face were investigated in this report. This was followed by a discussion on the motivations for automating the translation process by machine, as well as the issues this raises.

Several designs for the architecture of an MT system were described, including the interlingual, transfer and transformer approaches. The linguistic representation and processing in these systems was next illustrated, while a concluding section specified how MT systems are evaluated.

The implementation proper started by the creation of a corpus on which to base the translating system. This restriction was based upon the fact, corroborated in detail, that general MT systems are extremely difficult to create, while a restricted domain system offers better chances for success. A chart parsing subsystem was next created in order to analyse the input text, and this was augmented with feature structures to increase its representational capabilities.

A formalism for describing target structure representation was next defined, and features were used as a means for expressing transfer rules. A generation component performing on the resulting structures was next created, and the whole translation system was finalised with a transfer lexicon and grammar based on the corpus available.

Finally, some other enhancements, such as the inclusion of syntactic information on the TL in the generation component, fail-safe procedures and the creation of a visual interface, were added to the system. The system was then

evaluated under different aspects. The report ended with a description of the source code structure, as well as of the utilisation of the implemented program.

Looking back, one can say that the field of MT has a relatively long history, but one that did not have major breakthroughs or many successes. Unfortunately, this project does not really change the state of affairs. All the project objectives were achieved, and the field concerned was extensively covered. However, the relative naivety of the author regarding the subject, coupled with a very limited amount of time, not amounting to much more than half a person-year, could only lead to the development of what is really a toy system. This has to be put into perspective by considering that development of an MT system is, in general, a huge undertaking, with many specialists in linguistics and computer science working together over a long period of time. Even the successful but highly restricted METEO system, which partly inspired this project, took eight person-years to be developed by a group of dedicated researchers[37].

A lot of effort was put into developing a sophisticated parser capable of deep semantic representations. However, the machinery created was not utilised to full effect, in that the representations extracted are rather shallow. Time limits did not allow the development of sophisticated transfer and generation components, either. Consequently, the system has too much word for word emphasis.

However, the system cannot be classified of the simple transformer type, since it is actually capable of extracting the type of linguistic knowledge required for LK architectures. As in the latter, error recovery was added to main system rather than made an integral part of the translation engine. It must also be mentioned that the proper development of deep meaning representations, and corresponding transfer rules, require linguistic knowledge and skills that the author does not possess.

[37] Figure taken from “Machine Translation at the TAUM Group”, [13], by Pierre Isabelle.

Despite this, reasonable results were obtained as far as quality is concerned. This was helped in a large way by the highly regular format of the domain, practically void of semantic ambiguity given the context, as well as the low threshold that Maltese has as far as linguistic constructions are concerned.

This report is hereby concluded with some considerations as to how the system could be improved and used to handle broader domains. This issue was in fact addressed up to the point that a separate text archive and lexicon were built, based upon tax-form filling instructions, which were also available in bilingual format, but provided a broader linguistic domain. However, it was not deemed feasible to make major changes to the system in the limited time remaining, so this approach had to be abandoned.

In view of this, a number of minor improvements for the system had already been envisaged. The first involved the handling of numbers, such as 123, 4.5 and so on. These present a different problem from parsing words, since the former cannot be inserted into the lexicon, there being an infinite number of them. An input pre-processor, similar in concept with that for handling unknown words, could be used to identify numbers and tag the input correspondingly.

The second enhancement was meant to be the handling of punctuation, such as parenthesis, semicolons and periods. These can help in identifying the syntactic structure of the input, but may also require a more complex tokeniser since punctuation can itself be ambiguous. For instance, a period can be used both to mark the end of a sentence and to specify an abbreviation.

However, the major issues in the extension to a larger domain are those involving semantic representation and transfer from source to target language. It is thought that the analysis facilities provided by the system are quite sound, and could be used without major modification. However, the definition of word for word translation in the lexicon and transfer rules in the system would be removed, and replaced by a separate bilingual dictionary and a separate TL grammar and

monolingual dictionary. Perhaps the same transfer formalism based on features could be used as well, but certainly, it would have to operate upon higher level semantic specifications than simple syntactic trees. In addition, it would be necessary to establish a deeper semantic correlation in the transfer rules, rather than just assume that transfer can be achieved by tree manipulation. This in turn would require deeper knowledge of the structural relationships between the languages concerned.

Finally, of course, there would be the need of increasing the size of the dictionaries and grammars to take into account the vocabulary of the broader domain. This might involve adding morphological processing capabilities to the system, whose complexity may become feasible when handling large system dictionaries. This, of course, is not a simple matter of introducing new entries, but requires careful and controlled editing, based on sound linguistic knowledge of both the source and the target language.

Bibliography

[1] Allen J. 1995. *Natural Language Understanding*, (2nd Edition). The Benjamin/Cummings Publishing Company, Inc.

[2] Arnold D., Balkan L., Meijer S., Humphreys R.L., Sadler L., 1994. *Machine Translation: An Introductory Guide*. Blackwells-NCC, London.

[3] British Computer Society, Natural Language Translation Specialist Group. 1997. *Machine Translation Review*, Journal of the BCS NLT Specialist Group, October 1997. See http://www.bcs.org.uk/siggroup/sg37.htm.

[4] Burton-Roberts, N. 1986. *Analysing Sentences: An Introduction to English Syntax*. Longman Group Limited.

[5] Busuttil V. 1900. *Diziunariu mill-Ingliz ghall-Malti*. Stamperia ta' Malta, 16, Strada Zecca. Malta.

[6] Cole, R. A. (Editor in Chief). 1996. *Survey of the State of the Art in Human Language Technology*. Refer to http://cslu.cse.ogi.edu/HLTsurvey/.

[7] Cremona A. 1962. *Tagħlim fuq il-Kitba Maltija*, (7th Edition). LUX Press, Malta.

[8] Gazdar G., Mellish C. 1989. "Well-Formed Substring Tables and Charts", *Natural Language Processing in Prolog*. Addison Wesley.

[9] *Java* Programming Language Homepage. For further details, refer to http://www.java.sun.com/.

[10] *JavaCC*, The Java Parser Generator Homepage. For further details, refer to http://www.suntest.com/JavaCC/.

[11] Johnson R, 1985. *Notes on Tabular Parsing*. UMIST Manchester. Adapted by Mike Rosner 1998.

[12] Karttunnen L., Chanod J-P., Grefenstette G., Schiller A. 1997. *Regular Expressions for Language Engineering*. Cambridge University Press.

[13] King M (Editor). 1987. *Machine Translation Today: The State of the Art*. Proceedings of the Third Lugano Tutorial, Lugano, Switzerland, 2-7 April, 1984. Edinburgh University Press.

[14] Landsbergen J. 1987. "Montague Grammar and Machine Translation", *Linguistic Theory and Computer Applications*. Academic Press Limited.

[15] McConnel, S. 1997. *PC-PATR Reference Manual*, version 0.99b5. PC-PATR is an implementation of the PATR-II computational linguistic formalism for personal computers. For further details, refer to http://www.sil.org/pcpatr/.

[16] Miller, G. A. (Principal Investigator). *WordNet*, A Lexical Database for English. For further details refer to http://www.cogsci.princeton.edu/~wn/.

[17] Pereira, Fernando C. N., Shieber, S. M. *The Semantics of Grammar Formalisms seen as Computer Languages*. Artificial Intelligence Centre, SRI International and Centre for the Study of Language and Information, Stanford University.

[18] Power Translator Deluxe, English/German Version. Power Translator is an MT system developed by Globalink Inc. For further details, refer to http://www.globalink.com/.

[19] Reiter E. 1994. *NLG vs. Templates*. CoGenTex, Inc 840 Hanshaw Rd Ithaca, NY 14850 USA.

[20] Rosner M., Caruana J., Fabri R., Galea D., Vella O. *Maltilex*. A Computational Lexicon for the Maltese Language. For further details refer to http://www.cs.um.edu.mt/~mros/maltilex/.

[21] Russel S., Norvig P. 1995. "Practical Natural Language Processing", *Artificial Intelligence, A Modern Approach*. Prentice-Hall International.

[22] Shieber, S. M. 1986. *An Introduction to Unification-Based Approaches to Grammar*. CLSI Lecture Notes 4. Chicago: Chicago U Press.

[23] Slocum J. January-March 1985. “A Survey of Machine Translation, its History, Current Status and Future Prospects”, *Computational Linguistics*, Volume 11, Number 1.

[24] SYSTRAN Translation Software Homepage. For further details, refer to http://www.systransoft.com/.

[25] Verbmobil Homepage. For further details, refer to http://www.dfki.uni-sb.de/verbmobil/.

[26] Vogel, S. 1998. “Lost In Translation”, *PC Plus Issue 136, February 1998*. Future Publishing, Bath, UK.

Appendix A Text Archive Samples

A.1 Sample English Entries

Date	Source	General Statement	Weather Forecast/ Tbassir tat-Temp	Visibility/ Vi_zibilta	Wind/ Ri_h	Sea/ Ba_har	Swell/ Imbatt
01-Jun-98	Times	An anticyclone covers the area	Sunny	Good	Light and variable (Force 3 over sea areas)	Slight	Little
02-Jun-98	Times	High pressure persists over the central Mediterranean	Fine and sunny	Good	Light and variable with onshore sea breezes (Force 3 over sea areas)	Slight	Negligible
03-Jun-98	Times	An anticyclone persists over the Ionian Sea	Sunny and warm	Good	Light to moderate southsoutheasterly (Force 3 to 4 over sea areas)	Slight to moderate	Low southeasterly
04-Jun-98	Times	An anticyclone persists over the Ionian Sea	Sunny and warm	Good	Light to moderate southeasterly (Force 3 to 4 over sea areas)	Slight to moderate	Low southeasterly
05-Jun-98	Times	Anticyclone persists over the Ionian Sea	Fine and warm	Good	Light to moderate southeasterly (Force 3 to 4 over sea areas)	Slight becoming slight to moderate	Low southeast
06-Jun-98	Times	A ridge of high pressure from the Balkans to the central Mediterranean is almost stationary	Mainly sunny and warm	Good	Moderate southeast (Force 4 over sea areas)	Moderate	Low southeast
07-Jun-98	Times	Low will persist over Tunisia	Sunny periods becoming rather cloudy by evening	Good	Moderate occasionally rather strong southeast (Force 4 to 5 over sea areas)	Moderate	Low to moderate southesterly
08-Jun-98	Times	An anticyclone persists over the Ionian Sea	Rather cloudy with possibility of some rain in places	Good	Light southeast becoming moderate northwest (Force 3 becoming force 4 over sea areas)	Slight becoming moderate	Low southeasterly becoming low northwesterly later

Date	Source	General Statement	Weather Forecast/ Tbassir tat-Temp	Visibility/ Vi_zibilta	Wind/ Ri_h	Sea/ Ba_har	Swell/ Imbatt
09-Jun-98	Times	A ridge of high pressure extends from the Balearics to Egypt	Partly cloudy becoming rather cloudy at times	Good	Moderate northwest (Force 4 over sea areas)	Moderate	Low northwest
10-Jun-98	Times	Ridge of high pressure extends from the mid-Atlantic across northern Algeria to Tunisia	Fine or partly cloudy	Good	Northwest moderate occasionally fresh (Force 4 to 5 over sea areas)	Moderate	Low northwest
11-Jun-98	Times	A ridge of high pressure covers the western Mediterranean	Sunny periods	Good	Moderate westerly (Force 4 over sea areas)	Moderate	Low westerly
12-Jun-98	Times	A ridge of high pressure extends from Spain to Tunisia	Partly cloudy	Good	Rather strong NW'W (Force 5 to 6 over the sea)	Rough	Low to moderate NW
13-Jun-98	Times	A ridge of high pressure extends from the Azores to the Eastern Mediterranean	Sunny periods	Good	Strong northwest decreasing slowly (Force 6 to 7 becoming force 5 over sea areas)	Very rough becoming rough	Moderate to heavy northwest
14-Jun-98	Times	Ridge of high pressure covers the central Mediterranean	Sunny periods	Good	Moderate northwest (Force 4 over sea areas)	Moderate	Low northwest
15-Jun-98	Times	Anticyclone near Crete is slow moving	Mainly sunny	Good	Southerly moderate becoming southeast later (Force 4 over sea areas)	Moderate	Low southerly

A.2 Sample Maltese Entries

Date	Source	General Statement	Weather Forecast/ Tbassir tat-Temp	Visibility/ Vi_zibilta	Wind/ Ri_h	Sea/ Ba_har	Swell/ Imbatt
01-Jun-98	Nazzjon		Xemxi	Tajba	_Hafif u varjabbli (Forza 3 fuq l-ib_hra)	_Hafif	Ftit li xejn
02-Jun-98	Nazzjon		Sabi_h u xemxi	Tajba	_Hafif u varjabbli b'xi _iffa minn fuq il-ba_har (Forza 3 fuq l-ib_hra)	_Hafif	Ftit li xejn
03-Jun-98	Nazzjon		Xemxi u s_hun	Tajba	_Hafif _yal moderat min-Nofsin_har ix-Xlokk (Forza 3 _yal 4 fuq l-ib_hra)	_Hafif _yal moderat	Baxx mix-Xlokk
04-Jun-98	Nazzjon		Xemxi u s_hun	Tajba	_Hafif _yal moderat mix-Xlokk (Forza 3 _yal 4 fuq l-ib_hra)	_Hafif _yal moderat	Baxx mix-Xlokk

Date	Source	General Statement	Weather Forecast/ Tbassir tat-Temp	Visibility/ Vi_zibilta	Wind/ Ri_h	Sea/ Ba_har	Swell/ Imbatt
05-Jun-98	Nazzjon		Bnazzi u s_hun	Tajba	Xlokk _hafif _yal moderat (Forza 3 _yal 4 fuq il-ba_har)	_Hafif li jsir _hafif _yal moderat	Baxx mix-Xokk
06-Jun-98	Nazzjon		Ikun il-bi_c_ca l-kbira xemxi u s_hun	Tajba	Moderat mix-Xlokk (Forza 4 fuq il-ba_har)	Moderat	Baxx mix-Xlokk
07-Jun-98	Mument		Ikun b'waqt_it xemxin li jissa_h_hab _yal fil-_yaxija	Tajba	Moderat kultant ftit qawwi mix-Xlokk (Forza 4 fuq il-ba_har)	Moderat	Baxx _yal moderat mix-Xlokk
08-Jun-98	Nazzjon		Ikun xi ftit imsa_h_hab u jista' jkun _hemm xi xita f'xi n_hawi	Tajba	_Hafif mix-Xlokk li jsir moderat mill-Majjistral (Forza 3 li jsir forza 4 fuq il-ba_har)	_Hafif li jsir moderat	Baxx mix-Xlokk li jsir baxx mil-Lbi_c
09-Jun-98	Nazzjon		Jissa_h_hab f'xi _hinij_it	Tajba	Moderat mill-Majjistral (Forza 4 fuq il-ba_har)	Moderat	Baxx mill-Majjistral
10-Jun-98	Nazzjon		Sabi_h jew ftit imsa_h_hab	Tajba	Moderat mill-Majjistral kultant qawwi (Forza 4 _yal 5 fuq il-ba_har)	Moderat	Baxx mill-Majjistral
11-Jun-98	Nazzjon		Ikun b'waqt_it xemxin	Tajba	Moderat mill-Punent (Forza 4 fuq il-ba_har)	Moderat	Baxx mill-Punent
12-Jun-98	Nazzjon		Ikun ftit imsa_h_hab	Tajba	Ftit qawwi mill-Majjistral (Forza 5 _yal 6 fuq il-ba_har)	Qawwi	Baxx _yal moderat mill-Majjistral
13-Jun-98	Nazzjon		Ikun b'waqt_it xemxin	Tajba	Ftit Majjistral qawwi li jonqos bil-mod (Forza 6 _yal 7 fuq il-ba_har)	Qawwi	Qawwi _yal moderat mill-Majjistral
14-Jun-98	Mument		Ikun b'waqt_it xemxin	Tajba	Moderat mill-Majjistral (Forza 4 fuq il-ba_har)	Moderat	Baxx mill-Majjistral
15-Jun-98	Nazzjon		Il-bi_c_ca l-kbira xemxi	Tajba	Moderat min-nofsin_har isir mix-Xlokk aktar tard (Forza 4 fuq il-ba_har)	Moderat	Baxx min-Nofsin_har

Appendix B Lexicon and Grammar BNF

B.1 Lexicon BNF

Lexicon ::= LexiconTemplates LexiconEntries <*EOF*>

LexiconTemplates ::= (LexiconTemplate)*

LexiconTemplate ::= <*LET*> TemplateName <*BE*>
(TemplateName | FeatureEquation)+ <*DOT*>

FeatureEquation ::= FeaturePath <*EQUALS*> (ConstantValue | FeaturePath)

FeaturePath ::= <*LT*> (String)+ <*GT*>

TemplateName ::= String

ConstantValue ::= String

LexiconEntries ::= (LexiconEntry)+

LexiconEntry ::= WordSpec CategorySpec (TranslationSpec)?
(FeaturesSpec)?

WordSpec ::= <*WORD_DELIMITER*> Word

Word ::= (String)+

CategorySpec ::= <*CATEGORY_DELIMITER*> Category

Category ::= String

TranslationSpec ::= <*TRANSLATION_DELIMITER*> Word

FeaturesSpec ::= <*FEATURE_DELIMITER*>
(TemplateName | FeatureEquation)+

String ::= (<*STRING*> | <*LET*> | <*BE*>)

B.2 Grammar BNF

Grammar ::= Parameters Rules <*EOF*>

Parameters ::= ParameterStartSymbol

ParameterStartSymbol ::= <*PARAMETER*> <*COLON*> <*START*> <*SYMBOL*> <*IS*> SimpleCategory <*DOT*>

Rules ::= (Rule)+

Rule ::= <*RULE*> RuleLHS <*ARROW*> RuleRHS <*COLON*> FeatureEquations <*DOT*>

RuleLHS ::= Category

RuleRHS ::= (Category)+

Category ::= (SimpleCategory | IndexedCategory)

SimpleCategory ::= <*STRING*>

IndexedCategory ::= <*INDEXED_STRING*>

FeatureEquations ::= (FeatureEquation)*

FeatureEquation ::= FeaturePath <*EQUALS*> (ConstantValue | FeaturePath)

FeaturePath ::= <*LT*> Category (<*STRING*>)* <GT>

ConstantValue ::= <*STRING*>

Appendix C Bilingual Lexicon Samples

```
;*************************************
; Templates
;*************************************

Let Sing be <agreement number> = sing.

Let Plur be <agreement number> = plur.

Let Masc be <agreement gender> = masc.

Let Fem be <agreement gender> = fem.

Let Time be <prop time>   = +.

Let Phen be <prop phen>   = +.

Let Dir be <prop dir>     = +.

Let Month be <prop month> = +.

Let Place be <prop place> = +.

Let Ing be <vform> = ing.

Let Pres be <vform> = pres.
```

```
;****************************************
; Nouns
;****************************************

\w afternoon
\c N
\t wara nofsinhar
\f Time Sing Masc

\w air
\c N
\t arja
\f Phen Sing Fem

\w airflow
\c N
\t kurrent t'arja
\f Phen Sing Fem

\w anticyclone
\c N
\t anti_ciklun
\f Phen Sing Masc

\w anticyclones
\c N
\t anti_cikluni
\f Phen Plur

\w outbreaks of rain
\c NP
\t xita minn _hin _yall-_i_hor
\f Phen Plur
```

```
\w period
\c N
\t perijodu
\f Time Sing Masc

\w periods
\c N
\t waqt_it
\f Time Plur

\w weather
\c N
\t temp
\f Phen Sing Masc

;****************************************
; Direction nouns
;****************************************

\w north
\c N
\t tramuntana
\f Dir Sing Fem

\w northnortheast
\c N
\t gr_ig it-tramuntana
\f Dir Sing Masc

\w northeast
\c N
\t grigal
\f Dir Sing Masc
```

```
\w eastnortheast
\c N
\t gr_ig il-Lvant
\f Dir Sing Masc

;****************************************
; Month Names
;****************************************

\w january
\c N
\t jannar
\f Month Sing Masc

\w february
\c N
\t frar
\f Month Sing Masc

\w march
\c N
\t marzu
\f Sing Masc

;****************************************
; Places
;****************************************

\w Alps
\c N
\t Alpi
\f Place Plur

\w Balearics
\c N
\t Baleari_ci
\f Place Plur
```

```
\w Maltese Islands
\c N
\t g_zejjer Maltin
\f Place Plur

\w Europe
\c N
\t Ewropa
\f Place Sing Fem

\w France
\c N
\t Franza
\f Place Sing Fem

\w Malta
\c N
\t Malta
\f Place Sing Fem

;***************************************
; Verbs
;***************************************

\w covering
\c V
\t j_yatti
\f Ing Sing Masc

\w covering
\c V
\t t_yatti
\f Ing Sing Fem
```

```
\w covering
\c V
\t j_yattu
\f Ing Plur

\w cover
\c V
\t j_yattu
\f Plur

\w covers
\c V
\t j_yatti
\f Sing Masc

\w covers
\c V
\t t_yatti
\f Sing Fem

;***************************************
; Quantitative Adjectives
;***************************************

\w about
\c QADJ
\t kwa_zi

\w little
\c QADJ
\t ftit
```

```
\w some
\c QADJ
\t xi

;*************************************
; Adjectives
;*************************************

\w average
\c ADJ
\t fil-medja

\w cloudy
\c ADJ
\t imsa_h_hab
\f Sing Masc

\w cloudy
\c ADJ
\t imsa_h_hba
\f Sing Fem

\w cloudy
\c ADJ
\t imsa_h_hbin
\f Plur

\w sunny
\c ADJ
\t xemxi
\f Sing Masc

\w sunny
\c ADJ
\t xemxija
\f Sing Fem
```

```
\w sunny
\c ADJ
\t xemxin
\f Plur

\w thundery
\c ADJ
\t bir-ra_yad

;***************************************
; Adverbs
;***************************************

\w above
\c ADV
\t fuq

\w at times
\c PP
\t xi kultant

\w at first
\c PP
\t _yall-ewwel

\w generally
\c ADV
\t _generalment
```

```
;****************************************
; Direction adverbs
;****************************************

\w northerly
\c ADV
\t mit-tramuntana
\f Dir

\w northeasterly
\c ADV
\t mill-grigal
\f Dir

;****************************************
; Prepositions
;****************************************

\w in the morning
\c PP
\t fil-_yodu

\w in the afternoon
\c PP
\t wara nofsinhar

\w in the evening
\c PP
\t fil-_yaxija

\w across
\c PREP
\t fuq

\w by
\c PREP
\t sa
```

```
\w by evening
\c PP
\t sa fil-_yaxija


\w in
\c PREP
\t f'

\w with
\c PREP
\t b'

;****************************************
; Articles
;****************************************

\w a
\c ART
\f Sing

\w an
\c ART
\f Sing

\w the
\c ART
\t il-
```

```
;*************************************
; Auxiliaries
;*************************************

\w may
\c AUX
\t jista
\f Sing Masc

\w may
\c AUX
\t tista
\f Sing Fem

\w may
\c AUX
\t jist_yu
\f Plur

\w will
\c AUX
\t se

;*************************************
; Conjunctions
;*************************************

\w and
\c CONJ
\t u

\w but
\c CONJ
\t imma
```

```
\w or
\c CONJ
\t jew

;*************************************
; Pronouns
;*************************************

\w none
\c PRON
\t xejn

\w which
\c PRON
\t l_ima
```

Appendix D Transfer Grammar

```
;*************************************
; Parameters
;*************************************

Parameter: Start symbol is S.

;*************************************
; Top-Level Phrases
;*************************************

Rule S → NP:
 <NP agreement gender> = masc
 <NP agreement number> = sing
 <S  trans>           = <NP trans>.

Rule S → VP:
 <VP agreement gender> = masc
 <VP agreement number> = sing
 <S  trans>           = <VP trans>.

Rule S → ADJP:
 <ADJP agreement gender> = masc
 <ADJP agreement number> = sing
 <S    trans>           = <ADJP trans>.

Rule S → PP:
 <S trans> = <PP trans>.
```

```
Rule S → PREP:
 <S trans> = <PREP trans>.

Rule S → ART:
 <S trans> = <ART trans>.

;*************************************
; Noun Phrases
;*************************************

Rule NP → N:
 <NP agreement> = <N agreement>
 <NP trans>     = <N trans>.

Rule NP → NP_1 PP:
 <NP agreement> = <NP_1 agreement>
 <NP trans 1>   = <NP_1 trans>
 <NP trans 2>   = <PP   trans>.

Rule NP → ADJP NP_1:
 <ADJP agreement> = <NP_1 agreement>
 <NP   agreement> = <NP_1 agreement>
 <NP   trans 1>   = <NP_1 trans>
 <NP   trans 2>   = <ADJP trans>.

Rule NP → QADJ NP_1:
 <QADJ agreement> = <NP_1 agreement>
 <NP   agreement> = <NP_1 agreement>
 <NP   trans 1>   = <QADJ trans>
 <NP   trans 2>   = <NP_1 trans>.
```

```
Rule NP → NP_1 CONJ NP_2:
 <NP   agreement number> = plur
 <CONJ lex>              = and
 <NP   trans 1>          = <NP_1 trans>
 <NP   trans 2>          = <CONJ trans>
 <NP   trans 3>          = <NP_2 trans>.

Rule NP → NP_1 ADV:
 <NP agreement> = <NP_1 agreement>
 <NP trans 1>   = <NP_1 trans>
 <NP trans 2>   = <ADV  trans>.

Rule NP → ART NP_1:
 <NP  agreement> = <NP_1 agreement>
 <ART agreement> = <NP_1 agreement>
 <NP  trans 1>   = <ART  trans>
 <NP  trans 2>   = <NP_1 trans>.

Rule NP → NP_1 V ADJP:
 <NP   agreement> = <NP_1 agreement>
 <NP_1 agreement> = <V    agreement>
 <NP_1 agreement> = <ADJP agreement>
 <NP   trans 1>   = <NP_1 trans>
 <NP   trans 2>   = <V    trans>
 <NP   trans 3>   = <ADJP trans>.

Rule NP → NP_1 CONJ ADJP:
 <NP   agreement> = <NP_1 agreement>
 <NP_1 agreement> = <ADJP agreement>
 <NP   trans 1>   = <NP_1 trans>
 <NP   trans 2>   = <CONJ trans>
 <NP   trans 3>   = <ADJP trans>.
```

```
;*************************************
; Verb Phrases
;*************************************

Rule VP → V:
 <VP agreement> = <V agreement>
 <VP trans>     = <V trans>.

;*************************************
; Adjective Phrases
;*************************************

Rule ADJP → ADJ:
 <ADJP agreement> = <ADJ agreement>
 <ADJP trans>     = <ADJ trans>.

Rule ADJP → ADV ADJ:
 <ADJP agreement> = <ADJ agreement>
 <ADJP trans 1>   = <ADV trans>
 <ADJP trans 2>   = <ADJ trans>.

Rule ADJP → ADJP_1 V ADJP_2 ADV:
 <ADJP_1 agreement> = <V      agreement>
 <ADJP_1 agreement> = <ADJP_2 agreement>
 <ADJP   agreement> = <ADJP_1 agreement>
 <ADJP   trans 1>   = <ADJP_1 trans>
 <ADJP   trans 2>   = <V      trans>
 <ADJP   trans 3>   = <ADJP_2 trans>
 <ADJP   trans 4>   = <ADV    trans>.

Rule ADJP → ADJP_1 CONJ ADJP_2:
 <ADJP_1 agreement> = <ADJP_2 agreement>
 <ADJP   agreement> = <ADJP_1 agreement>
 <ADJP   trans 1>   = <ADJP_1 trans>
 <ADJP   trans 2>   = <CONJ   trans>
 <ADJP   trans 3>   = <ADJP_2 trans>.
```

```
Rule ADJP → ADJP_1 PP:
 <ADJP agreement> = <ADJP_1 agreement>
 <ADJP trans 1>   = <ADJP_1 trans>
 <ADJP trans 2>   = <PP     trans>.

Rule ADJP → ADJP_1 V ADJP_2:
 <ADJP_1 agreement> = <V      agreement>
 <ADJP_1 agreement> = <ADJP_2 agreement>
 <ADJP   agreement> = <ADJP_1 agreement>
 <ADJP   trans 1>   = <ADJP_1 trans>
 <ADJP   trans 2>   = <V      trans>
 <ADJP   trans 3>   = <ADJP_2 trans>.

;*************************************
; Prepositional Phrases
;*************************************

Rule PP → PREP NP:
 <PP trans 1> = <PREP trans>
 <PP trans 2> = <NP   trans>.

Rule PP → CONJ PP_1:
 <PP trans 1>   = <CONJ trans>
 <PP trans 2>   = <PP_1 trans>.
```

```
;*************************************
; Idioms – Handling translation of
; 'early <month>' as a special case
;*************************************

Rule NP → ADJ N:
 <ADJ lex>        = early
 <N   prop month> = +
 <ADJ agreement>  = <N agreement>
 <NP  agreement>  = <N agreement>
 <NP  trans 1>    = bidu
 <NP  trans 2>    = ta'
 <NP  trans 3>    = <N trans>.
```

Appendix E Parse Samples

The two top-level parses obtained for the noun phrase '*sunny periods becoming cloudy*' are shown below. The notation used is

$$[\textit{start vertex}, \textit{end vertex}, \textit{dotted rule}, \textit{feature structure}]$$

```
[0, 4, S → NP •,

[cat:   S
 trans: $9
 NP:    [cat:       NP
         agreement: [number: $3
                     gender: $2 ]
         trans:     $9 [1: $7 [1: $8 perijodu
                               2: $5 xemxi ]
                           2: $4 isir
                           3: $1 imsa_h_hab ]
         NP1:       [NP1:       [cat:       NP
                                 trans:     $8
                                 agreement: [number: $3 sing
                                             gender: $2 masc ]
                                 N:         [lex:       period
                                             cat:       N
                                             agreement: [number: $3
                                                         gender: $2 ]
                                             prop:      [time:  +]
                                             trans:     $8]
                     cat:       NP
                     agreement: $6 [number: $3
                                    gender: $2 ]
                     trans:     $7
                     ADJP:      [cat:       ADJP
```

```
                                agreement: $6
                                trans:     $5
                                ADJ:       [cat:       ADJ
                                            lex:       sunny
                                            agreement: $6
                                            trans:     $5 ]]]
        V:          [lex:       becoming
                     cat:       V
                     vform:     ing
                     agreement: [number: $3
                                 gender: $2 ]
                     trans:     $4]
        ADJP:       [cat:       ADJP
                     agreement: [number: $3
                                 gender: $2 ]
                     trans:     $1
                     ADJ:       [cat:       ADJ
                                 lex:       cloudy
                                 trans:     $1
                                 agreement: [number: $3
                                             gender: $2 ]]]]]
]
```

[*0*, *4*, S → NP •,

```
[cat:   S
 trans: $1
 NP:    [NP1:       [trans:     $10 [3: $6 imsa_h_hab
                                     2: $7 isir
                                     1: $8 perijodu]
                     cat:        NP
                     NP1:        [cat:       NP
                                  trans:     $8
                                  agreement: $9 [number: $3 sing
                                                 gender: $2 masc]
                                  N:         [cat:       N
                                              agreement: $9
                                              trans:     $8
                                              lex:       period
                                              prop:      [time:  +]]
                     agreement:  [number: $3
                                  gender: $2 ]
                     V:          [vform:     ing
                                  lex:       becoming
                                  trans:     $7
                                  cat:       V
                                  agreement: [number: $3
                                              gender: $2 ]]
                     ADJP:       [cat:       ADJP
                                  ADJ:       [cat:       ADJ
                                              agreement: [number: $3
                                                          gender: $2 ]
                                              trans:     $6
                                              lex:       cloudy]
                                  trans:     $6
                                  agreement: [number: $3
                                              gender: $2 ]]]
          cat:       NP
          ADJP:      [cat:       ADJP
                      ADJ:       [cat:       ADJ
```

```
                                agreement: $5 [number: $3
                                               gender: $2 ]
                                trans:     $4 xemxi
                                lex:       sunny]
                    trans:     $4
                    agreement: $5 ]
          trans:     $1 [2: $4
                         1: $10 ]
          agreement: [number: $3
                      gender: $2 ]]]
]
```

Appendix F Evaluation Form Sample

English Source	Maltese Translation	Excellent	Good	Fair	Bad
Cloudy with rain becoming partly cloudy later	imsaħħab b' xita isir xi ftit imsaħħab aktar tard				
	imsaħħab b' xita issir xi ftit imsaħħba aktar tard				
Cloudy with showers	imsaħħab b' ħalbiet tax-xita				
Fine and partly cloudy	sabiħ u xi ftit imsaħħab				
Fine and sunny	sabiħ u xemxi				
Fine and sunny with temperatures above normal	normal sabiħ u xemxi b' temperaturi fuq				
	sabiħ u xemxi b' temperaturi fuq in- normal				
Fine and warm	sabiħ u sħun				
Fine with some cloud at times	sabiħ b' xi sħaba xi kultant				
	sabiħ b' xi sħab xi kultant				
Fine with some high cloud at times	sabiħ b' xi sħaba xi kultant għolja				
	sabiħ b' xi sħaba għolja xi kultant				
	sabiħ b' xi sħab xi kultant għoljin				
	sabiħ b' xi sħab għoljin xi kultant				
Long sunny periods	waqtiet xemxin twal				

English Source	Maltese Translation	Excellent	Good	Fair	Bad
Mainly cloudy	il-biċċa l-kbira imsaħħab				
Mainly cloudy with isolated showers with some sunny or bright intervals	il-biċċa l-kbira imsaħħab b' ħalbiet tax-xita b' xi intervalli xemxin jew sbieħ iżolati				
	il-biċċa l-kbira imsaħħab b' ħalbiet tax-xita iżolati b' xi intervalli xemxin jew sbieħ				
Mainly cloudy with occasional rain becoming brighter later in the afternoon	il-biċċa l-kbira imsaħħab b' xita okkażjonali isir aktar xemxi aktar tard wara nofsinhar				
	il-biċċa l-kbira imsaħħab b' xita okkażjonali issir aktar xemxija aktar tard wara nofsinhar				
	il-biċċa l-kbira imsaħħab b' xita issir aktar xemxija okkażjonali aktar tard wara nofsinhar				
	il-biċċa l-kbira imsaħħab b' xita issir aktar xemxija aktar tard okkażjonali wara nofsinhar				
	il-biċċa l-kbira imsaħħab b' xita issir aktar xemxija aktar tard wara nofsinhar okkażjonali				
Mainly fine becoming cloudy with showers in the afternoon	il-biċċa l-kbira sabiħ isir imsaħħab b' ħalbiet tax-xita wara nofsinhar				
Mainly fine becoming cloudy with some rain late in the afternoon	il-biċċa l-kbira sabiħ isir imsaħħab b' xi xita tard wara nofsinhar				

English Source	Maltese Translation	Excellent	Good	Fair	Bad
Mainly fine becoming rather cloudy at times	il-biċċa l-kbira sabiħ isir aktarx imsaħħab xi kultant				
Mainly fine becoming rather cloudy with the chance of an isolated shower in the afternoon	il-biċċa l-kbira sabiħ isir aktarx imsaħħab biċ- ċans ta' ħalba xita wara nofsinhar iżolata				
	il-biċċa l-kbira sabiħ isir aktarx imsaħħab biċ- ċans ta' ħalba xita iżolata wara nofsinhar				
Mainly sunny	il-biċċa l-kbira xemxi				
Mainly sunny and warm	il-biċċa l-kbira xemxi u sħun				
Mainly sunny and warm becoming partly cloudy at times	il-biċċa l-kbira xemxi u sħun isir xi ftit imsaħħab xi kultant				
Mainly sunny with patchy cloud and temperature near normal	il-biċċa l-kbira xemxi bi sħaba kultant u temperatura qrib in-normal				
	il-biċċa l-kbira xemxi bi sħaba u temperatura qrib in-normal kultant				
	il-biċċa l-kbira xemxi bi sħab kultant u temperatura qrib in-normal				
	il-biċċa l-kbira xemxi bi sħab u temperatura qrib in-normal kultant				
	il-biċċa l-kbira xemxi bi sħaba u temperatura kultant qrib in-normal				
	il-biċċa l-kbira xemxi bi sħab u temperatura kultant qrib in-normal				