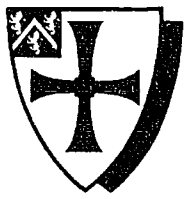
Natural language generation in the LOLITA system an engineering approach

Smith, Mark H.

University of Durham



Natural Language Generation in the LOLITA System: An Engineering Approach.

Mark H. Smith

Laboratory for Natural Language Engineering,
Department of Computer Science.

Submitted in partial fulfilment of the
requirements for the degree of

Doctor of Philosophy

Abstract

Natural Language Generation (NLG) is the automatic generation of Natural Language (NL) by computer in order to meet communicative goals. One aim of NL processing (NLP) is to allow more natural communication with a computer and, since communication is a two-way process, a NL system should be able to produce as well as interpret NL text.

This research concerns the design and implementation of a NLG module for the LOLITA system. LOLITA (Large scale, Object-based, Linguistic Interactor, Translator and Analyser) is a general purpose base NLP system which performs core NLP tasks and upon which prototype NL applications have been built. As part of this encompassing project, this research shares some of its properties and methodological assumptions: the LOLITA generator has been built following Natural Language Engineering principles, uses LOLITA's SemNet representation as input and is implemented in the functional programming language Haskell.

As in other generation systems the adopted solution utilises a two component architecture. However, in order to avoid problems which occur at the interface between traditional planning and realisation modules (known as *the generation gap*) the distribution of tasks between the planner and plan-realiser is different: the plan-realiser, in the absence of detailed planning instructions, must perform some tasks (such as the selection and ordering of content) which are more traditionally performed by a planner. This work largely concerns the development of the plan-realiser and its interface with the planner. Another aspect of the solution is the use of Abstract Transformations which act on the SemNet input before realisation leading to an increased ability for creating paraphrases.

The research has lead to a practical working solution which has greatly increased the power of the LOLITA system. The research also investigates how NLG systems can be evaluated and the advantages and disadvantages of using a functional language for the generation task.

Acknowledgements

I would like to thank my supervisor Roberto Garigliano for his advice and support throughout the four years over which this research has been conducted.

I am also grateful for all past and present members of the Laboratory for Natural Language Engineering who have helped provide such a pleasant research and social environment. Thank you to all those people who commented on the various drafts of this thesis, particularly Derek Long and Maria Fox.

Thanks also to the subscribers to the Leeds United mailing list who took part in the evaluation experiment.

Finally I would like to thank Rebecca and my family and apologise for my bad moods when things were not going as planned!

Declaration

The material contained within this thesis has not previously been submitted for a degree at the University of Durham or any other university. The research reported within this thesis has been conducted by the author unless indicated otherwise.

Contents

1 Methodological Introduction **1**

1.1 Traditional NLP Approaches 1

1.1.1 Cognitive Science 2

1.1.2 Artificial Intelligence 2

1.1.3 Computational Linguistics 3

1.2 Natural Language Engineering 3

1.2.1 The General Philosophy of NLE 4

1.2.2 Scale 4

1.2.3 Robustness 5

1.2.4 Maintainability 5

1.2.5 Flexibility 5

1.2.6 Integration 6

1.2.7 Feasibility 6

1.2.8 Usability 7

1.2.9 The Use of a Full Range of Techniques 7

1.2.10 Cost-Benefit Analysis 8

1.2.11 Motivation for Adopting the NLE Approach 9

1.3 Methodological Criteria for Success 10

1.3.1 The AI Goal: Criteria for success 10

1.3.2 The NLE Goals: Criteria for success 11

1.4 Context of this Work: The LOLITA project 15

1.5 Terminology Issues 17

1.5.1 Meaning 17

1.5.2 Concepts 17
1.5.3 The Relationship Between Language and Concepts 18
1.5.4 Natural Language Generation 19
1.5.5 General Purpose Base 19
1.5.6 Planning 20
1.5.7 The Plan-realiser 20
1.6 Logical Progression of the Thesis 21

2 The Problem Area and Project Aims **23**
2.1 Natural Language Generation 23
2.2 NLG in LOLITA: Project Aims 25
2.2.1 Aim 1: The AI Goal and NLE Principles 26
2.2.2 Aim 2: Generation of SemNet Node Descriptions 26
2.2.3 Aim 3: Generation for Prototype Applications 27
2.2.4 Aim 4: The Suitability of SemNet 28
2.2.5 Aim 5: Broad Coverage 28
2.2.6 Aim 6: Suitability of Haskell 29
2.2.7 Aim 7: Evaluation 30

3 Related Work **31**
3.1 Organisation of This Chapter 31
3.2 General Criticism of State of the Art 33
3.3 Input to the Generator 34
3.4 Control 35
3.5 Architecture 37
3.5.1 Separated Systems 38
3.5.2 Integrated Systems 39
3.5.3 Architecture: Notes on Relevance 41
3.6 Realisation 42
3.6.1 Introduction 42
3.6.2 Functional Unification 42
3.6.3 Systemic Grammars 44

3.6.4 MUMBLE 47
3.6.5 Augmented Transition Networks 50
3.6.6 The Use of a Formative Lexicon 51
3.6.7 Realisation: Notes on Relevance 52
3.7 Planning 53
3.7.1 Early work 53
3.7.2 The Schema Approach 53
3.7.3 Rhetorical Structure Theory 55
3.7.4 Combination of Planning Resources 60
3.7.5 Planning: Notes on Relevance 62
3.8 The Generation Gap 62
3.8.1 SPOKESMAN 63
3.8.2 The PENMAN Upper Model 64
3.8.3 IGEN 66
3.8.4 WEIBER 67
3.8.5 Crossing the 'Generation Gap' with FUGs 67
3.8.6 Generation Gap: Notes on Relevance 67
3.9 Lexicalisation 68
3.9.1 The use of Discrimination Networks 68
3.9.2 The use of Taxonomic Knowledge Bases 70
3.9.3 Lexicalisation: Notes on Relevance 71
3.10 Creating Variation 72
3.10.1 Controlling Variation 73
3.11 Other Areas of NLG 74
3.11.1 Revision 74
3.11.2 Connectionism 74
3.12 Generation from a Semantic Network or Graph input 76
3.12.1 Generation from CDT 76
3.12.2 Generation from Conceptual Graphs 83
3.12.3 Generation from SNePS 91
3.12.4 Generation from MTM 94

3.12.5 Other Similar Work 101
3.13 Conclusions 108

4 The LOLITA System 110
4.1 History and Background 110
4.2 Advantages of General Purpose Base Research 111
4.3 System Overview 111
4.3.1 Syntactic Analysis 114
4.3.2 Knowledge Representation 116
4.3.3 Semantic and Pragmatic analysis 120
4.3.4 Dialogue 123
4.4 LOLITA Applications 125
4.4.1 Analysis of Text 125
4.4.2 Query 125
4.4.3 Translation 125
4.4.4 Database Front-end 126
4.4.5 Contents Scanning 126
4.4.6 Chinese Tutoring 128
4.5 The Role of Generation in LOLITA 128

5 Solution: The General Approach and the Plan-realiser 131
5.1 The General Approach 131
5.1.1 The Input 132
5.1.2 The Architecture: Introduction 133
5.1.3 The Architecture: The Role of the Planner 134
5.1.4 The Architecture: The Role of the Plan-realiser 139
5.1.5 The Architecture: The Interface between Planner and Realiser 142
5.2 Solution Detail : The Plan-realiser 144
5.3 Generation of Language-Isomorphic Concepts 144
5.4 Generation of Entities 146
5.4.1 How can an Entity be Adequately Described? 146
5.4.2 Realising Entity Concepts that are non LI 148

5.4.3 Determiners and Quantifiers 149
5.4.4 Describing Entities with Relative Clauses 153
5.4.5 Describing Entities with 'Special' Relative Clauses 154
5.4.6 Proper Nouns 156
5.5 Generation of Events 157
5.5.1 Generation of Actions 157
5.5.2 Generating Event Roles 159
5.6 Generation of Relative Clause Events 162
5.7 Complex Events 164
5.7.1 Causal Links 164
5.7.2 Events within Events 166
5.7.3 Temporal Links 168
5.8 The Realisation of Commands, Questions and Answers 168
5.8.1 Commands 169
5.8.2 Answers 169
5.8.3 Questions 169
5.9 Punctuation 171
5.10 Generation of Special Portions of Semantic Input 171
5.10.1 Internal Events 171
5.10.2 Time Representation 172
5.10.3 Positions 173
5.11 Generation of Anaphora and Referring Expressions 173
5.12 Conclusion 176

6 The Solution: Abstract Transformations 177

6.1 Other Work at this Level 178
6.2 Substitution of an Antonym Action 180
6.3 Transformations on Copula Actions 181
6.4 Transformations on Complemented Verb Pairs 181
6.5 Transformations on Multi-subject Events 184
6.6 'Give' Related Transformations 184

6.6.1 Making an Implicit Object Explicit 186
6.7 Other De-lexical Transformations 187
6.7.1 Example of De-lexical Rules for 'to have' 188
6.7.2 Example of De-lexical Rules for 'to make' 190
6.7.3 Cost-Benefit Analysis of the Rule Based Approach for Handling De-Lexical verbs 191
6.8 Generalisation or Specialisation of Concepts 194
6.8.1 Action Specialisation 195
6.9 Multiple Transformations 197
6.10 Conclusion 198

7 Implementation **200**
7.1 Implementation Overview 201
7.1.1 Some Important Types 201
7.1.2 General Operation 202
7.2 Features of Haskell 204
7.2.1 Referential Transparency 206
7.2.2 Higher-order Functions 206
7.2.3 Currying 207
7.2.4 Abstract Types 208
7.2.5 Lazy Evaluation 209
7.2.6 The Haskell Type System 210
7.2.7 Data Structures and Management 210
7.2.8 Prototyping 210
7.2.9 Suitability for Parallel Execution 211
7.2.10 Disadvantages of Haskell 212
7.3 Other Implementation Details 213
7.4 Conclusions 213

8 Evaluation and Results **215**
8.1 Evaluation of Natural Language Systems: A survey 216
8.1.1 Competitions 216

8.1.2 Galliers and Sparck Jones 216
8.1.3 NLG Evaluation 218
8.2 Example Evaluation 221
8.2.1 Introduction 221
8.2.2 The Evaluation Remit 222
8.2.3 The Evaluation Design 223
8.2.4 The Evaluation Review 227
8.3 Evaluation of Natural Language Systems: Conclusions 229
8.4 Results Versus Criteria for Success 232
8.4.1 Aim 1 232
8.4.2 Aim 2 239
8.4.3 Aim 3 240
8.4.4 Aim 4 246
8.4.5 Aim 5 249
8.4.6 Aim 6 249
8.4.7 Aim 7 249

9 Conclusions **251**
9.1 Successes of the Project 251
9.1.1 Theoretical Impact 251
9.1.2 Practical Impact 253
9.2 Project Shortcomings and Suggestions for Further Work 254

A Examples of Generator output **257**

B Summary of Systems **260**

C The Evaluation Instructions **270**

D Glossary **277**

List of Figures

3.1 Generator organisations 38

3.2 A simple example of unification of two FDs 43

3.3 Input to NIGEL for "John gives a blue book to MARY" 45

3.4 Top level systems of the NIGEL grammar 47

3.5 Realisation specification for "53rd Mechanised Division" 49

3.6 An ATN and associated grammar 50

3.7 The constituency schema 54

3.8 The RST schema 56

3.9 Text Structure for "Karen likes watching movies on Sundays" . . . 65

3.10 Example of annotated linguistic options in IGEN 66

3.11 Hovy's rhetorical goals and values 79

3.12 Example of a formal and informal text produced by PAULINE . . . 81

3.13 Example utterance graph input 84

3.14 Example APSG grammar rule 85

3.15 Example utterance graph input 87

3.16 Example lexicon entries 87

3.17 Example SNePS interaction 92

3.18 SNePS semantic network built by the example interaction 93

3.19 Example output from Kalos before and after revision 94

3.20 MTM SemR for sentence 1 97

3.21 KING's commercial transfer event 103

3.22 Examples of Horacek's terminological transformations 107

3.23 Variations caused by application of differing ZOOM schema 108

4.1 The LOLITA system 112

4.2 Example of parsing 115

4.3 An example of parsing the grammatically incorrect sentence 'and I likes him own' 116

4.4 Example of an event in the LOLITA representation 121

4.5 Example of a Chinese parse tree 128

5.1 Example of the 'story' command 140

5.2 Examples of implicit quantification associated with verbs 150

5.3 SemNet representation for 'John's motorbike' 155

5.4 An Example of the SemNet representation of positions 174

6.1 Example of antonym substitution abstract transformation 180

6.2 Examples of copula verbs and copula action abstract transforms . . 181

6.3 Examples of complemented action pair transformations 182

6.4 Example of a multi-subject transformation 184

6.5 Examples of 'give' related transformations 185

6.6 Examples of natural and unnatural uses of the de-lexical verb 'to have' with intransitive verbs 189

6.7 Examples of natural and unnatural uses of the de-lexical verb 'to make' with sentential verbs 190

6.8 Example generalisation paraphrases 195

6.9 Examples of verb specialisation by instrument clause 195

6.10 Simplified SemNet portion showing how instrument/action transforms can be made 197

6.11 Example of a multiple transformation 199

7.1 Simple input event and instruction passed to the plan-realiser . . . 202

7.2 Simplified portion of the NLG Haskell code 205

8.1 Framework for building an evaluation remit and design 219

8.2 Evaluation remit for the LOLITA NLG evaluation experiment . . . 224

8.3 Example Heap Profile 237

8.4 Example Time/Memory Profile 237

8.5 Example of a SemNet node with its generated NL description . . . 242

8.6 Example of a SemNet node with its generated NL description . . . 243

8.7 Portion of the SemNet from which the utterance in diagram 8.6 was generated 244

8.8 Example query session 245

8.9 Example of translation 246

8.10 Example of contents scanner 247

Chapter 1

Methodological Introduction

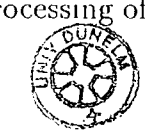
The subject of this research is natural language generation (NLG): the automatic generation of natural language (NL) by computer. Before this problem area is described, however, it is necessary to discuss important background methodological issues.

This chapter will begin by discussing three disciplines that provide researchers in natural language processing: cognitive science, artificial intelligence (AI) and computational linguistics. The methodology adopted in this work will then be described further by defining principles of Natural Language Engineering (NLE). Following this, methodological criteria for success will be provided for aspects of the AI goal and the NLE principles.

This chapter will also discuss the effects of the parent project and system, LOLITA, on the methodology adopted for the design of a NLG module (section 1.4) and clarify some terminological issues (section 1.5). Finally the chapter will discuss the logical progression of the thesis (section 1.6).

1.1 Traditional NLP Approaches

Natural Language Processing (NLP) lies at the crossroads of many disciplines all concerned with the automated processing of natural language (NL) using computer



systems. This section will briefly introduce the three main backgrounds to which researchers in this field tend to belong.

1.1.1 Cognitive Science

Generally, the aim of a Cognitive Scientist is to model processes in the brain. Specifically, cognitive scientists working in the field of NLP try to model the brain's communication processes. Work in this area is often founded on psychological and sometimes physiological experiments on how humans process language. These experiments are typically performed on children or on people with language disabilities. The aim of the cognitive scientist is to produce systems which not only have the same behaviour as humans, but also model the process which govern this behaviour. However, research in cognitive science is often restricted to small isolated areas of human behaviour such as, for example, how children learn to spell a few selected words. Furthermore, this research does not always result in a computer program; rather a computational model that could be implemented. Even if these models were implemented, they would only model those restricted areas in question.

1.1.2 Artificial Intelligence

Researchers with a background in artificial intelligence (AI) typically relax the constraints imposed by cognitive scientists: they aim for systems which mimic human behaviour without concern for whether the processes which lead to this behaviour are the same as those in the brain. They use whatever means are available to produce human-like behaviour. Because the aim of AI is to produce useful behaviour rather than an understanding of the mechanisms involved, the system has to cover a wider scope than the isolated processes studied by cognitive scientists. AI is the background approach adopted in the work presented in this thesis.

1.1.3 Computational Linguistics

Linguistics, the study of language, is yet another background providing researchers in NLP. The advent of computers provided a tool, first for testing, and then for developing, linguistic theories. The term computational linguistics (CL) was originally 'concerned with the application of a computational paradigm to the scientific study of human language' [Ballard and Jones, 1990]. However, CL has more recently expanded to include 'engineering of systems that process or analyse written or spoken language'[Ballard and Jones, 1990]. It is this latter branch of the discipline for which the term NLP is most frequently used. In practice therefore, the term CL is used by a wide variety of researchers: linguists working on the intricacies of language use, cognitive scientists using psycholinguistics, and scientists from an AI background adopting a more practical approach.

1.2 Natural Language Engineering

As mentioned above, this work is concerned with NLP from an AI viewpoint but this particular branch still encompasses a wide spectrum of methods. More specifically this work is concerned with Natural Language Engineering (NLE).

NLE is a recent endeavour which applies the ideas and practices of other engineering disciplines to the field of NLP. The use of the NLE approach is becoming increasingly popular as indicated by: the commencement of an EEC Language Engineering initiative; the publication of the 'Journal of Natural Language Engineering', and the increasing number of conferences devoted to practical applications of NL Systems (e.g., the recent ANLP conference in Stuttgart and the NLE convention in Paris).

The EEC LRE programme [LRE, 1992] defines Linguistic Engineering thus:-

> "Linguistic Engineering (LE) is an engineering endeavour, which is to combine scientific and technological knowledge in a number of relevant domains (descriptive and computational linguistics, lexicology and

terminology, formal languages, computer science, software engineering techniques, *etc.*). LE can be seen as a rather pragmatic approach to computerised language processing, given the current inadequacies of theoretical Computational Linguistics."

The next sections will detail the important aspects of NLE.

1.2.1 The General Philosophy of NLE

Traditional approaches to NLP, whether originating from a cognitive, linguistic or AI point of view, have tried to formulate either universal theories that cover all aspects of language or to develop very restricted but detailed theories that model small areas. The utilisation or expansion of these ideas to produce realistic systems which are not highly restricted by their task or domain has proved to be a great problem.

The belief adopted here is that there is a set of critical engineering criteria which should be applied to the field of NLP. While the more traditional research on core or specialised theories may be necessary for future improvements, the adoption of these NLE principles is important so as to utilise existing technology in order to produce useful systems. As new technology becomes available from more traditional methods, it can then be incorporated into a NL engineered system. However, the concentration of all resources on such improvements without consideration of how they are to be ultimately utilised is unproductive and will not fulfil the immediate demand for robust and versatile working systems.

The following subsections will detail important aspects of NLE.

1.2.2 Scale

The size of NLE systems must be sufficient for realistic large-scale applications. Properties such as the vocabulary size, grammar coverage, and the number of

word senses are critical. There are a number of ways in which to measure these properties: grammar coverage, for example, could be measured by the number of rules utilised, its perplexity, or the type of text that it can manipulate.

1.2.3 Robustness

Robustness in NLE concerns not only the linguistic scope of the system, but also the acceptability of effects when the input falls outside this scope. To quote [Galliers and Sparck Jones, 1993], "while it [robustness] may not be a serious problem for any individual application, it has to be faced up to in general"(page 45). At the very least a system should not crash when it receives input which is outside its scope: it should be able to carry on and try its best to cope with the conditions it is working under.

Robustness in NLE also encompasses the more general criteria for robustness imposed by software engineering practices.

1.2.4 Maintainability

Maintainability is a measure of how useful the system is over a long period of time. As in any large software system, the maintainability of a NLE system is important. Corrective (e.g., removing bugs), perfective (e.g., adding functionality), adaptive (e.g., changing the environment) and preventative (e.g., preventing future errors) maintenance will be required[1].

1.2.5 Flexibility

Flexibility or portability is a measure of the ability to modify the system for different tasks in different domains. This could be considered as maintenance, but is separated to emphasise the difference between major adaption accommodating

[1] See [Lientz and Swanson, 1980] and [Bennett *et al.*, 1990] for a software engineering view of maintenance.

large changes in task functionality (flexibility) and more subtle adaption due, for example, to smaller modifications to the functionality (perfective maintenance) or to changes in the environment (adaptive maintenance).

1.2.6 Integration

There are two related aspects of integration :-

- Firstly, system components should not make unreasonable assumptions about other components. Such assumptions are often made when specific NLP problems are tackled in isolation. Likewise, components should not attempt to perform tasks which belong in other components. The delimitation of the scope of each component is crucial and should not be made merely on the basis of what can be accomplished in the current state of the art. In some cases this delimitation is already well-defined (for example, the traditional separation into realisation and planning in NLG, see chapter 3). However in other areas, the delimitation may not be so well defined and may even depend on components which are not yet available. In these cases it is important that the technology required to build missing components exists or is at least achievable with research in the near future.

- Secondly, components should be designed and built to actively assist other components. So, for example, the design of the knowledge representation module should assist other parts of the system (e.g., parser, semantic analysis, generator etc). Even if the 'other parts' in question do not exist, components should be designed so that they will assist future components.

1.2.7 Feasibility

This aspect concerns ensuring that constraints on the running of the system are acceptable. For example, hardware requirements should not be too great and execution speed must be adequate. Feasibility incorporates making the system

and its components efficient.

Some areas of theoretical computer science and AI make extensive use of complexity analysis as a measure of the feasibility of algorithms. Algorithm complexity in NLE is not always paramount, due to the fact that in practice the 'size' of the data to which the algorithms are applied often has an upper limit. For example, when considering the processing of a string of words, an algorithm of high complexity (e.g., exponential) may perform better than one of lower complexity (e.g., polynomial) when the number of words has an upper limit. A complexity analysis which examines the worst case scenario may only be relevant for a few cases. If an algorithm is designed to process sentences (as is often the case in NLE) both the algorithm's complexity and size of the data string are important. A sentence could in theory be of infinite length, but in practice this will never occur. A better measurement of feasibility would be performance over sentences of average length.

1.2.8 Usability

Systems produced using NLE techniques should support the functions that real end-users want. These are often different from those that researchers think that end-users want and sometimes even different to what potential end-users say they want: careful gathering of requirements is necessary and may involve simulations. Delivered systems should also be user-friendly.

1.2.9 The Use of a Full Range of Techniques

NL engineered systems should use a full range of AI techniques. Where they are available, it is advantageous to use long-standing, well-worked and general theories from computational linguistics and logic (for example set-based semantics or multi-sorted object-orientated logic). However, a key aspect of NLE is that when these theories are not available alternative methods are employed. These alternatives range from more localised theories (which despite being unable to cover global possibilities are sufficient to handle what is required), knowledge based approaches,

individual heuristics to adaptive or evolutionary techniques. Incorporating such a wide range of methods ensures that the development of the system does not stall due to the difficulty in following a particular logical or linguistic theory while still allowing the benefits of such well established theories to be enjoyed.

1.2.10 Cost-Benefit Analysis

Cost-benefit analysis is an important aspect of other engineering based disciplines and should equally be applied to NLE. It is often the case that the best theoretical solution is not be the best practical one. There may, for example, be a trade-off between the depth and breadth of a solution to a particular problem. If a simple algorithm has only a slightly worse coverage than a very complex one then it might be better to use the former. Cost-benefit analysis involves reaching a balance between two or more aspects of NLE. So, for example, a simple algorithm may not have the same *robustness* as a more complicated one but may lead to a more *feasible* final system (e.g., one which has a more acceptable execution speed).

The process of cost-benefit analysis has been adopted by other researchers: most notably [Reiter and Mellish, 1993] try to apply such principles to NLG.

There are also meta-level economic issues of using extensive cost-benefit analysis. Such techniques are often dangerous as:-

- the cost of time spent in changing from one decision to another may out-weigh the benefit the change will have.
- the measurement of cost-benefit itself may cost more than the resulting benefit. This often happens in, for example, the commercial world: government surveys and reports often cost more than the potential savings, and court cases may cost more than the amount of money initially under question.

1.2.11 Motivation for Adopting the NLE Approach

More traditional approaches to NLP have concentrated on formulating central ideas, but the expansion of these ideas to a system with the properties listed in the previous sections has proved a major problem. This is reflected in the small number of systems which have the properties of a large-scale system compared to the abundance of smaller systems which carry out specific tasks in limited domains.

Developing a large-scale NLE system has **intrinsic research problems of its own**. For example, the methods used by a small-scale system with a few hundred nodes to manipulate a semantic network for inference purposes, may not be directly transported to a larger system with a hundred thousand nodes. In the small-scale case an association method may be employed where all the nodes are searched for a match; in the large-scale case this would clearly be impractical. Such problems apply to all the NLE attributes and not just to that of scale. For example, the execution speed (and thus the feasibility) in a restricted system may be unimportant and only cause problems when the system is expanded. Software development practices may improve the efficiency of algorithms to some extent, but this is unlikely to be sufficient if the complexity of algorithms is high: complexity would not become important until the scale of the system is made larger, when no software engineering development could improve the situation significantly (for issues of complexity in large-scale NL algorithms see [Long and Garigliano, 1994]).

A view often repeated among computational linguistics, that the movement from core ideas to a working NLP system is just a matter of software engineering development seems, therefore, to be unfounded.

The building and study of the properties of large systems is useful from an applied point of view, but can also help to investigate the bottleneck which causes the disparity between the large amount of theoretical work done in the area and the relatively small number of realistic working systems.

1.3 Methodological Criteria for Success

This section will detail the criteria by which the success of the methodological approach of this project may be judged. As discussed above, this is to approach the NLP problem with the AI goal of mimicking humans, and by following the NLE principles.

In some areas of science it is sufficient to define criteria for success at an abstract level. However, in this field there are additional, problem-dependent, criteria. The main reason for this is that NLP and NLG are difficult problems, and solutions to all areas of these problems are still a long way off. In order to provide reasonable and achievable criteria, a more detailed examination is required of how the problems are to be viewed. Therefore, this section will concentrate on problem independent criteria and a later section (chapter 2) will present the problem-dependent criteria once problem specific details have been provided.

After a solution to the problem has been presented (in chapters 5 and 6) it will be evaluated against both the methodological and problem dependent criteria for success. This evaluation will highlight the successes and failures of the proposed solution and of the project as a whole.

It is important to note at this early stage that evaluation is in itself a large unsolved problem[Galliers and Sparck Jones, 1993]. While for some areas of NLP well used evaluation techniques do exist, in others there are none. One of the specific aims of this work (see section 2.2) is to address the evaluation problem.

1.3.1 The AI Goal: Criteria for success

This section will examine the criteria for the success of the AI approach of mimicking human behaviour.

With the current state of the art, it is unlikely that within any branch of AI, a goal of mimicking all relevant human behaviour will be achievable in the near future. This is certainly the case in the area of NLP and NLG: it would be unrealistic to

expect a computer to have all the NL capabilities of humans, not least because humans can be unpredictable and make mistakes.

A more reasonable criterion would be to create system behaviour which reflects a small subset of that of humans. That is, any behaviour from a system should also be seen in humans, but not necessarily the reverse. However, this goal can lead to trivial solutions the worst being a system that does nothing or a system that is claimed to be modelling a human that is making mistakes.

Another problem is how to test for the successful mimicking of behaviour. One possibility is the well known Turing test. However, the criteria for success adopted in this project should not depend on trying to trick users into thinking the system is a human. Furthermore, a Turing test would require a complete system: it would be hard to evaluate sub-systems which aim to mimic a subset of human behaviour using such a test. A NLG system may, for example, fail the Turing test not because the generation module is not up to standard but because the system's world knowledge is not sufficient.

In summary therefore, the criterion for success with respect to the AI goal is that a small subset of an 'open' (i.e. no element of trickery) Turing type test is achievable. The system should be able to produce results that humans find acceptable.

1.3.2 The NLE Goals: Criteria for success

This section will examine the criteria for success with respect to the principles of NLE defined in section 1.2. It is important to note that these principles form very broad criteria and the current state of the art in NLP means that not all of them can be met for all areas of the NLP problem.

Scale:

Ideally NL systems should be able to process real-life, free text of any length. This aim, in the current state of art, is still far from being achieved. It is more reasonable to impose a limit on text length to, at most, a few paragraphs (this is the type of text adopted by the MUC [DAR, 1993] competitions for example). As well as defining scale by using a property (e.g., length) of the input text a system can handle, there are other measures of scale which could be used:-

- The size of the grammar, measured by the number of rules. However this measurement can depend on the particular formalism used (for example a unification formalism, see section 3.6.2, may need less rules than other formalisms to achieve comparable coverage) .
- The number of entries in the lexicon.
- The amount and depth of semantic knowledge. If the entries in the lexicon are not well related to each other then the 'depth of knowledge' of a system will be low. Measurable properties could be the number and size of hierarchies connecting information or the number of 'typical' events which give information about particular common actions.

Robustness:

Ideally systems should be robust in any domain. With the current state of the art however this is not achievable. If systems are to be portable and used across domains then a certain amount of system training will be required.

It is unlikely that total robustness in terms of correct behaviour will be achieved. A weaker criterion is thus that a system shows graceful degradation and never goes wrong in a bad way. It should not crash, thereby destroying other results, nor ignore the error. Behaviour which explains what has gone wrong is beneficial. As the system is developed, extensive testing is required in order to check the acceptance of both existing and new functionality. Robustness in the broader

Software Engineering sense (i.e., system crashes, infinite loops etc.) should also be achieved.

Maintainability:

A system which has successfully evolved over a long period of time with a high turnaround of researchers indicates good maintainability. To be successful, it must be possible for both the original developer and other programmers to understand the system so that they can perform maintenance (whether corrective, enhancement, perfective *etc.*, see section 1.2.4) in a reasonable time. To be maintainable, a system which is being developed by many people simultaneously must have strict revision control and testing mechanisms.

Flexibility:

The ability of a system to be used as a prototype in different domains or for different tasks shows good flexibility.

Possible measurements are the proportions of time and code that are spent on development in a specific domain or on a specific task, compared to that spent on general base (see section 1.5.5) development. For a highly flexible system this proportion of domain and task specific development will be low.

Integration:

Successful integration is indicated by:-

- The re-use of code. Various parts of the system may require similar abstractions, and should therefore, be able to share or re-use parts of other modules. There should be no repetition of functionality.
- Prototyping. The ability of a system to be used as prototype for many applications is a good indication that it is well integrated. One measurement

could be the proportion of code dedicated to a specific application compared to the core code (see *flexibility* above).

- Analysis of design. Integration can be measured by analysis of the design. A well integrated system should be easy to map onto a block diagram of the system's organisation.

Feasibility :

The acceptability of a system's execution time is dependent on the task for which it has been designed: execution time is much more critical for an on-line system than for one which is left to do some task overnight. For on-line systems, the ultimate aim would be for real-time operation. However, more realistically an execution time in the order of a few minutes would be acceptable (again this is task dependent). One criterion could be that systems should operate faster than a human doing the same task.

With respect to hardware requirements, an aim could be to produce systems which could run on cheap and widely available machines such as, for example, a 486 PC. Again, more realistically with respect to this stage of research, a Unix-based Sparc workstation would be acceptable.

Independently of these envisaged physical environment measurements complexity analysis on algorithms could be performed. As discussed in section 1.2.7 however, theoretical complexity is not as important as a more practical evaluation of algorithms. Other practical techniques such as profiling could also be employed.

Usability:

For a final product, the ultimate criteria for success is that end-users are happy with the delivered system. However, this is not practical in a research environment when aspects such as user friendliness are not as important as the core functionality. However, products should not be developed blindly with the assumption that any end-product will be useful. The use of simulation experiments (such as *'Wizard of*

Oz' simulations) with potential end-users are important to show that the problem of usability has not been ignored.

Use of techniques:

If complete well worked theories for all aspects of the NL system are not available (which they are not in the current state of art) then the other methods which have been utilised should be described.

This does not mean that to meet this criterion each subpart of a system must utilise every possible technique. If a particular problem can be solved using a single technique then it is of course irrational to force the use of others. But if a general technique does not lend itself well to a particular subproblem then the success criterion should be that an alternative technique is employed.

This criterion is related to the use of cost-benefit analysis (see below). It would be a big cost, for example, to redesign the whole of a general theory so as to accommodate one more single case. The cost of adding a simple rule exception would be far less with similar benefits.

Cost-Benefit:

Formal, extensive cost-benefit analysis is not a required criterion for success. As discussed in section 1.2.10, the cost of this analysis itself may greatly outweigh the benefits! Despite this, informal investigations of alternatives to various aspects of the system during its development is useful and should be undertaken.

1.4 Context of this Work: The LOLITA project

The work described in this thesis forms part of a larger project. LOLITA is the acronym for Large scale, Object-based, Linguistic Interactor, Translator and Analyser, a general purpose base (see section 1.5.5) NL system. A more detailed de-

scription of the LOLITA system will be left until chapter 4, but some its properties have an important bearing on the methodology adopted for the work described here:-

- The principles of Natural Language Engineering. The LOLITA system has been built according to NLE principles. These principles have already been detailed and their success criteria examined.

- The need for a generation module. When this project was initiated, the LOLITA system had very little NL generation capability. The LOLITA system had existing prototype applications and there were plans to develop new prototypes: there was, therefore, an urgent need for a module that could provide improved generation capabilities. The method adopted in this work, therefore, was to aim for a practical broad coverage generation system which could meet these demands. This is in contrast to an alternative method where a specific subproblem in the generation process might have been examined in much more detail.

- The use of Haskell. The LOLITA system is written in the functional programming language Haskell [Hudak *et al.*, 1994]. Although it would be possible to build the generation subsystem in an alternative language, interfacing problems and the desire for system coherence mean that the use of Haskell is a starting assumption for this work.

- The input to the generator. As will be described in chapter 3 the chosen input to a generation system is one of the most important factors constraining the generator's design. The LOLITA system uses a novel form of semantic network representation (SemNet, see chapter 4) and it is this representation that the generator has to take as input. Two particularly important issues related to the SemNet input are those of 'meaning' and the relationship between concepts and language. These issues are discussed in section 1.5 below.

- Specification requirements. Because the system developed in this work is part of the encompassing LOLITA system, its requirements for hardware and

execution time must be similar (or less) than those for the LOLITA system.

1.5 Terminology Issues

This section will discuss terminological issues on which there is no precise agreement in the research community or which might cause misunderstanding. A glossary is also provided to explain non-controversial terms which are used throughout this thesis.

The terminological definitions presented here are often derived from a deep philosophical background. However, these philosophical arguments are beyond the scope of this work[2].

1.5.1 Meaning

In LOLITA's SemNet representation, the *meaning* of a node (whether it be an entity or an event) is represented by that node together with the whole of the semantic network. The distribution of knowledge in SemNet means that nodes which are close to a specific node will contribute more to its meaning than those further away. However, it is impossible to define the meaning of a node by choosing an arbitrary distance and 'cutting out' a particular SemNet portion.

1.5.2 Concepts

A *concept* in the LOLITA system is any node in the SemNet representation. Its *meaning* is given by that particular node together with the whole of the semantic network (see above). A concept could therefore be a simple entity (for example the node representing 'cheese') or a very complex event (for example the assassination of Kennedy). Under this definition there is potentially an infinite number

[2]A book is planned to explain the philosophical assumptions on which the LOLITA system is based.

of concepts: some will be 'static' and correspond to LOLITA's background world knowledge, others will be 'dynamic' and be built as the LOLITA system runs (i.e as it analyses text and builds concepts to represent the text's meaning). Other researchers use the word 'concept' to mean something different: some use the term to describe a set of primitive concepts from which meanings can be built, others simply use it to mean entries in a knowledge base.

1.5.3 The Relationship Between Language and Concepts

A background assumption to LOLITA's SemNet representation is that language is concept driven: language has evolved so that words are available for concepts that need to be talked about. Whether a concept is 'needed' depends on the environment and culture. In different cultures and environments different concepts are required so a word for a particular concept may be present in one language but not in another.

The practical effect of this assumption is that LOLITA's SemNet representation comprises concepts (i.e nodes) which have a smaller 'grain size' than words: for every word there is a different concept (except for exact synonyms[3]) but there are many concepts that do not correspond to a particular word. Because some words have different senses, one particular surface level word may be related to more than one concept (for example the word 'strip' is related to many entity concepts as well as event concepts). Concepts which can be expressed with a word (or lexical entry) in a particular language are termed *language isomorphic (LI)* concepts.

Having said this, it is often useful to use language to identify useful concepts because language has evolved so that useful concepts can be talked about. The LOLITA system has used WordNet to help build its concepts (see section 4.3.2).

Other systems assume that concepts have a 'larger grain size' than words: they utilise 'primitive concepts' that can be expressed by a variety of words (the most

[3]It could be argued that even synonymous words correspond to different concepts because, for example, they convey different stylistic effect.

extreme example being CDT [Schank and Abelson, 1977] where, for example, the concept **INGEST** could be expressed by the words 'drink', 'eat' or 'breathe', see section 3.12.1). Other systems assume a one-to-one correspondence between concepts and the words that can be used to express them (systems based on MTM, see section 3.12.4, for example, comprise nodes which are lexical surface strings).

1.5.4 Natural Language Generation

The term *natural language generation (NLG)* is used to mean different things. All researchers agree that the output of a NLG module should be an utterance in a surface language, but they define the NLG task differently with respect to the input it receives and the processes it has to carry out (for example, some consider generation to include the triggering of the urge to speak and the delimitation of content, whereas others consider generation to be the simple realisation of some detailed specification of what has to be said).

In this thesis NLG is defined as the process of producing English utterances given the whole of LOLITA's SemNet representation as input.

1.5.5 General Purpose Base

A *system* has been defined [Galliers and Sparck Jones, 1993] as the entire automatic software and hardware entity. An NLP system carries out a *task* and any system which is used to perform a task in a specific domain is an *application.* A *generic system* is designed to perform a certain task, or more broadly a task type, in different domains: it can be tailored (by adding domain specific resources) to different applications. *General purpose* systems are intended to be directly usable without further tailoring for more than one application. Galliers and Sparck Jones state that "*general purpose systems do not exist even for any one NLP task, let alone a range of tasks*". They also note that "*within certain limits, or on certain assumptions about the scope of language processing, generic NLP systems are essentially general purpose. i.e they will serve language-processing needs within any*

task system". It is at the intersection between a general purpose and a generic system that LOLITA belongs. It is more than a generic system as it is not restricted to a single task type, but it is not, as it stands, a general purpose system which can be used for any task in any domain. The terminology is extended by defining LOLITA as a *general purpose base.*

1.5.6 Planning

Many researchers include a *planning* module or process in their generation systems, but, again, the term is used for differing things. Some researchers, for example, include content delimitation in their definition of planning, others use planning to refer to the process of organising clause sized predicates (see chapter 3).

In the work presented in this thesis, the definition of the planning process is different from those adopted elsewhere. To avoid the 'generation gap' (see section 3.8) some responsibility may be moved from the planner module to the plan-realiser. The planning module will pass suggestions of how an utterance should be produced to the plan-realiser. A more precise definition of planning in this work, together with a description of how it interfaces with the plan-realiser, is left until chapter 5.

1.5.7 The Plan-realiser

The traditional approach to generation includes a *realiser* as well as a planning process (see chapter 3). Again workers apply this term in different ways. To emphasise the difference between the approach taken in this work and the more traditional methods, the term *plan-realiser* will be used instead of *realiser.* In this approach the *plan-realiser*, in the absence of detailed instructions from the *planner*, will be autonomous and perform some tasks that are more traditionally assigned to planners. This distinction will be detailed in chapter 5.

1.6 Logical Progression of the Thesis

The thesis is organised according to the following plan :-

Chapter 1: Methodological Introduction (this chapter), provides important methodological information about the work presented in the thesis. The chapter provides a detailed explanation of the natural language engineering (NLE) methods adopted and provides background criteria for success for these methods as well as to those of the artificial intelligence (AI) approach. The chapter also describes how the parent project, the LOLITA system, influences the methods adopted and provides a discussion of controversial terminology.

Chapter 2: The Problem Area and Project Aims, provides an overview of the problem of natural language generation and lists the seven different aims of the project together with criteria for their success (which are a refinement of the general criteria discussed in chapter 1).

Chapter 3: The State of the Art, will discuss the state of the art in the area of NLG by examining the different problems and approaches to their solution. After giving a broad overview of these areas the chapter will concentrate on those systems that take similar input to the LOLITA generator. Because the chapter is largely organised by subproblem rather than system, Appendix B provides a system by system description with cross references back to this chapter. The appendix also contains a table summarising some of the important properties of these systems.

Chapter 4: The LOLITA System, provides details of the parent project LOLITA. The chapter will discuss the advantages of the general purpose base approach used in LOLITA together will details of its subcomponents and prototype applications. Special attention will be paid to those components and applications that are of relevance to NLG.

Chapter 5: Solution: The General Approach and the Plan-realiser, discusses the novel framework adopted for generation before detailing one subcomponent of the solution, the plan-realiser. The first part of the chapter discusses the adopted architecture and how the roles of the components (the planner and

plan-realiser) differ from other approaches. The later component, the plan-realiser, is then discussed in more detail and heuristics and examples are provided.

Chapter 6: Solution: Abstract Transformations, provides details of another aspect of the solution. Abstract transformations are transformations which act on the SemNet input to the generator, giving rise to paraphrases. The chapter discusses other systems which perform a similar process before giving specific heuristics and examples.

Chapter 7: The Implementation, provides some implementation details. The LOLITA generator is implemented in the functional programming language Haskell. The use and advantages of this language for NLG will be examined with particular focus on how the properties of such a language have an impact on the solution. This investigation is one of the specific project aims listed in chapter 2.

Chapter 8: Evaluation, discusses the state of the art in evaluation of NL (and more specifically NLG) systems before going on to evaluate this particular project. The study of evaluation techniques (which is another specific aim of the project presented in chapter 2) includes details of one particular evaluation framework together with suggestions for its extension. A detailed example of how this framework can be applied is given in the form of an evaluation example. The second part of the chapter takes each of the project aims detailed in chapter 2 and discusses whether or not their criteria for success (also provided in chapter 2) as well as the general methodological criteria (chapter 1) have been met.

Chapter 9: Conclusion, the final chapter, summarises the project's theoretical and practical successes. It also describes some of the shortcomings of the project and suggests possibilities for further work.

Chapter 2

The Problem Area and Project Aims

2.1 Natural Language Generation

Natural Language Generation (hereafter NLG) is the automatic generation of Natural Language by computer in order to meet communicative goals. One aim of NLP is to allow more natural communication with a computer and, since communication is a two-way process, a NL system should be able to produce as well as interpret NL text. A computer system which responded with internal jargon would be unsatisfactory. NLG is not just the process of using NL output: programs have been printing out messages in English since the advent of computers. However such 'canned' text (e.g., **PRINT "please type your name"**) or text that can be parameterised with variables (e.g., **PRINT "there is an error in module" + MOD$**) do not represent anything to the program and any connection between the string of words and the state of the program are restricted to the mind of the person who preprogrammed the responses [McDonald, 1990]. Nowadays however, programs need to communicate a much wider range of information to their users and a simple canned or template approach is often insufficient.

NLG is recognised to be a challenging area of NLP: Gabriel, for example, goes

as far as saying that "writing is the ultimate problem for artificial intelligence research" [Gabriel, 1988]. This claim can justifiably be opposed by researchers in other AI areas but NLG is often an important way in which the manifestation of results from other modules can be presented.

Until recently, it was common for work in the area to start with the statement that NLG was a young field which previously had not been required due to the lack of sophistication of underlying programs. It was claimed that any generation that was needed was simple and could be done with, for example, simple canned text. However, this is no longer true, sophisticated NL systems have emerged and impressive NLG modules have been built for them. Over the last 15 years, NLG has become one of the fastest growing areas of NL research.

NLG, like NLP in general, has been approached from different viewpoints (see section 1.1) and people have concentrated on different aspects and adopted a wide range of initial assumptions.

One interpretation of the division of the problem [Mykowiecka, 1991a] is:-

1. Choosing the contents:
    - Choosing the facts which are adequate in a particular context,
    - Different treatment of new information and the facts which are already known to the reader/hearer.
2. Constructing the plan :
    - Ordering the facts to be presented,
    - Deciding on the subjects of sentences,
    - Choosing the contents of each sentence,
    - Choosing the form of sentence structure.
3. Final realisation:
    - Ordering the sentence parts,
    - Choosing the proper words,
    - Choosing the proper morphological forms.
4. Reviewing.

Many systems group the tasks required for generation into two components, a planner and a realiser. This traditional architecture, however, often causes problems at the interface between the two modules: a phenomenom termed the *generation gap* (see chapter 3). In order to avoid this problem, the solution to NLG presented in this thesis adopts a novel architecture in which the distribution of tasks between the two modules is different. This solution will be detailed in chapter 5.

Although a lot of work has been undertaken in the field, human NLG is far more sophisticated than the current state of the art in automated NLG. A detailed examination of the history and state of the art in NLG is presented in chapter 3.

2.2 NLG in LOLITA: Project Aims

The underlying aim for this project was to build a natural language generation module for LOLITA in order to increase its generation capabilities (see section 1.4). The prototype applications that already existed, were being designed, or were to be designed in future, needed such capabilities. NLG was also required to aid in development: the SemNet representation on which LOLITA is based can be very difficult to understand quickly and a NL utterance to describe each node in the SemNet would be beneficial. This underlying aim is very abstract and could lead to anything from a trivial to a very complex generation solution. Therefore there is a need to refine this underlying aim into more concrete aims to which can be assigned criteria for success.

For success in this project, it will not be sufficient to tick off the aims casually and say that they have been fulfilled. Instead, detailed success criteria for each aim have to be established *a priori*. When the solution has been presented these criteria for success will be re-examined in the evaluation chapter (chapter 8).

2.2.1 Aim 1: The AI Goal and NLE Principles

The project should follow the background methodological criteria for success with respect to the AI goal and the principles of NLE discussed in chapter 1. The NLE principles introduced in chapter 1 can be applied to the problem of NLG. Adherence to some of the principles is already implicit in other aims (e.g., aim 4 is related to *scale*, aim 3 is related to *integration*). However, this aim makes them all explicit.

Aim 1: Criteria for success

The methodological criteria for success for the AI goal have already been discussed in section 1.3.1.

The NLE principles discussed in chapter 1 can be directly applied to the subproblem of NLG. The design, development, implementation and evaluation of the generation system must follow these principles. However the background methodological criteria discussed in chapter 1 are issues which apply to NLE in general: it is not always necessary for a solution to each individual subproblem to meet all of the possible criteria discussed. If, for example, evidence for the feasibility of a system is provided by practical results, theoretical complexity analysis need not be undertaken.

2.2.2 Aim 2: Generation of SemNet Node Descriptions

To build a generator which can produce English descriptions of nodes in the semantic network. The 'general purpose base' operation of the LOLITA system is to analyse text and produce a semantic representation of its meaning (SemNet). The generator should be able to produce expressions from these semantic representations. This is a vital requirement which, not least, is essential for future development and 'debugging' of the rest of the system.

Aim 2: Criteria for success

The possible success criteria for this aim could be very wide ranging. At the least, the generator could simply produce a single word utterance to which the concept node is most closely attached. At the other end of the scale however, a node could be represented using much more information. If for example, a node in the semantic network represents the assassination of J.F.Kennedy, the possible NL utterances could range from 'assassination' to a very long 'novel-length' utterance (people have written many books and even made films to represent such a node). An acceptable and achievable criteria for success obviously lies between these two extremes.

The amount of information represented by a node could affect the length and complexity of a NL utterance to describe it. The best way to constrain the amount of information is to limit the length of the input information from which the concept information is gleaned. The current state of the art (e.g., MUC [DAR, 1993] competitions etc, see section 8.1.1) is constrained to input in the order of a few small paragraphs of input text (for example, newswire bulletins). This is a reasonable constraint to impose.

A background methodological criterion is that the system output mimics the behaviour of humans. The success of the system output will therefore depend on its comparison with human produced descriptions of nodes in the SemNet representation. The methods (and experiments) of testing for this criteria will be discussed more fully in chapter 8

Because the need for NL descriptions is important for future development, a final criterion for success will be that utterances produced by the generation system have indeed been useful for this purpose.

2.2.3 Aim 3: Generation for Prototype Applications

To build a generator with sufficient capabilities for the existing LOLITA prototype demonstrations. To highlight any inadequacies of these underlying sub-components which hinder generation.

Aim 3: Criteria for success

Evaluation of generation is difficult but is made easier when a scope and context is defined. Generation capabilities within the framework of an application are simpler to evaluate than in abstract terms. The criteria for the success for this aim is that the generation capabilities are sufficient for the existing prototypes. This is different from evaluating the applications themselves although interrelated because the final manifestation of a result is via NLG.

2.2.4 Aim 4: The Suitability of SemNet

To investigate the suitability of LOLITA's representation for generation. The use of LOLITA's semantic representation (SemNet, see section 4.3.2) is an initial assumption for the project but an explicit aim to investigate its suitability is useful. Having said this, it is always important to remember that SemNet was not designed specifically for generation. The representation used in other generation systems may have been designed specifically for generation and could be unsuitable for other tasks. This is an aspect of integration, see section 1.2.6.

Aim 4: Criteria for success

The criteria for success for this aim are not concrete. The project will merely discuss and draw conclusions as to whether the generator is helped or hindered by the input representation. As will be discussed in section 3.3 the type of input a generator takes will have a very important bearing on its solution. It would be very difficult to build a generator in isolation and use different inputs so as to ascertain which is most suitable: this will not be a success criteria.

2.2.5 Aim 5: Broad Coverage

To aim for broad coverage on the problem of generation. The project will not confine itself to one particular subproblem in the generation process. Rather than

picking out a particular generation sub-problem and designing a solution to solve it (for example generating referring phrases, lexicalisation, anaphora, specialised domain or task, see chapter 3), the onus on the project will be to produce a practical system which covers all the associated subproblems. This aim is driven by the starting point requirement to build a generator for the LOLITA system (see section 1.4). This will of course lead to an oversimplification of some of the problems but the use of cost-benefit analysis will help decide how 'deep' into each of these subproblems the solution will be required to delve. It is important to note that in future the LOLITA system does not expect to remain at this 'shallow' but 'broad' situation. As solutions to the individual problems are found (either elsewhere or by continuation of development) good integration should ensure that they can be incorporated into the existing LOLITA generator.

Aim 5: Criteria for success

The success of the aim to provide broad coverage is highly dependent on the success criteria of other aims. In order for aims 1 and 2 to be successful, for example, the solution must cover a broad range of subproblems. It is, of course, not necessary to show that these various sub-problems are handled too 'deeply' in order to achieve this broad coverage.

2.2.6 Aim 6: Suitability of Haskell

To investigate the suitability of the functional programming language Haskell for NLG. Although for reasons of coherence and interfacing with the parent LOLITA system, the use of Haskell is a starting assumption for this work (see section 1.4), it is useful to ascertain what effects this assumption has on the implementation of the solution.

Aim 6: Criteria for success

To be successful with respect to this aim, the unique features of Haskell must be examined and the impact on the implementation (both negative and positive) discussed. Features common to other languages (especially imperative languages) that are not present in Haskell will also be discussed. The investigation should determine whether other languages will allow things to be done more easily, and what is specific to Haskell that aids the implementation.

The success criteria is not to prove that Haskell is better or worse than an alternative language, as this would, of course, be extremely subjective.

2.2.7 Aim 7: Evaluation

Evaluation of NLP and in particular NLG systems is a difficult and not well understood process. It is therefore an explicit aim of this work to investigate existing NLG evaluation techniques and perhaps suggest alternatives.

Aim 7: Criteria for success

The discussion on how to evaluate success criteria for aims 1 to 6 has shown that methods available for evaluation of NLE systems and in particular NLG are vague. As well as needing good evaluation techniques to evaluate a NLG system, a NLG system is needed to evaluate evaluation methods. This work is also an investigation of evaluation methods. To be successful in this aim, the project should add to knowledge about how to evaluate NLP and in particular NLG systems.

Chapter 3

Related Work

Natural Language Processing (NLP) and in particular Natural Language Generation (NLG) are comparatively new endeavours and there has been a great deal of recent work in the area. This is highlighted in the ever increasing number of workshops, conferences and journal issues dedicated solely to the subject (see, for example, [NLG94, 1994], [Cercone and Pattabhiraman, 1992],[Dale *et al.*, 1992],[Dale *et al.*, 1990], [Horacek and Zock, 1993]).

3.1 Organisation of This Chapter

Despite being such a young field, the problem of NLG has been already approached from a wide variety of backgrounds in a wider number of ways and it would be impractical and unnecessary to give a detailed overview of all significant work in the area. This chapter will be split into two parts: the first (sections 3.2 to 3.11) will present a broad overview of the field, the second will examine in more detail those systems and techniques which are deemed to be more relevant to the work presented in the rest of this thesis.

The first part of the chapter will present a brief overview of some of the most important subproblems and pioneering approaches to their solution. This overview will include sections on what input generation systems assume (section 3.3), the

control mechanism of generation systems (section 3.4), different architectures they adopt (section 3.5), approaches to realisation (section 3.6) and planning (section 3.7), the problem of the generation gap (section 3.8), lexicalisation (section 3.9), how generation systems produce variation (section 3.10) and finally a section on other areas of NLG such as connectionism and revision (section 3.11).

The section on more relevant work (section 3.12) concerns generation systems which assume a semantic network or graph input. As will be discussed in section 3.3, the assumption on the type of the input is one of the most constraining aspects on the design of a generation system. The section will detail systems and approaches based on Conceptual Dependency Theory (CDT) [Schank, 1975], Conceptual Graphs [Sowa, 1984] , the Semantic Network Processing System (SNePS) [Shapiro and the SNePS Implementation Group, 1993] and the Meaning Text Theory (MTT)[Mel'čuk and Polguère, 1970].

This chapter is not meant to be a stand-alone description of the aspects of the state of the art in NLG, but a more active part of the thesis. Therefore the overview will include discussion about the relevance of different aspects with respect to systems that take a semantic network or graph as input and, more specifically, to the generator described in this work. A criticism of different techniques and systems will also be given; especially for the 'more relevant work' presented in section 3.12. There are a few important criticisms that can be applied to the field in general. Thus, as well as presenting a criticism of each individual system or technique as they are introduced, a section is presented which discusses some of these important common criticisms (section 3.2). The reader should bear these general criticisms in mind whilst reading the rest of the chapter.

As outlined above, the chapter has been organised by subproblem rather than by system or school of thought. Some systems cover more than one subproblem therefore information about individual systems is often distributed throughout the chapter. Furthermore, some systems have evolved and have been given different names, with only the researcher's name being a constant. For these reasons Appendix B is provided to detail NLG work organised by system. This appendix, as

well as cross referencing different systems with the information contained in this chapter, provides a table summarising the properties of each system according to the properties described in this chapter.

3.2 General Criticism of State of the Art

This section will discuss criticisms that can be applied to the field of NLG in general. While it is definitely not the case that all these criticisms apply to all the systems and approaches, there are few, if any, systems to which none apply.

The overriding criticism is that most NLG work does not bear up to the important NLE principles defined in chapter 1. More specifically:-

- Many systems seem still to be tied to particular limited domains. Even when researchers claim that the techniques they adopt can be transported from domain to domain, the actual examples they give do not provide much evidence for this.

- Many systems can still be described as 'toy' systems. Although in theory the rules could be applied to large-scale systems (see section 1.2.2), actual implementations have only been built on a small 'toy' scale. Problems of feasibility as the scale is increased have not been addressed.

- Some systems are based on assumptions which tie them to a particular natural language. Some, for example, assume an isomorphic correlation between concepts and words (see section 3.9).

- Some systems, more commonly those that concentrate on a particular subarea of NLG, place unlikely assumptions on the output of other components (this is the *integration* problem). For example, some realisers assume a very rich input specification (section 3.6) and planners often assume the information which they have to organise is already presented (by some missing module) in clause-sized chunks.

- Finally, it is often very difficult to determine the strengths of a particular generator system. It would seem likely that scientific papers concerning NLG would be littered with example output but this is not the case. Even when example output text is presented it is often misleading as the starting point assumptions (i.e. the type of input) is not made explicit. There is a clear need for researchers to present more information so that their generation systems can be more easily evaluated. Investigation into how NLG systems can be evaluated is a specific aim of this work (see section 2.2).

3.3 Input to the Generator

One of the most important factors which determine a generator's characteristics is the input it assumes. The type of input dictates the delimitation of the problem as well as constraining the approaches to the design of its solution. Some generation systems are responsible for a wide range of subproblems such as triggering the urge to produce an utterance, choosing and ordering the content of the utterance and final grammatical realisation. Others restrict the area of the problem by building a component to cover one of the subproblems and assuming the rest is tackled elsewhere. (The most common group of these systems are grammatical realisers which produce a grammatical utterance for some specification: the specification is assumed to be provided by other components.)

Generation systems can be coarsely split into two groups: those that assume the content is a side effect of the application program and those that take on the responsibility of extracting the content from the application program. Meteer[Meteer, 1993](pg.26) argues that the first group is typical of more 'active' programs such as simulations, expert systems or automatic translators; the latter being a good example as there is no distinct independent application and the content is determined by the input text. Less active underlying systems such as databases, which are static in the sense that they are not taking actions or passing representations of actions to the generator, have to determine the content themselves.

Typical inputs to a generation system are as follows :-

- Those which comprise a knowledge base and a communication goal. According to the goal, relevant knowledge must be retrieved from the knowledge base and expressed in a coherent manner.
- Those which assume that the content of the utterance is provided in the form of clause-sized chunks. An utterance is produced by the generator by ordering the clauses into coherent sentences.
- Those which assume a complete specification of an utterance. They assume that 'higher' modules in the system have provided the specification in sufficient detail. In some cases these higher modules exist, but in others they are simulated. The detail of the specification differs from system to system. Some completely define the grammatical as well as semantic content of the utterance and leave only the 'read out' and morphology to the generator. Others are more vague leaving more grammatical choice to the generator.
- Another group of systems take as input a semantic representation of the information to be expressed in form of a network or graph. This is the type of input to the generator described in this work. Other systems which take the same type of input are the most comparable and will be examined in more detail in section 3.12. It will be seen that even within this group of systems, there are variations as to the exact type of input and the methods used to produce an utterance.

3.4 Control

Another categorisation of NLG systems is according to their method of control. Meteer [Meteer, 1993] distinguishes between two types of control:-

- In a *grammar directed* (or *declarative* [Paris and McKeown, 1987]) system, control lies in the reference knowledge, that is it is governed by some predetermined body of tests that gate and order actions.

- In a *message directed* (or *procedural* [Paris and McKeown, 1987]) system, control lies in the input itself and is interpreted by some general control loop within the process.

This distinction is applicable to all stages of the generation process whether it be during content delimitation, planning or realisation. Meteer highlights the difference between the two methods of control by examining the interface between the application program and planner:

> "In a message driven system events in the application trigger the 'urge to speak' and provide the structures that determine the content of what is to be communicated. This is in contrast to systems where the application program only provides a 'communication goal' and the generator searches the knowledge base of the application for information to fulfil the goal using the generator to control the search."

An example of a message directed control generation system is SPOKESMAN (see section 3.8.1). Paris and McKeown [Paris and McKeown, 1987] also argue for a procedural approach especially when generating text to describe physical objects. In this case they claim that the structure of the text often mirrors the structure of the object being described. They use an analogy (originally identified by Linde and Labov [Linde and Labov, 1975]) to a person describing an apartment: the description could follow the layout of the apartment with the speaker taking an imaginary tour through the different rooms. Examples of grammar directed or declarative systems are the NIGEL realiser (see section 3.6.3) and most RST and schema based planners (see section 3.7.2).

[McDonald *et al.*, 1987] argue that because the action sequence is already implicitly determined by the process that built the input structure and no effort needs to be expended on control decision, message directed control is more efficient. On the other hand, other researchers (e.g., [McKeown and Swartout, 1988]) state that the description directed or procedural control affects the clarity of the system. They also argue that for message directed realisation systems there is no distinct

grammar which means it is difficult to examine, adapt or expand.

There is a very strong link between the method of control and the type of input assumed by the generator (see previous section). If, for example, input is just a list of clause size predicates then a declarative system will have to be adopted. If on the other hand the input is richer and contains information such as temporal progression and causal information it would be irrational to ignore this information and mirror ordering knowledge in a declarative generator. Generally generators with static underlying programs (such as databases) adopt a declarative approach whilst more active underlying programs (such as expert systems and translators) adopt a procedural approach.

However, the categorisation of systems into the message and grammar directed approaches is not alway possible. Rather, the definitions lie at either end of a range with different systems shifting different emphasis to either control method without belonging wholly to one category.

Generators which start from a semantic graph representation usually contain rich semantic information which is useful for formulating utterances and thus tend to place emphasis on message directed or procedural control. This is indeed the case for the LOLITA generator which despite having a certain degree of declarative control is highly dependent on the semantic network representation (SemNet) it receives as input.

3.5 Architecture

As introduced in chapter 2, generation involves three distinct stages (although some include a fourth revision stage): choosing the information to communicate, organising this information and then realising it in NL. Traditionally, however, these processes have been lumped together into two general classes of decisions: one involves determining the content and organisation of the text, the other concerns choosing lexical items and syntactic constructions which can present the information most effectively. These 'what to say' and 'how to say it' stages have been

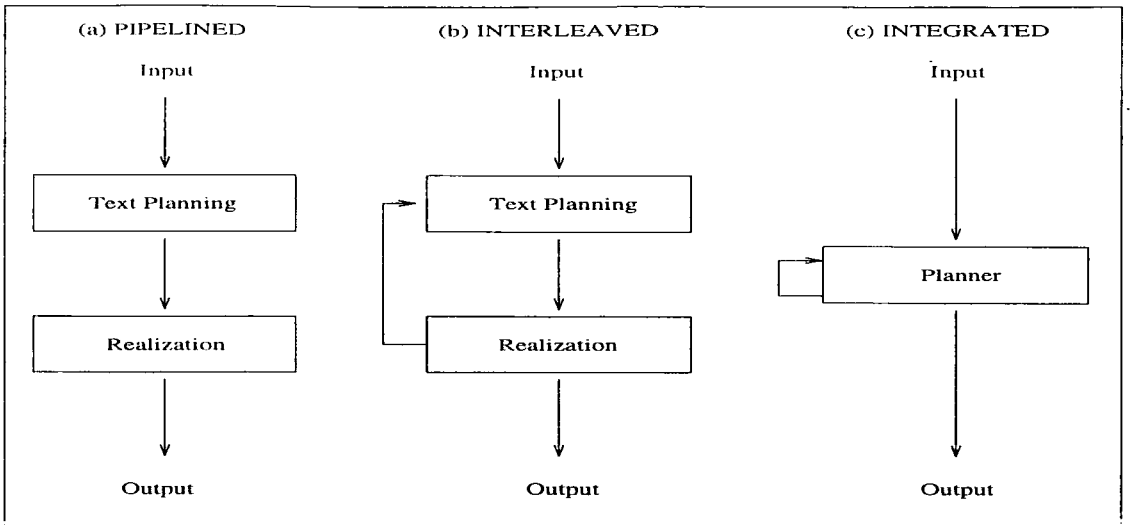


Figure 3.1: Generator organisations

called *Planning* and *Realisation* components, *Strategic* and *Tactical* levels, *Deep* and *Surface* generation, *Text planning* and *Plan execution*, *Message* and *Form* levels, *Functional* and *Positional* levels and *Conceptual* and *Grammatical* levels. This section will provide an overview of approaches to the architecture of generation systems.

Assuming this distinction between planning and realisation tasks, [Kantrowitz and Bates, 1992] claim that there are two genres of generation architecture: separated systems and integrated systems. Separated systems can be further categorised into pipelined (or linear) and interleaved systems (see figure 3.1). These generation architectures will be introduced in the following subsections.

3.5.1 Separated Systems

Some researchers claim that it is best to have a separate module for each of the two main tasks. For example McDonald and Meteer [McDonald and Meteer, 1988] suggest that:-

> "(we should modulise) our systems so that the parts which handle well understood processes need not be compromised to accommodate weaknesses in other parts of the system."

In the case of NLG, McDonald claims that the well understood process is linguistic realisation while weaknesses are in the conceptual models and representations of the programs underlying the generator. He thus developed his MUMBLE realiser (see section 3.6.4) independently of a text planner (later, Meteer developed SPOKESMAN to interface MUMBLE with a variety of underlying programs, see [Meteer, 1993] and section 3.8.1).

The approaches to realisation and planning in separated systems are introduced in sections 3.6 and 3.7.

It has become apparent, however, that the original realisation and planning split is more complicated and a linear or pipelined execution of each component is not necessarily the best way forward. The problem arises because semantic and syntactic structures are not isomorphic [Elhadad and Robin. 1992] so that meanings of lexemes and more generally grammatical functions are related unevenly to conceptual meanings [Horacek, 1992]. Researchers have discovered that finding suitable lexical information and grammatical forms to express conceptual information is not necessarily a straight forward task (see also [Rubinoff, 1992]).

This problem has been called 'the generation gap' [Meteer, 1993]. Section 3.8 describes how this problem has been tackled by the use of either a pipelined or interleaved interface between the two modules.

3.5.2 Integrated Systems

The alternative to having separate components is motivated by the same 'generation gap' problem. Kantrowitz and Bates, in defence of their integrated system GLINDA , state [Kantrowitz and Bates, 1992]:-

> "Unfortunately in generating even modestly sophisticated texts the planning stage is not independent of the realisation stage. In particular if the planner isn't aware of syntax, it can't take into account opportunities and inadequacies that arise during the realisation stage."

The integrated approach is however relatively rare and will not be discussed further except for the following summarisation of the main systems which come into this category:-

- DIOGENES [Nirenburg *et al.*, 1989] is unusual as it is based on a *blackboard architecture* [Engelmore and Morgan, 1988].The blackboard architecture uses an agenda-style control with each decision module operating independently and posting its decisions on the 'blackboard'. There are five decision making modules in DIOGENES (text structuring, lexical selection, syntactic selection, co-reference treatment and constituent ordering) and whilst the ordering of the decisions is not fixed, each has a different base priority level. This means that initially text structuring decisions are made before lexical decisions for example. However, this strict ordering is not maintained as decisions can be retracted. A criticism of this approach (e.g., [Meteer, 1993]) is its inefficiency: whenever a decision is retracted all other decisions which are dependent on it must also be re-checked.

- GLINDA [Kantrowitz and Bates, 1992] is the generation module of the OZ 'virtual reality' system. Its ambitious aims are to :-

  > "tune the generation to engender subtle emotional reactions in and exert influences on the human 'player' and to present a variety of vivid views of the simulated world'.

  Because of the need for more flexible communication between what would be separated planning and realisation levels, the strategy was to identify similarities in the operations and representations of both levels and to generalise them into a single framework that can be used for all aspects of generation. The resulting integrated generator has a single engine and only the organisation of the rules creates modularisation.

- Appelt's KAMP [Appelt, 1985] was one of the first systems to break away from the traditional separated approach. Appelt argues that the planning and action are essentially the same process:-

> "Human language behaviour is part of a coherent plan of action directed toward satisfying a speaker's goals... Thus, the planning at each level involves consideration of both linguistic rules and goal satisfaction. the distinction between 'what' and 'how' then becomes merely two points on a continuum between goal-satisfaction and rule-satisfaction"

The KAMP system uses a uniform representation method and planning mechanism (based on Sacerdoti's NOAH [Sacerdoti, 1977]) throughout.

KAMP begins with a set of axioms about the state of the world, agent's beliefs and goals, and knowledge about physical and linguistic actions. The complex planning mechanism involves building a network where each node represents possible world and arcs represent actions which change the state of these worlds. The KAMP planning mechanism is criticised for its complexity (it took nearly an hour to produce one complex sentence [Meteer, 1993],pg.158) and more recently Appelt has modified the integrated approach by separating out the linguistic component TELEGRAM [Appelt, 1983].

3.5.3 Architecture: Notes on Relevance

The majority of generation systems adopt a separated approach comprising a planner and a realiser. The distribution of tasks between these modules differs from system to system but, in general, the planner does much of the hard work (e.g content selection and delimitation, clause organisation) whilst the realiser simply produces a surface level utterance from the planner's (often detailed) specification.

The approach taken in this work is also based on a separated architecture. However, the distribution of tasks and effort between the two components is different. The **planner** does provide instructions but not necessarily highly detailed ones. The **plan-realiser** (named so as to distinguish the module from traditional realisers) has to follow these instructions but, in the absence of specific details, it must perform tasks that are more traditionally achieved by the planner. The solution is

presented in detail in chapters 5 and 6.

3.6 Realisation

3.6.1 Introduction

All NLG systems must have a realisation component as it is this stage which actually maps internal representations into surface NL. The most traditional NL realisers have three distinct entities [Hovy, 1988a] :-

- A set of grammar rules which govern how words can be put together.
- A lexicon comprising a collection of words together with their idiosyncratic features.
- A mechanism that produces text by accepting an input representation, building a syntactic tree structure on applying the rules of grammar to the input, inserting into the tree lexical entries that are accessed from the input representation and finally saying the words.

There are many variations to this simplistic approach depending, for example, on the input representation used, the control of the process (i.e description or message directed, see section 3.2) and the organisation of the grammar and lexicon. Some of the more widely used methods and their development are discussed in the following subsections.

3.6.2 Functional Unification

A unification grammar (invented by Kay [Kay, 1979]) characterises linguistic entities by collections of features called functional descriptions (FDs). In realisation, the input to the unification process is another FD which specifies the content of the required utterance. Two functional descriptors can be *unified* by an algorithm that is similar to set union [Appelt, 1983] [McKeown *et al.*, 1990]. The unification

```
FD1 = {article:{definite:yes}, head:{lex:'cat'}}

FD2 = {article:{lex:'the'}, modifier:{lex:'black'}}

unify(FD1,FD2)=  {article:{definite:yes, lex:'the'},
                  head:{lex:'cat'},
                  modifier:{lex:'black'}}
```

Figure 3.2: A simple example of unification of two FDs

of two FDs merges the features from both to produce a more specific FD, the *total* FD [Elhadad and Robin, 1992]. Figure 3.2 shows a simple example of unification (taken from [McKeown *et al.*, 1990]).

Those who advocate the unification approach argue for the following advantages:-

- The ability to encode functional information directly in the grammar. There has been a lot of linguistic research on the relation between functional information and syntactic construction which can be incorporated [McKeown and Swartout, 1988].
- Unification allows the input to grammar to be specified in simplified form [McKeown and Swartout, 1988].
- FDs allow the encoding of discourse features in the grammar [Appelt, 1983].
- The formalism relieves the high-level planning process of the need to consider low-level grammatical details.

The main disadvantage with the approach is that the process of unification is non-deterministic and therefore inefficient [McKeown and Swartout, 1988].

Systems which use a unification approach

- A functional unification grammar (FUG) was first used in generation for the final surface realisation stage of McKeown's TEXT system [McKeown, 1985].

- Appelt's TELEGRAM [Appelt, 1983][1] considered the problem of efficiency caused by non-determinism by closer integration of the planning and unification stages. Appelt modified the unification process by allowing the planner to be re-invoked at various choice points in the grammar in order to guide the process.

- McKeown *et al.* in their COMET system [McKeown *et al.*, 1990], expanded the idea of FUG to include a unification stage for lexical selection and for deciding when to explain information graphically or textually. These new extensions of the FU method termed *Functional Unification Formalisms* (FUF) allow it 'to be used more efficiently and to be used for other than purely syntactic tasks'. The grammar was extended, for example, to also include pragmatic features (such as mood and focus) that constrain the way the message should be expressed.

3.6.3 Systemic Grammars

Systemic functional linguistics [Halliday, 1985] divides language not just into syntax and semantics but on three functional lines of analysis [Meteer, 1993]:-

- ideational: the content of the utterance and the organisation of the speakers experience in terms of processes, things, qualities etc.

- interpersonal: The relation of the speaker and hearer.

- textual: The organisation and cohesion of text.

A grammar comprises a network of *systems* which represent a choice point where a feature must be selected from a set of alternatives. There are three lines of traversal through the network, one for each of the functional lines of reasoning listed above. The syntactic unit is specified cumulatively by all three lines.

[1] TELEGRAM evolved from Appelt's KAMP planner [Appelt, 1985] (section 3.5.2)

```
((GIVE1 / GIVE ;; GIVE1 is an instance of GIVE in the domain model
   :actor JOHN1
   :destination MARY1
   :object BOOK1
   :tense PRESENT
   :speechact ASSERTATION)
 (JOHN1 / PERSON
   :name John)
 (MARY1 / PERSON
   :name Mary)
 (BOOK1 / BOOK
   :determiner A
   :relations  ((C1 / COLORING
                  :domain BOOK1
                  :range BLUE )))
 (BLUE / COLOUR))
```

Figure 3.3: Input to NIGEL for "John gives a blue book to MARY"

There are three main generative systemic grammars in existence:- NIGEL [Mann, 1983b] [Matthiessen, 1991], the systemic grammar of the PENMAN project [Mann, 1983a], GENESYS [Fawcett and Tucker, 1990] [Fawcett, 1994] part of the COMMUNAL project and SLANG [Patten, 1988]. The rest of this section will present examples from the NIGEL grammar and PENMAN generator (GENESYS is of similar size to NIGEL whilst SLANG is much smaller).

PENMAN comprises a systemic grammar (NIGEL), an input specification language (SPL) and the general part of a knowledge base [McKeown and Elhadad, 1991]. Although originally a stand alone sentence generator, PENMAN has more recently been interfaced to planning systems, mainly those based on variants of RST [Hovy, 1991] (see section 3.7.3).

The SPL specification to the systemic grammar is an extremely rich input represented as a set of features. A special interpreter, however, allows the user to enter specifications in a simpler way by only having to partially specify features. See figure 3.3 for an example of a SPL specification [McKeown and Elhadad, 1991] for the sentence 'John gives a blue book to Mary'.

The values of the slots represent entities in the knowledge base. The NIGEL grammar queries the knowledge base to make decisions through functions called *inquiries* and *choosers*. The grammar comprises a number of *systems* representing the different choices possible for each type of syntactic constituent. The choice made by one system may form the input for another independent system. A sentence is produced by traversing the grammar systems starting with the least delicate choices (e.g. clause type or choice of passive/active voice). In each system a choice is made by invoking the chooser function associated with the system which in turn invokes one or more inquiry operators. Depending on the results of the choice and the selected system, other systems will be invoked until very delicate systems are reached (corresponding to lexical choice). Features can be preselected in the input which means the path from higher systems to that particular system can be avoided.

For example, figure 3.4 (from [McKeown and Elhadad, 1991]) shows the top level systems of the NIGEL grammar. This root system will decide the rank of the expression to generate. To make this decision, the inquiry function of the first branch will determine whether the input includes a speech act. If so, the next system *CLAUSE-CLASS* will be chosen and entered. The inquiry function of this system asks whether the speech act has a propositional parameter. If so, as in our example, the *CLAUSE-ELLIPSIS* system will be entered. The *CLAUSSETE* system, entered if the input has no propositional content will produce exclamations or greetings. The *CLAUSE-ELLIPSIS* system will determine if the clause is to be an answer to a question (in which case ellipsis can be produced) or not (the clause must be fully expanded).

Meteer [Meteer, 1993] states that the NIGEL grammar has one of the greatest competences of any linguistic component. She also argues that its grammar directed approach is very inefficient. She gives the example that when determining the features of a noun phrase, the system asks 'Is there a colour attribute' or 'Is there a size attribute?' etc. even when there are no such attributes for the head of the phrase in the demand expression. McKeown and Elhadad [McKeown and Swartout, 1988] in their case study of connective choice, argue that the strict ordering of

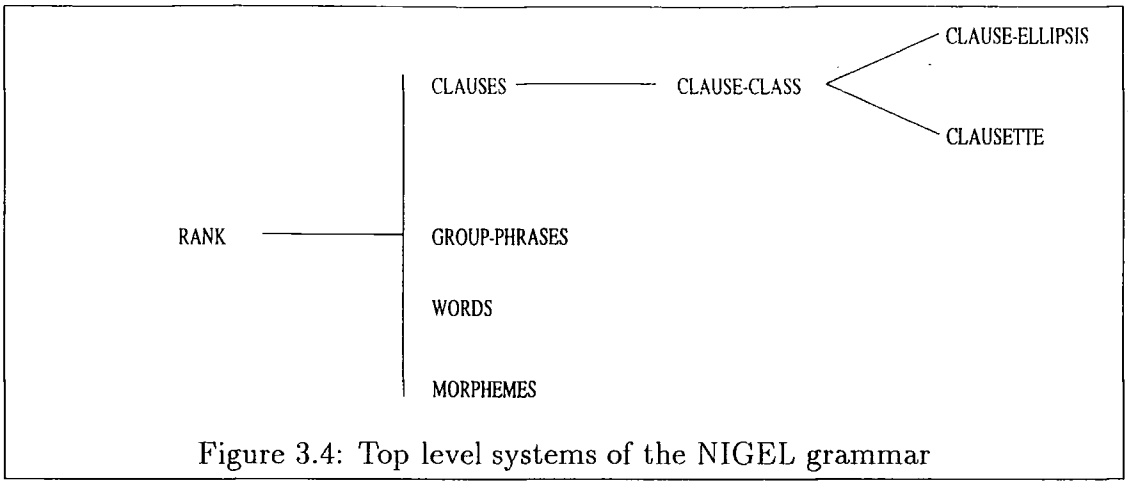


Figure 3.4: Top level systems of the NIGEL grammar

choices in the NIGEL grammar cannot allow 'low level' choices to effect higher ones.

3.6.4 MUMBLE

Introduction

McDonald [McDonald and Meteer, 1988] developed the final realisation component MUMBLE together with a specification language which facilitates interfacing MUMBLE to a wide range of underlying programs and planners. Meteer [Meteer, 1992] mentions five examples of different underlying programs in differing representations (including those that use the SPOKESMAN representation, see section 3.8.1).

The three main characteristics of MUMBLE's design are [McKeown and Swartout, 1988]:-

- MUMBLE is *Description-directed* (see section 3.4): MUMBLES's input is a fully specified message called an *input realisation specification.* Realisation is carried out by 'executing' the message as if it were a program in a special programming language (i.e by passing it through an interpreter). Thus the control rests in the message rather than in the knowledge of the grammar.

- MUMBLE relies on indelible processes. Once a decision has been made it

cannot be retracted. This is equivalent to Marcus's notion of determinism in parsing [Marcus, 1980] and means that there is no parallelism or backtracking in the generator.

- MUMBLE is psychologically motivated. McDonald hopes to model the human language process. Since decisions are indelible it models a speaker rather than a writer of human language. McDonald claims that his program will only produce errors only in instances where humans would make them.

As already mentioned in section 3.4, whilst McDonald claims that a description orientated approach leads to greater efficiency [McDonald *et al.*, 1987], other researchers believe that it affects the clarity and maintainability [McKeown and Swartout, 1988]. The indelibility of the process is also a source of efficiency. A negative side-effect to this is that there can be no bi-directionality allowing low level choices to affect higher level ones [McKeown and Elhadad, 1991]. However, because MUMBLE has been developed and extended over a long period of time, its coverage is extremely good [McKeown and Swartout, 1988].

An example of an input specification representing the noun phrase "53rd Mechanised Division" is shown in figure 3.5 (from [McDonald and Meteer, 1988]). McDonald claims that the specification language is easy to learn and has been used by others researchers not involved in MUMBLE's development. However, the example shows that even for simple noun phrases, a great deal of detail is required in the input specification. To alleviate this problem [McDonald and Meteer, 1988] describe the use of domain dependent templates which map messages from underlying applications to the MUMBLE input specification. The templates are abstractions of specifications which stipulate some of the terms in the specification and parameterise others.

MUMBLE's realisation process

MUMBLE's realisation process comprises three subprocesses each of which builds and refines the surface structure of the text to be generated.

```
#<bundle general-np
  : head #<kernel : realisation-function
                          np-common-noun
                  : arguments ("division") >
  : further-specifications
    ((:specification
          #<kernel : realisation-function adjective
                   : arguments ("53rd") >
      :attachment-function restrictive-modifier)
     (:specification
          #<kernel : realisation-function adjective
                   : arguments ("mechanised") >
      :attachment-function restrictive-modifier))
  : accessories (:number singular
                 :gender neuter
                 :person third
                 :determiner-policy no-determiner)>
```

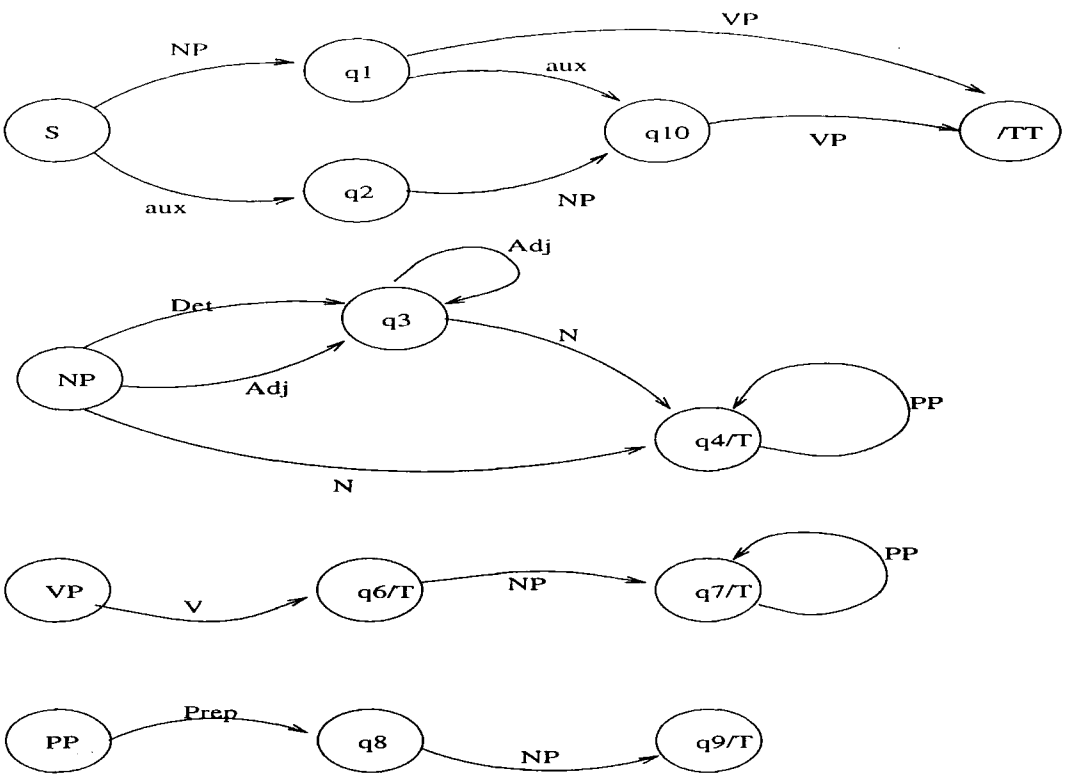
Figure 3.5: Realisation specification for "53rd Mechanised Division"

- **Attachment**: assigns plan units to positions within the tree representing surface structure. At any time in the process, this surface structure has *attachment points* to which new structures can be added. Initially the only attachment point is at the node dominating the first sentence but as the process progresses there may be a choice of attachment point.

- **Phase Structure Execution** (PSE). As soon as a plan has been attached, the PSE takes over. PSE performs a depth first traversal of the tree performing transformations or invoking constraints indicated by tree labels. If plan units are found in the tree then realisation is invoked to determine how they should be realised. If an attachment point is encountered then Attachment is called to determine if new structures should be attached.

- **Realisation**: Realisation selects appropriate words or phrases to 'realise' plan units. The realiser chooses between different syntactic choices according to various tests and may only partially realise a particular plan unit before re-invoking the PSE component.

MUMBLE's grammar is based on *Tree Adjoining Grammar* (TAG, [Joshi, 1987]).



NP → (DET) + (ADJ*) + N + (PP*)
PP → PREP + NP
S → NP + (AUX) + VP
S → AUX + NP + VP
VP → V + (NP) + (PP*)

Figure 3.6: An ATN and associated grammar

3.6.5 Augmented Transition Networks

Augmented transition networks (ATNs), originally used in NL analysis by Woods [Woods, 1970], have been used in generators which take semantic networks or graphs as input. The method has been used, for example, by Simmons and Slocum [Simmons and Slocum, 1972], Goldman [Goldman, 1975] (section 3.12.1), Shapiro [Shapiro, 1982] (section 3.12.3) and McKeown (section 3.7.2). Figure 3.6 shows a context free grammar and its ATN representation (from [Simmons and Slocum, 1972]). The grammar presented at the bottom of the figure is in the standard notation where labels on the left can be rewritten with the labels on the right, with parentheses indicating optionality and an asterisk representing one or more occurences. In the ATN representation, the nodes (or states) are circles and the

arcs are labelled with phrase names of other states (such as 'NP', 'VP' *etc.*) or the name of a terminal word class ('det','n', 'adj' *etc.*). States marked with '/T' show possible terminators of the net or subnet. By traversing the graph from node to node and jumping between subnets when an arc labelled with a state is encountered, all possible combinations of the grammar can be achieved. For example for the sentence 'Will the little red man break a waggon' the order of nodes/arcs traversed will be :- S, AUX (will), q2, NP, Det (the), q3, adj(little), q3, adj(red), q3, N (man), q4/t, q10, VP, V (break), q6, NP, Det (a), q3, N (waggon), q4/t, q7/t, /tt.

ATN networks were originally used to generate random sentences but were then used for message-directed generation by allowing them to be driven by a semantic representation of the desired utterance. In this case (see for example [Simmons and Slocum, 1972]) the arcs are labelled with names of relations in the semantic network and paths through the ATN are guided by the presence of such relations in the semantic input.

3.6.6 The Use of a Formative Lexicon

Another approach to realisation is to contain formative (i.e grammatical) information in the lexicon. Hovy, for example, argues [Hovy, 1988b] that instead of spreading formative rules between the grammar and the lexicon, it would be better if they were in the same place. In Hovy's PAULINE system all formative rules are associated with entities in the lexicon: each entry includes rules for how the particular word or phrase can be combined with others. PAULINE uses a phrasal lexicon (based on Becker's lexicon [Becker, 1975]) which comprises 'stock' phrases as well as individual words (e.g., 'to kick the bucket', 'Davy Jones' Locker'). The use of such a lexicon is especially suited when trying to create text in different styles as stock phrases often have a great deal of stylistic effect.

Other systems not only contain grammatical information in their lexicons but also contain information which shows how lexical entries are related to each other semantically. The use of a 'full' lexicon seems especially common in those sys-

tems which generate from a semantic network or conceptual graph input (see section 3.12). These systems proceed by finding appropriate lexical entries to express particular semantic concepts (usually verbs) in their input representation (see section 3.12). Once such a lexicon entry has been found, its formative rules are used to combine it with other concepts in the input.

A problem with some of the systems which adopt this approach is that of scale. If a system is limited to a particular domain and has a limited number of lexical entries then including formative and semantic information may be possible. When considering a system in a wider domain where there is a need for many more lexical items then the work and space required to encode each item may prove unrealistic and, due to repetition between similar classes of item, wasteful.

The problem of 'stock' phrases is a difficult one: phrases have fixed levels of rigidity (for example in some cases phrases may seem strange when transformed to the passive voice e.g., 'The bucket was kicked by John' whereas others maybe more flexible e.g., 'The excrement hit the ventilation system') and even phrases which may not be thought of as rigid always appear so in everyday use (e.g the unnaturalness of 'butter and bread', 'Juliet and Romeo' etc). As yet there is no 'deep' solution to the problem but, in the meantime, the use of fixed phrases in the lexicon can provide an effective and cheap solution.

3.6.7 Realisation: Notes on Relevance

Realisation is important in generation as it produces the output text: any generation system will need some sort of realiser. However, realisation depends heavily on input. Systems that take sentence length specifications of what has to be said will be different from those which take semantic network portions. As mentioned above (section 3.5), although the work described in this thesis uses a separated architecture, it places different responsibilities on the planning and realisation components.

Perhaps the most relevant work discussed above is the approach based on ATNs (section 3.6.5) as this has been used by semantic network systems. There may be

concern that this work was done early on and has been superceded by more recent approaches. However, this may be because the use of semantic network systems themselves declined for a period before regaining popularity (see section 3.12).

3.7 Planning

3.7.1 Early work

Early computational systems which generated multi-sentence text ignored the issue of text structure [Hovy, 1993]. They adopted such mechanisms as hill climbing (KDS [Mann and Moore, 1981], which described what to do in case of a fire alarm) or proceeded according to the organisation of the domain semantics (e.g TALESPIN a simple story generator [Meehan, 1977], PROTEUS which provided commentary for tic-tac-toe games [Davey, 1979], BLAH based on the hierarchical structure of tax-forms [Weiner, 1980]).

Some of the more modern generators are still domain restricted and often rely on domain dependent organisations to plan their discourse (e.g., Dale's EPICURE generates recipes [Dale, 1990], Mellish's house building planner [Mellish, 1988], Zukerman and Pearl's mathematical proof describer[Zukerman and Pearl, 1986], Horacek's financial advisor WEIBER [Horacek, 1990] and office space allocator OFFICE-PLANNER [Horacek, 1992], Cawsey's electronic circuit explainer EDGE [Cawsey, 1990]).

There are however two common methods used in a variety of systems for planning multi-sentence pieces of text: the use of *schemas* and *rhetorical structure theory* (RST) . These approaches will be considered in the following sections.

3.7.2 The Schema Approach

McKeown's TEXT system [McKeown, 1985] was one of the first generators that took multi-sentence discourse structure into account. It was developed to

SCHEMA

```
Constituency
Cause-effect*/Attributive*/
  {Depth-identification/Depth-attributive
    {Particular-illustration/evidence}
    {Comparison/analogy}}+
  {Amplification/Explanation/Attributive/Analogy}
```

'{}' indicates optionality,'/' indicates alternatives, '+' indicates that the item may appear 1-n times, and '*' indicates that the item may appear 0-n times.
EXAMPLE:

> "Steam and electric torpedoes. (1) Modern Torpedoes are of 2 general types. (2) Steam-propelled models have speeds of 27 to 45 knots and ranges of 4000 to 25,000 yds.(4,367-27,350 meters). (3) The electric powered models are similar (4) but do not leave the telltale wake created by the exhaust of a steam torpedo"

CLASSIFICATION OF EXAMPLE:

1. Constituency
2. Depth-identification; (depth-attributive)
3. Comparison
4. Depth-identification; (depth-attributive)

Figure 3.7: The constituency schema

provide paragraph-length responses to meta-level questions about the structure of a database containing information about military vehicles and weapons.

McKeown's approach was to analyse naturally occurring texts which describe objects and class each clause as one of a set of possible rhetorical predicates. By factoring out common patterns she was able to identify four patterns of discourse strategies which she called *schema*. Coherent paragraphs are generated by enforcing the correct nesting and filling of these schema. An example of a schema and the output generated from it by the TEXT system is shown in figure 3.7.

Other examples of systems which plan using a schema based strategy are Kukich's ANA [Kukich, 1988] (which summarises stock market moves), Cawsey's EDGE system [Cawsey, 1990] (a dialogue system used to explain how electrical circuits work, the plan content was built using domain dependent schemas), Horacek's WEIBER [Horacek, 1990] (a financial consultation dialogue system which planned utterances such as ASK, ASSERT and RECOMMEND using schemas) and Paris's generation work in the EES project [Paris, 1991] and the TAILOR system [Paris, 1993](which selected between different schemas according to a user model).

3.7.3 Rhetorical Structure Theory

Rhetorical structure theory (RST) was initially developed as a descriptive theory of text organisation [Mann and Thompson, 1987]. RST analysis is built up using instances of schema (not to be confused with the schema introduced in the previous section) which indicate how a particular unit (or *span*) of text structure is decomposed into other units. Each schema (see figure 3.8) consists of a *NUCLEUS* and zero or more *SATELLITES* whose function is to support the nucleus in some way. Satellites are linked to the nucleus by a *RELATION* which indicates how the satellite provides support. The schemata are unordered (satellite and nucleus can appear in any order in the text) and are recursive. A text span serving as the satellite of one schema may itself be decomposed into a nucleus and satellite of its own. By this recursive application of schema, paragraph length pieces of text can be described.

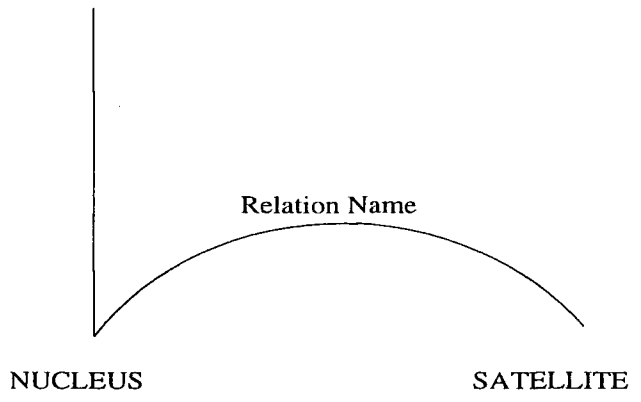


Figure 3.8: The RST schema

Each relation also has constraints on the nucleus, constraints on the satellite(s), constraints on the combination of nucleus and satellite(s) and an effect. Before a particular relation can be applied to a piece of text, these constraints must be satisfied.

For example, for the *Evidence* relation:-

- **Constraints on the Nucleus (the CLAIM)** : The reader possibly does not already believe the claim.
- **Constraints on the Satellite (the EVIDENCE)**: The reader either already believes the satellite or will find it credible.
- **Constraints on the combination of Nucleus and Satellite**: Comprehending the evidence will increase the reader's belief in the claim.
- **The Effect**: The reader's belief in the claim is increased.

Textual markers can suggest the application of a rhetorical relation (or be used to realise the relations during generation). For example the *Antithesis* relation can be marked by 'rather than', 'instead of', 'however', 'yet'. The *Evidence* relation by 'because', 'therefore' etc. Some relations however can only be indicated (or constructed) by syntax (for example the *Elaboration* relation).

An early criticism of the RST approach was that it was unclear how the originally descriptive theory could be used to constructively build text. How, for example, could a generator control the application of relations with such a wide number of possibilities of optionality and repetition for satellites? It was also clear that the relations provided (originally about 25) were not sufficient to encompass all types of text and that discrepancies occurred when different researchers tried to analyse the same piece of text.

Comparison between the schema and RST approaches

Mann and Moore [Mann and Thompson, 1987] claimed that schemas are nothing more than stereotypically appearing collections of RST relations, or conversely, RST relations are elemental building blocks of schemas.

Whilst schemas are easier to understand and to implement they do not contain enough information to dynamically reassemble the basic parts of a text into new paragraph types. RST relations, however, describe much smaller spans of text leading to more variation.

At this early stage of development, the way forward was to either generalise schemas and build a hierarchy of increasingly general schema types (the approach taken in COMET [McKeown *et al.*, 1990] for example) or to use RST to identify the basic building elements from which coherent paragraphs (and also schemas) are composed and develop a method of assembling them dynamically into paragraphs on demand (see following sections). In fact, in more modern planning systems a combination of both methods is being used (see section 3.7.4).

Using RST to plan discourse content

Using RST in a constructive process during generation is very different from using it in analysis. When generating a text, none of the information to be included is necessarily given. Instead an abstract specification of the utterance has to be provided via a discourse goal and the planner must chose what information to

include. This section will briefly describe some of the work which uses RST based planning formalisms.

- Hovy [Hovy, 1991] was one of the first researchers to apply the originally descriptive RST formalism to a constructive text structure planner. The planner operates after the application program (e.g., expert system) and before the sentence generator NIGEL[2] (see section 3.6.3). The planner assumes as input one or more communicative goals and a set of clause-sized predicates. It proceeds by recursively applying RST relations to units of the input and other RST relations in order to build a tree which represents the paragraph structure (non-terminals are RST operators, the leaves are the input predicates). In addition to constraints, nuclei and satellites contain growth points (collections of goals that suggest the inclusion of additional input material in the places they frequently occur in typical paragraphs). When a RST relation is found whose effects match one of the communicative goals, the planner searches for input predicates that match the requirements for the nucleus and satellite. If fulfilled, the planner considers growth points and continues recursively until either all the input entities have been incorporated into the tree or no goals can be achieved by the remaining input entities. Once constructed the tree is traversed in a depth first left to right manner adding each RST relation's characteristic cue words or phrases to the input entities and transmitting them to the sentence generator NIGEL.

- Moore, Swartout and Paris's Explainable Expert System (EES, [Paris, 1991] [Moore and Swartout, 1991])[3] uses an RST-based text planner to construct short explanatory dialogues of an expert system. They argue that expert systems need to explain their decisions as well as just state them but the problem is that there is often a lack of underlying domain knowledge. Expert systems have information about problem solving techniques in a particular

[2]The sentence generator based on the NIGEL grammar was originally called PENMAN. However as the project evolved, PENMAN was used for the whole of the project incorporating text planning and realisation.

[3]EES developed from the XPLAIN system [Swartout, 1983]

domain but do not have any information regarding why the problem solving rules can achieve the goals. Workers on EES 'tightened' the constraints so as to guide the search required to choose information to include in a response.

One of Moore and Swartout [Moore and Swartout, 1991] criticisms of Hovy's planner is that all the inputs are assumed to be present in advance. They argue that any process which can identify all and only the information to be included in a response would have to do much of the structurer's work anyway. In contrast to Hovy's planner, the EES system does not assume that all of the topics to be discussed are given to the planner as input. The planner is given a discourse (or intentional goal, e.g., persuade the hearer to do an act), the knowledge base of the expert system, the execution trace of the expert's system's reasoning, the user model and the dialogue history containing past utterances. The planner must plan an utterance to achieve the goal, choosing what to say from the knowledge base and organising it into a coherent structure.

- [Scott and de Souza, 1990] argue that the aim of text is to represent a message and this will only be done if the reader can derive this message. In order to do this they claim that the relations described by RST must be as accurate and unambiguous as possible. Some phrases can be used to link more than one RST relation (e.g., 'and') so Scott and de Souza try to use the strongest markers possible. The authors list other heuristics governing clause expression using RST operators so that clear text results. They are concerned with Portuguese and English.

- Fawcett and Davis in the COMMUNAL project [Fawcett and Davies, 1992] consider the problem of moving from monologue to dialogue generation. They have adapted RST to include an 'exchange mechanism' implemented using systemic principles. They argue that RST is the best available method of modelling text structure although they acknowledge its use comprises many unresolved issues.

- [Rösner and Stede, 1992] describe TECHDOC : an RST-based system which considers the automatic production of technical manuals (more specifically automobile maintenance instructions). They identify problems with describing texts using 'classical RST' and make their own (re)definitions. Because their domain leads to highly regular text they have combined RST with a schematic approach to define high level patterns.

- The IMAGENE (Instruction MAnual GENErator) [Vander-Linden *et al.*, 1992] is another system which has combined two different approaches. It has used a systemic network to translate process structures that need to be communicated into grammatically annotated RST relations. The constructed rhetorical relations are then passed to NIGEL (section 3.6.3) for realisation. The exact nature of the process structures that form the input is unclear but a great deal of work is left to a higher planning module which must provide this input. IMAGENE concentrates on the domain of describing the operation of cordless telephones.

- Other examples of very recent work based on RST are [Granville, 1994], [Delin *et al.*, 1994], and [Wanner, 1994].

3.7.4 Combination of Planning Resources

Hybrid planning systems are now being developed which consider more than one text planning representation and knowledge type. An example is the PENMAN system built jointly at USC/ISI and GMD-IPSI [Hovy *et al.*, 1992]. This system combines different knowledge source networks to build a declarative planner with each resource co-constraining the others. The resources are:-

- **Text type**. Because linguistic phenomena closely reflect the genre of text, the planner chooses the particular text type to be produced. The text type chosen will help pre-select or de-activate certain options in the other generation resources by, for example, constraining the communicative goals it entails, which discourse relations it favours and any grammatical features. The text

types are represented in a property-inheritance network ranging from very general genre (e.g., scientific) to very restricted text types (e.g., financial newspaper article, yearly public reports).

- **Communicative goal hierarchy**. Another property-inheritance network contains a taxonomy of communicative goals which describe the discourse purpose of the speaker. This hierarchy starts from very general goals (*INFORM, REQUEST, DESCRIBE, ORDER*) right down to specific goals to describe specific types of information for specific contexts (e.g., *describe-domestic-sales*).

- **Schema**. A lot of text exhibits stereotypical structures to which the schema approach is well suited. Stereotypical structures can define text at a clausal level (as in TEXT [McKeown and Swartout, 1988]) but equally well at a more general level (i.e defining the order of sub-topics). A schema, therefore, could be in the form of a list of detailed communicative goals. For example a descriptive financial report could comprise the schema of goals: *describe-total-sales-briefly, describe-total-sales-detail, describe-domestic-sales describe export-sales.*

- **Discourse structure relations** [Maier and Hovy, 1993]. This comprises three systemic networks of extended RST relations organised according to the three systemic functions of language (ideational, interpersonal and textual, see section 3.6.3). These networks constrain the relationships which hold between segments of text as well as co-constraining the other knowledge sources by, for example, preselecting theme patterns, posting communicative goals or specifying aspects of grammatical realisation.

- **Theme development**. The final resource is another systemic network describing potential theme developments and shifts of focus in order to signal the introduction of new topics and provide the relationships to previous topics.

3.7.5 Planning: Notes on Relevance

The role of the planning module is, again, dependent on assumptions about input. Planners which have to retrieve information from static databases will be different from those that simply have to organise a set of clause sized predicates, or more importantly with respect to this work, plan how to produce text from a semantic network.

The planner described in this work has different responsibilities from those described above. Furthermore, the plan-realiser has to undertake some planning responsibilities in the absence of planning instructions. In most of the work described above, the central concern of a planner is to choose the content of, and then organise, sentences. The job of the planner in the LOLITA system, however, will be to decide or even simply suggest how much information present in the semantic network should be expressed. The planner may or may not impose further constraints on exactly how this information should be realised (see chapter 5).

Semantic network or graph representations typically have very rich inherent information. This means that some of the ordering information which can be imposed by the use of methods such as RST is already explicit in the input. For example causal or temporal relationships between events will already be known.

3.8 The Generation Gap

Although modulising the planning and realising components may have advantages, it leads to the problem at the interface between components caused by the non-isomorphic nature of semantic and syntactic levels (see section 3.5). Even planning systems which reduce competence by assuming clause-sized inputs can have difficulties when moving from a semantic text plan to a syntactic realisation. This section will discuss how this 'generation gap' interfacing problem has been tackled when the decision to separate components has been taken. An alternative is to pursue an integrated approach as described in section 3.5.2.

There are two general approaches to filling the generation gap depending on whether the system is pipelined or interleaved. A pipelined system must make sure at each decision stage that its representation must be 'expressible' in surface language, whilst an interleaved system can relieve this constraint by passing information and control between the components or by using a backtracking mechanism.

3.8.1 SPOKESMAN

Meteer[Meteer, 1993] first coined the phrase 'generation gap' and it is her work on SPOKESMAN which has tackled the problem in most detail.

The SPOKESMAN planner and the MUMBLE realiser (section 3.6.4) on which it is built are psychologically motivated. Because of reasons of efficiency [McDonald *et al.*, 1987] both systems rely on indelible processes so that once a decision is made, it cannot be withdrawn and backtracking is not allowed: the architecture is purely *pipelined.* Because the SPOKESMAN planner is indelible, an expressibility requirement has to be imposed: at all points in the generation process, the generator must make sure that the representation of the utterance it has built is realisable in the language. Meeter states:-

> "In order to plan complex utterances and ensure they are expressible in language, the text planning process must know:-
>
> 1. what realisations are available to an element, that is what linguistic resources are available,
> 2. the constraints on the composition of the resources,
> 3. what has been committed to thus far in the utterance which constrains the choice of resource. "

Meteer's work meets these requirements and fills the 'generation gap' by introducing a new representational level, *Text Structure*, that both provides the choices necessary for the text planner to take advantage of the expressiveness of natural

language and prevents it from composing an utterance that is not expressible in the language.

SPOKESMAN starts with an input from an underlying application program and uses it to build the *Text Structure*. The *Text Structure* is a tree which captures relationships between the subparts (constituents) of the utterances. Each node represents a constituent and holds information such as what it expresses, how it relates to its parent and discourse information such as what entities have already been referred to. The input objects drive the building of the text structure using mappings associated with their types. The text structure also constraints its own construction by determining where additional information may be added. Figure 3.9 (from [Meteer, 1993]) shows the *Text Structure* for the utterance 'Karen likes watching movies on Sundays'. The following information is shown in the figure:-

- **Constituency**: The nodes in the Text Structure reflect the constituency of the utterance.
- **Structural relations among constituents**: Each node is marked with its relation to parent (top label) and children (bottom label of parent node).
- **Semantic Category**. The central node labels (in bold) show the lexical head (italics) and the semantic category the constituent expresses.

Once the text structure has been built, it is traversed, top down and left to right, to build the *linguistic specification* (see section 3.6.4) for the MUMBLE realiser. Recent work using and extending the SPOKESMAN formalism is described in [Meteer, 1994] and [Panaget, 1994]

3.8.2 The PENMAN Upper Model

The PENMAN system uses the systemic grammar NIGEL(see section 3.6.3). NIGEL is traversed for every 'rank' to be realised from high level clauses right down to low level words and phrases. At the end of the traversal enough features would

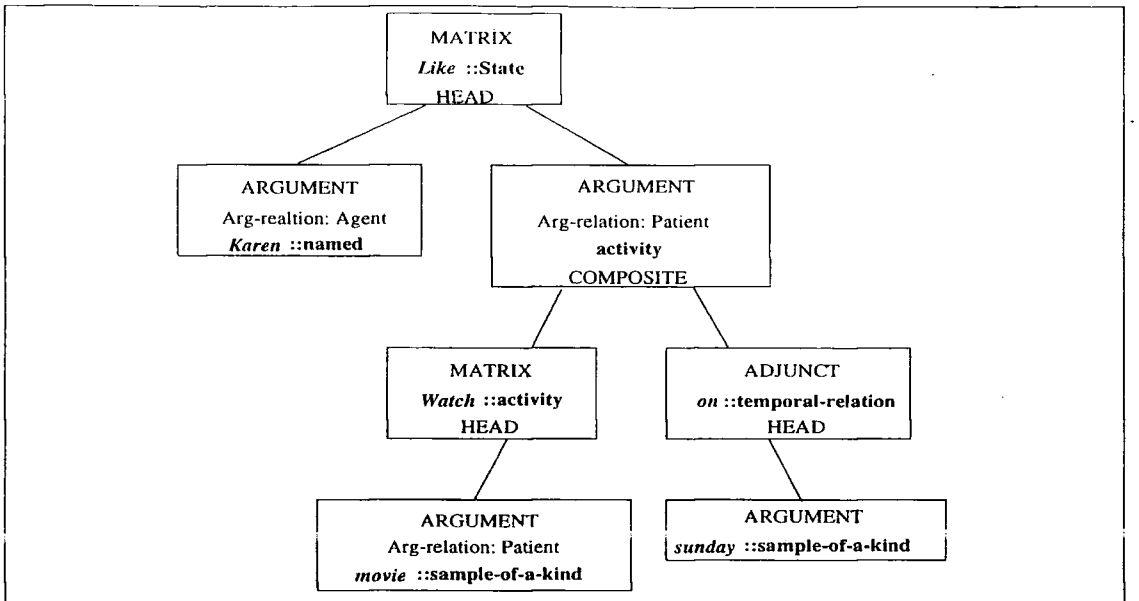


Figure 3.9: Text Structure for "Karen likes watching movies on Sundays"

have been collected to be able to look up an appropriate word or syntactic structure. The PENMAN system overcomes the generation gap and the assumption that there will be a suitable word or structure available by utilising the Upper Model [Bateman *et al.*, 1990]. The Upper Model is a 'concept hierarchy encoding basic semantic distinctions that manifest themselves on the linguistic surface'. The Upper Model, which is accessed to make both grammatical and lexical decisions, is a semantic ontology that classifies the properties of words in a particular language (for example a classification of verbs and restrictions on their arguments). Underlying applications (or planners) have to assign each object that they want to 'talk about' to a category in the ontology. As each of these ontological categories have carefully defined properties relating to how they can be generated, the generation gap problem is avoided. However, it is not always clear to which category an entity may belong. For example the domain concept **computer** may be assigned to the Upper Model concept of **conscious-being** or **non-conscious-thing** depending on whether computers are to be talked about as active conscious entities or not. Thus the ontology is designed to be used not to categorise things as to what they are in the 'real world' but to how they are talked about. Fawcett also describes a similar ontology used in GENESYS [Fawcett, 1994].

```
REQUEST :<TEMPERATURE>

- 'weather' : MEDIUM
   (MAKES-EXPLICIT <WEATHER>)
   (INDIRECTLY-SUGGESTS <TEMPERATURE>)
- 'temperature' : VERY-HIGH
   (MAKES-EXPLICIT <TEMPERATURE>)
- 'it' : VERY-HIGH
   (MAKES-EXPLICIT <TEMPERATURE>)
   (CONCISE-CONSTRUCTION)
```

Figure 3.10: Example of annotated linguistic options in IGEN

3.8.3 IGEN

Rubinoff's IGEN generator [Rubinoff, 1992] is an interleaved system in which the linguistic realisation component provides feedback to the planner. This feedback is in the form of annotations which give abstract information about the effects of choosing between various syntactic or lexical items. The planner can use these annotations to evaluate different options and indicate which it prefers. The preferred options can then be assembled subject to syntactic constraints to produce English output. Rubinoff says the ranking of possible linguistic choices can be affected by pragmatic and stylistic features. The annotations given are **makes-explicit**, **makes-implicit** and **indirectly-suggests**. Figure 3.10 shows an example of the annotated linguistic structures returned to the planner after a request to express the concept **TEMPERATURE**. Both 'temperature' and 'it' get the highest rating and the preference will depend on the verbosity parameter. It is not clear how the linguistic component chooses possible structures to express a concept but as it can return inappropriate items (such as "temperature" for the concept **SEPT-26-1992**) the suggestion is that this process is exhaustive and therefore inefficient and infeasible for large-scale systems.

3.8.4 WEIBER

Horacek's WEIBER system [Horacek, 1990] appears to be a pipelined system. He describes a 'new' processing level responsible for finding a suitable selection of predicates at the lexical level (lexemes and grammatical structures) that express the information content associated with predicates in the conceptual level (concepts and roles). He says that this transition can involve considerable restructuring because meanings of lexemes and grammatical functions may be related unevenly to the information content associated with the conceptual predicates. To bridge this 'generation gap' he uses *ZOOM Schemata* to map elements between representation levels. *ZOOM* schemata can be categorised further : *MICRO ZOOM* schema relate one-to-one mapping whilst other types (e.g., *MIX* and *MACRO*) map items which are related in more complicated ways. The choice of one out of the possible selection of valid schemata can lead to the production of variation (see section 3.10).

3.8.5 Crossing the 'Generation Gap' with FUGs

Elhadad and Robin [Elhadad and Robin, 1992] discovered the generation gap problem when they tried to extend the functional unification grammar to include content realisation and lexical choice (see section 3.6.2). To overcome the problem they had to alter the top-down control regime by adding two control tools to the formalism. **bk-class** is used for dependency-directed backtracking and **external** allows a FUG to cooperate with external constraint sources during unification.

3.8.6 Generation Gap: Notes on Relevance

Due to the differing distribution of responsibility adopted in the solution presented in this work, the generation gap problem manifests itself differently and in a less critical way. If the planner does not provide detailed instructions or even provides conflicting instructions, the plan-realiser will be able to make decisions more traditionally assigned to the planner itself.

These aspects will be detailed in the solution chapters (chapters 5 and 6).

3.9 Lexicalisation

Lexicalisation is the process of selecting words and phrases to represent concepts in an internal semantic representation. This process of linking representation to words would appear to be the most basic requirement of a NL generator but has been the subject of much recent discussion (for example the panel discussion in [Horacek and Zock, 1993], [Stede, 1994], [McKeown and Elhadad, 1991], [McDonald, 1991], [Fawcett, 1994], [McDonald and Busa, 1994], [Viegas and Bouillon, 1994]). This is because many early or domain dependent systems have assumed a one to one mapping between the concepts in their representation and words or phrases in their lexicon. Natural language itself has often been used in the design of internal representations; a concept is defined only if a word exists in the particular NL used. In such systems lexicalisation is made trivial as all concepts are linked directly to a suitable word or expression. In general however, the relationship between the 'grain size' of concepts may well be more elaborate and an isomorphism between lexical and conceptual structure cannot be assumed [Novak, 1993].

Although lexicalisation has to be considered in all generation systems the problem is especially important when generating from semantic networks or graphs. The following subsections will provide a brief overview of how the problem of differing word and concept granularity has been tackled by other researchers.

3.9.1 The use of Discrimination Networks

NLP systems in the 1970s were based on representations which comprised a few highly abstract concepts (for example Conceptual Dependency Theory or CDT [Schank, 1975]). It was using these systems that *discrimination networks* (or *d-nets*) were first employed to relate these large 'grain size' concepts to surface words. Goldman's BABEL generator [Goldman, 1975] (see section 3.12.1) was the first to

employ *d-nets* to generate from the semantic primitives of CDT. Each concept was associated with a *d-net* which consisted of a decision tree with words on the leaves and path selection procedures attached to the nodes. These selection procedures repeatedly queried the context of the concept to be expressed until a suitable lexical entry was found. The classic example given is for the primitive conceptual act INGEST representing the activity of some substance entering the bodies of animate beings. The selection procedures made queries as to the properties of the substance being ingested and differentiated between verbs such as *eat*, *drink*, *inhale* etc. Many other later systems adopted the d-net approach albeit in a sometimes disguised manner [Stede, 1994]. For example :-

- Horacek's LOQUI [Horacek, 1987] uses 'epistemological primitives' that are organised in an inheritance hierarchy. Horacek assumes that in general 'the nodes of the network are attached to nouns reflecting exactly the semantics of one node'. When this is not the case, a discrimination network is attached to the node.

- VIE-GEN [Buchberger and Horacek, 1988] again attaches d-nets to primitive concepts arranged in a taxonomic knowledge base in the style of KL-ONE [Brachman and Schmolze, 1985]. As well as action and object concepts, relations are enriched with d-nets. The inheritance mechanism of KL-ONE means that d-nets can be inherited from superordinate concepts.

- COMET [McKeown *et al.*, 1990] (see section 3.6.2) again uses a (disguised) d-net approach. Before a semantic content specification is passed to the unification grammar proper, it is enriched with lexical information. This is also based on unification but a provision is made which leaves this formalism to call LISP procedures for making finer word choices. For example, the concept C-TURN (turning a knob) can be lexicalised as *set* or *turn* depending on whether the knob has discrete positions.

- Pustejovsky and Nirenburg [Pustejovsky and Nirenburg, 1987] again use a system which discriminates concepts using d-nets. e.g., STOL, a subtype of

furniture produces items like *table, desk, coffee table* etc by asking questions about the location and height of the item in question.

- Hovy's PAULINE [Hovy, 1988b] (see section 3.12.1) adopts the same strategy but makes distinctions on pragmatic as well as semantic grounds.

- DIOGENES [Nirenburg and Nirenburg, 1988] (see section 3.5.2) represents lexical items using a *frame* which defines the concept an item expresses as well as restrictions on certain roles of that concept. For example, the word *boy* has its concept slot filled by 'person' and additional slots which prescribe 'sex' to be 'male' and 'age' to be between 2 and 15 etc. The result is similar to that produced by a *d-net* except a *d-net* will always comes up with an answer. DIOGENES uses a numerical 'meaning matching metric' to get the best match. This 'nearest neighbour classification' restores the robustness.

3.9.2 The use of Taxonomic Knowledge Bases

As systems based on such abstract concepts became less popular, so too did the discrimination net approach to lexicalisation. McDonald [McDonald, 1991] says :-

> "Applications with this style of representation are increasingly in the minority (having been replaced by designs where the comparable generalisations are captured in class hierarchies of taxonomic lattices)".

A taxonomic knowledge base (KB) organises objects (corresponding to nouns) and verbs (actions) in *is-a* hierarchies where subordinate concepts inherit the properties of their super-ordinates. Stede claims that "[these hierarchies have been] established as the de facto standard in knowledge representation [and] the idea of fully decomposing semantic definitions into minimal entities has been dispensed with."

However, even when using such knowledge bases, the problem of choosing concept granularity still remains. The interface between the taxonomic KB and its associated lexicon is not necessarily a straightforward mapping. Approaches to

this problem have been solved in a variety of ways. The methods adopted by those researchers who start generation from a semantic or conceptual graph will be examined in more detail in section 3.12.

3.9.3 Lexicalisation: Notes on Relevance

The problem of the non-isomorphism between concepts and words has recently been addressed. However, most researchers still adopt a representation where the concept *grain size* is larger than that of words. For example Stede [Stede, 1994] claims:-

> "The task of the word concept link is to mediate between the granularities of the KB and the lexicon: the problem is trivial when they are identical, but often there are good reasons to make *FINER* distinctions in the lexicon than they are required for reasoning purposes in the KB."

This is in direct contrast to the philosophical approach to representation adopted in the LOLITA project. Here the belief is that language is motivated by concepts and so, in general, all words which manifest themselves in a particular language will have a unique concept (except in the case of exact synonyms) whereas concepts will not necessarily have an associated word. Thus the grain size in the LOLITA SemNet representation is smaller for concepts than for words (this is discussed in more detail in section 1.5.3).

This is clearly exemplified when more than one NL is being considered (i.e in translation systems). There are often cases when a concept in one language can be associated directly to a word whilst no equivalent word exists in other languages. This problem has been confronted by Sondheimer *et al.* [Sondheimer *et al.*, 1989] (although they restrict themselves to objects and nouns and not actions and verbs). They investigate several cases of 'unnamed' concept. For example if an unnamed concept is distinguished from its superordinate one by an additional role, the lexical item from the superordinate concept is used as the head term for the noun group and restrictive modifiers are added to express the role.

This approach is similar to that undertaken in this work which will be discussed more fully in the solution chapter (chapter 5).

3.10 Creating Variation

Human speakers produce a wide variation of utterance and an aim of NLG is to be able to mimic this ability. As well as being more natural, this will enable an utterance to be tailored according to the communication goal, the intended listener, and a host of pragmatic constraints (e.g., situation, familiarity, time constraints, stylistic constraints). This section will provide a brief overview of the ways variation can be manifested in NLG generation systems. It will be concluded that the use of a rich semantic network type input to the generator is particularly appropriate for allowing variation. When systems of this type are examined in more detail in section 3.12, the individual ways in which they tackle the problem of variation will be examined in more detail.

Variation can be produced throughout the generation process. A list of possible areas and the relevance to this project is now presented:-

- Planning. The most obvious way to create variation in an utterance is by changing the way content is selected and ordered. Because the planning problem is still a long way from being solved there is relatively little work which considers how to produce variation at this stage. What is more, planning in the traditional sense is not a concern of the work presented here. Aspects of planning in those systems which are to be detailed in the second part of this chapter will be described.

- Lexicalisation. The previous section outlined the lexicalisation problem concerning linking concepts to words or phrases in NL. There are many words and phrases which have similar meanings but the selection of one rather than the other can create a wide variety of stylistic effects. As already mentioned, lexicalisation is a key procedure when generating from semantic network rep-

resentations and methods adopted to create variation at this stage will be outlined for individual systems in the second part of this chapter.

- Use of World knowledge. Systems which have a rich source of world knowledge may use this information to create variation. Having information about the different actors and roles in events, for example, enable such events to be talked about from different angles. The systems to be described in the second part of this chapter typically contain rich information which can be exploited in this way.

- Choice of starting point. This method of variation is particularly appropriate to semantic network generation. When a piece of semantic network is realised there is an obvious source of variation according from where in the input representation the utterance starts.

- Choice of grammar. Even when the majority of the content of an utterance has been chosen there are still a variety of ways in which it can be grammatically realised. Choices, for example, between passive and active, dative and non-dative sentences or the use of de-lexical verbs can lead to a variety of utterances. The systems which are particularly good at being able to produce a variety of grammatical variations are those which are based on linguistic theories. This is especially the case of the large systemic grammars such as NIGEL and GENESYS (see section 3.6.3).

3.10.1 Controlling Variation

There is a view that enabling a system to generate sentences in a variety of ways without a process which is able to control and choose between these variations is dangerous. It could be argued that adding such variation capabilities in generation modules only means that the problem of control in higher level components (e.g., high level planning) is made more difficult.

There is however the opposing view that, even without proper control, adding the ability of variation is a good step forward.

3.11 Other Areas of NLG

3.11.1 Revision

There is some confusion as to the meaning of a revision stage during the generation process. Some use it to mean revisions over a final piece of surface text while others use it when describing revisions over internal representations during initial utterance generation. While the latter type of revision (termed *optimisations* by Meeter [Meteer, 1993]) is wide spread (especially in interleaved systems) work on final revisions has received less attention. Exceptions are the work by Yazdani [Yazdani, 1987] who argues that the inclusion of a reviewing stage may simplify the process of generation as mistakes or irregularities may be resolved later and Cline[Cline, 1994] who includes a final revision stage in his SNePs based generator (see section 3.12.3). The former type of revision (or optimisation) is particularly useful when there is a rich semantic input to the generator such as semantic networks or graphs. Transformations can be applied to such networks before final realisation. This type of revision will be considered in this work (chapter 6). Conversely, the second type of revision is usually required when the input is not rich. By the time surface language has been achieved, there is little information available which can allow for further revisions. This kind of revision can be useful for those systems concerned with ordering clause-sized predicates when operations such as aggregation for sentences with the same subject can be performed. Final surface text revision will not be considered in this project

3.11.2 Connectionism

A quite different approach to NLG (in particular planning content in NLG) is the connectionism approach an example of which is described by [Hasida *et al.*, 1987].

Hasida attempts to generate abstracts from an internal network representation of knowledge by using a cognitive connectionist paradigm where

"processing in the human brain is accounted for in terms of signal prop-

agation in a network which reflects the topology of neural connections"

Nodes in the network representation are given an explicit degree of importance and the numbers are 'thrown at' the network so that when a node is activated, all of its linked nodes are also activated (to a varying degree). When a network has stabilised then the nodes with the top activation values are chosen for inclusion in the abstract. The method, it is argued, means that important concepts (nodes), concepts linked to important concepts and concepts that are mentioned often are likely to be those chosen.

The connectionist approach will not be considered in this work.

3.12 Generation from a Semantic Network or Graph input

The type of input has an important bearing on the design of of a generation system: systems which take similar input often have similar designs and must perform similar tasks. This section will discuss generation systems which take similar input to the LOLITA generator: rich semantic information represented using a network or graph (compared to, for example predicate calculus). The different inputs which will be considered are Conceptual Dependency Theory, Conceptual graphs, SNePs, MTT and others.

A meta-point worthy of discussion is the gap between early work and more recent work in this area with a obvious lull in between (for example [Simmons and Slocum, 1972] to [Nicolov *et al.*, 1995]). A suggested reason for this lull is that in the early 70s semantic network based systems were seen as the promising way forward but, due to inadequacies of other areas (not least hardware capabilities), only small scale 'toy' systems were built and progress was slow. The blame for this rate of progress was put on the semantic network representations and the approach was more or less abandoned in favour of other approaches (e.g predicate calculus systems). More recently, people have realised that it was not necessarily the representation that was to blame and research on such systems has increased once more.

3.12.1 Generation from CDT

Introduction to CDT

Schank's *Conceptual Dependency Theory* (CDT) was one of the earliest attempts at a representation which aimed to capture the content of NL sentences (with the view to perform reasoning activities such as translation or summarisation). In CDT, meaning representations are composed of semantic primitives. Actions for example, are decomposed into a small set of primitive acts such as INGEST (a substance

entering the body of an animate being), ATRANS (movement of a physical object) and MTRANS (movement of a non-physical object such as information in speech acts).

One or two dozen (depending on the version) of these primitives were employed which although allowed reasoning algorithms to perform well with the hardware available at the time, are too restricted for a serious large scale system. In fact, Schank later moved away from using such a restricted set of primitives and defined higher level primitives such as PROMISE and THREATEN.

However, Schank's effort was pioneering and it is worth considering the attempts at generation from his representation.

BABEL

Goldman's BABEL system [Goldman, 1975] produces English sentences and paraphrases of sentences from Schanks' CDT. There are obvious problems with starting from this representation due to the small number of primitives relations allowed in the theory. However, this helps the paraphrase problem as there are often many verbs applicable to a particular primitive relation.

The first stage in BABEL is to find the main verb which can be used to express a portion of CDT representation. This is achieved using a series of discrimination networks (see section 3.9.1). These are binary trees whose nodes comprise predicates which determine which child path to follow. At the leaves of the trees are pointers to a *concexion* entry which are used to build a syntactic network of the utterance. The discrimination network predicates can :-

- test for particular patterns/values in the input. For example, 'is the act in the RESULT of a conceptualisation ATRANS ?'
- consult the 'memory model' to see what semantic class a concept has. For example, milk is a fluid.
- consult the memory model to see what concepts have passed before. For

example if 'John gave Mary a book' is in the memory model and now 'Mary is giving John the same book', then the verb 'return' may be used.

- consult the memory model to find the likely outcome to the mental states of actors of a concept. The choice between 'threaten' and 'promise' may be made in this way. Note that these queries are answered by a human user and not inferred automatically by the system.

The concexion is used to build a syntactic network from the conceptualisation by tracing round the network finding values for various syntactic slots. This requires a different concexion entry for each word sense. Syntax networks are realised into surface NL using a ATN similar to that described by Slocum and Simmons [Simmons and Slocum, 1972] (section 3.6.5).

BABEL can produce paraphrases by returning more than one concexion for a particular input conceptualisation. The leaves of the discrimination net, as well as containing a pointer to a concexion, have pointers to other nodes in the network which can be followed to find more concexions. For example the concexions 'advise1', 'suggest1' or 'tell1' might be found leading to paraphrases:-

'John advised Mary to read the book'
'John suggested to Mary she would like to read the book'
'John told Mary she would like to read the book'

PAULINE

Hovy's PAULINE (Planning And Uttering Language In Natural Environments) [Hovy, 1988b] is the most sophisticated generator which takes CDT representation as input. Hovy's aim was to generate various text forms from the same semantic input according to parameters which describe the pragmatic setting. His list of features form the most complex system of parameters considered and concern conversational settings (e.g., time, tone, conditions), individual characteristics of the speaker (e.g., knowledge of topic, opinion of topic) and hearer (emotional state,

- **formality** (*highfalutin, normal, colloquial*)
- **simplicity** (*simple, normal, complex*)
- **timidity** (*timid, normal, reckless*)
- **partiality** (*impartial, implicit, explicit*)
- **detail** (*details only, interpretations, both*)
- **haste** (*pressured, unplanned, somewhat planned, planned*)
- **force** (*forceful, normal, quiet*)
- **floridity** (*dry, neutral, quiet*)
- **colour** (*facts only, with colour*)
- **personal reference** (*much ,normal, none*)
- **open-mindedness** (*narrow-minded, open-minded*)
- **respect** (*arrogant, respectful, neutral, cajoling*)

Figure 3.11: Hovy's rhetorical goals and values

language ability), their goals (e.g., affect hearer's opinions or emotional state) and the relationship between them (e.g., depth of acquaintance, relative social status). Once these factors have been determined (an assumed input to PAULINE) they are used to constrain the text generated. However, Hovy claims that these factors are too general to be used directly in the determination of an utterance:-

> "Since the interpersonal goals are too far removed from the syntactic concerns of language to provide such rules [to constrain generation], **there must exist a number of intermediate goals expressly designed for this purpose**"

These intermediate goals called *rhetorical goals* and their associated values are given in figure 3.11. The values are determined on the basis of supplied values of the conversation parameters. For example the rhetorical goal **Formality** is set according to the rule:-

1. set **RG:formality** to:-

- *colloquial* when the **depth of acquaintance** is marked *friends*, or when the **relative social status** is marked *equals* in an *atmosphere (tone)* marked *informal.*
- *normal* when the **depth of acquaintance** is marked *acquaintances*
- *highfalutin* when the **depth of acquaintance** is marked *strangers*

2. then reset **RG:formality** one step toward *colloquial* if **desired effect on interpersonal distance** is marked *close* or if **tone** is marked *informal.*
3. or reset **RG:formality** one step toward *highfalutin* if **desired effect on interpersonal distance** is marked *distant* or if **tone** is marked *formal.*
4. and invert the value of **RG:formality** if the **desired effect on hearer's emotion toward speaker** is marked *dislike* (since inappropriate formality is often taken as an insult), or if the **desired effect on hearer's emotional state** is marked *angry.*

Once set, the rhetorical goals affect future generation decisions in all subsequent stages of the generation including content delimitation, sentence structuring and final realisation. Continuing the **RG:formality** example, PAULINE can apply the following strategies:-

- Topic inclusion : For formal text make long sentences by selecting information that contains causal, temporal or other relations to other sentence topics.
- Topic organisation: For formal text make complex sentences, select options that are sub-ordinated in relative clauses, that co-join two or more sentences or that are juxtaposed into relations and multi-predicate enhancer and mitigator phrases. For more informal text select options without the above characteristics in order to build short simple sentences.
- Sentence organisation: For formal text make sentences appear 'weighty' by including many adverbial clauses at the beginnings and ends of sentences (rather than in the middle), build parallel clauses within sentences, use the

HIGHFALUTIN:

"In early April, a shanty-town - named Winnie Mandela city - was erected by several students on Beinecke Plaza, so that Yale University would divest from companies doing business in South Africa. Later, at 5:30 AM on April 14, the shanty town was destroyed by officials; also at that time, the police arrested 76 students. Several local politicians and faculty members expressed criticism of Yale's action. Finally, Yale gave the students permission to reassemble the shanty town there and, concurrently, the university announced that a commission would go to South Africa in July to investigate the system of Apartheid."

INFORMAL:

" Students put a shanty town, Winnie Mandela City, up on Beinecke Plaza in early April. The students wanted Yale university to pull their money out of companies doing business in South Africa. Officials tore it down at 5:30 on April 14, and police arrested 76 students. Several local politicians and faculty members criticised the action. Later, Yale allowed the students to put it up there again. The university said that a commission would go to South Africa in July to study the system of Apartheid."

Figure 3.12: Example of a formal and informal text produced by PAULINE

passive voice, use complex tenses such as the perfect tense, avoid ellipsis even when it is grammatical. For more informal text make simple clauses by selecting at most one adverbial clause (placed toward the end of the predicate), use the active voice, avoid complex tenses and ellipse words and clauses where this is grammatically allowed.

- Clause organisation: For formal text make weighty formal clauses by including adjective and adjectival clauses in noun groups, double nouns in noun phrases (e.g., 'Government and Emperor', 'statements and expressions'), include many adverbs and stress words in predicates, use long formal phrases, pronominalise where possible, do not refer directly to the interlocutors or the setting. For informal text make simple clauses by selecting at most one adjective in noun groups, use short, simple phrases, use verbs and adverbs instead of their nominal forms and refer to interlocutors and the setting directly.

- Word Choice: For formal text select formal phrases and words, avoid doubtful grammar, popular idioms, slang and contractions (e.g., 'man' rather than 'guy', 'cannot' rather than 'can't'). For informal text use informal phrases and words by selecting simple common words, using popular idioms, slang and contractions where possible.

Using heuristics such as those described above, Hovy gives examples of texts with the same informational content but differing styles. Two pieces of text exemplifying the difference between highfalutin and informal text are shown in figure 3.12. (Note that although the examples are of paragraph length it appears that PAULINE is not responsible for such a level of planning. It is probable that sentence length portions of CDT input are passed to PAULINE in turn.)

Criticism of PAULINE

Although the example shows that PAULINE is capable of producing a range of impressive output there are some doubts as to the depth of heuristics used [McKeown and Swartout, 1988] [Mykowiecka, 1991b]. For example, McKeown claims :-

> " Hovy attempts to show the effects of (rhetorical goal) influences on far too many decisions in the generation process. As a consequence he is unable to do a thorough analysis of the effect of these parameters on choice. He provides few satisfactory rationales for how his input parameters influence generation decisions. As a result, his values for RGs and his rules for setting them sometimes appear arbitrary"

However, the weakest aspect of PAULINE is the fact that its input is CDT which is based on a small set of primitives. Hovy did not pursue this work on PAULINE after his move from Yale to ISI, where he was concerned with higher level planning components (section 3.7.4) for the PENMAN project.

3.12.2 Generation from Conceptual Graphs

Introduction to Conceptual Graphs

Conceptual Structures(CS) (or Graphs,CG) is a modern knowledge representation developed primarily by Sowa [Sowa, 1984]. The driving motivation in the development of CG theory was exactly to represent natural language semantics:

> "Logical Form should be tailored to linguistic form in order to avoid unnecessary complications in the grammar" [Sowa, 1984]

There has been an enormous dispersal of CG research including work on neural networks, database systems and software development as well as NLP. [Nagle *et al.*, 1992] represents a recent summary of some of the best of this research. This section, after giving a brief introduction to CG theory, will concern work on generation starting from the CG representation.

A CG is a finite, connected, bipartite graph with nodes that are either concepts or conceptual relations that relate two concepts. Nodes are connected by arcs: a concept can only have arcs to conceptual relations and conceptual relations can only have arcs to concepts. Examples of conceptual graphs are given in figures 3.13 and 3.15.

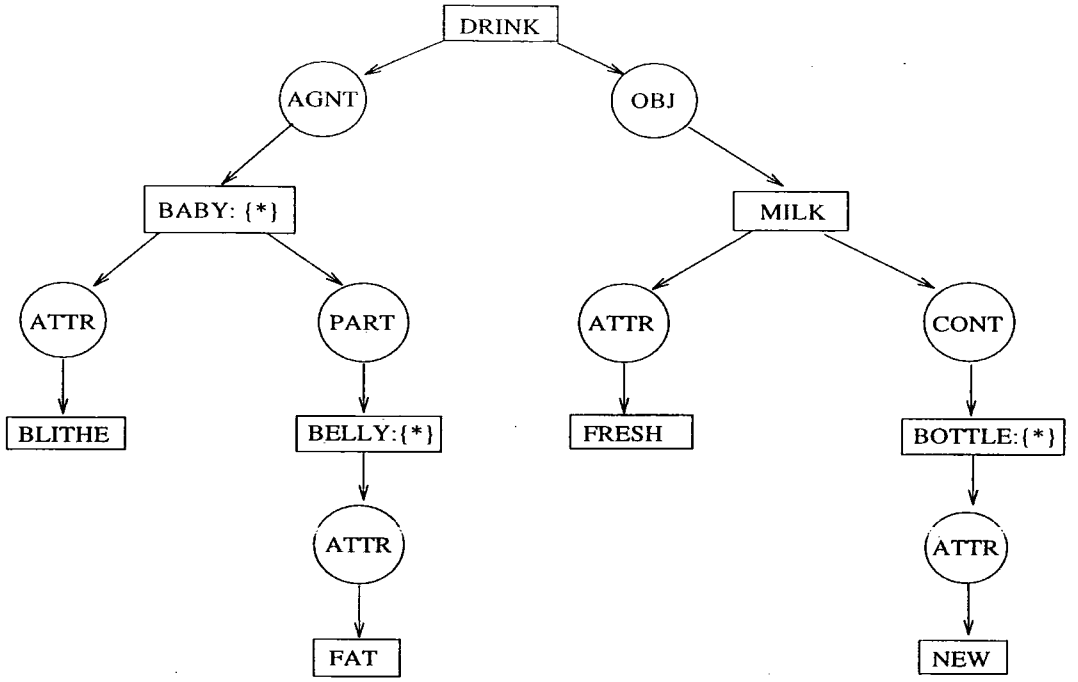


Figure 3.13: Example utterance graph input

Although there some similarities between the CG representation and the SemNet representation used in LOLITA, there are also many differences. A detailed comparison is, however, beyond the scope of this thesis.

Sowa's Generation

Sowa's generation chapter in his book [Sowa, 1984] and related paper [Sowa, 1983] concerns the mapping of conceptual graphs to natural language. He defines the sequence of nodes and arcs that must be traversed in mapping a graph to a sentence as the *utterance path.* He explains that for complex graphs, the utterance path may visit a concept more than once and depending on the type of language to be generated, words should be generated at the first, last or some intermediate visit to a node. In the case of an in-order language such as English, words have to be produced at intermediate visits.

Figure 3.13 shows an example conceptual graph from which Sowa generates. By starting from different nodes in the graph and following different utterance paths the following sentences could result:-

S (*type*(◯)= AGNT) ⟶
NP (move AGNT→ □; mark AGNT→ □ traversed;
case := NOMINATIVE;
person := *person(referent(□))*;
number := *count(referent(□)))*;
VP (move AGNT← □; voice := ACTIVE;
tense := tense of S; mode := mode of S;
person:= person of NP; number := number of NP).

Figure 3.14: Example APSG grammar rule

```
Blithe babies with fat bellies drink fresh milk in new bottles
Fresh milk in new bottles is drunk by blithe babies with fat bellies
Blithe babies that drink fresh milk in new bottles have fat bellies
Drinking fresh milk in new bottles is done by blithe babies with fat
bellies
etc...
```

However, not all word orders are possible. The utterance path can visit each node a number of times and a concept can only be uttered at one of those visits. To constrain the utterance path, Sowa presents six universal grammar rules which are claimed to be language independent. To further constrain the utterance path to allow for correct positioning of sentence constituents, these rules are supplemented by language dependent rules. These rules decide which arc to follow when there is a choice and also insert function words and inflections. These grammar rules are encoded in an *Augmented Phrase Structure Grammar (APSG)*. APSG is an extension of a context-free grammar augmented with conditions to be tested (the left hand side of the rule) and actions to be performed (right hand side). The rules are applied in a top-down goal directed manner. Figure 3.14 shows an example portion of this grammar which breaks a sentence (S) into a noun phrase (NP) and a verb phrase (VP). In the notation, □ refers to the current concept node of the conceptual graph and ◯ refers to the current conceptual relation.

There are three shortcomings to the generation approach described by Sowa.

Firstly, it is assumed that there is a one to one correspondence between concepts

in the conceptual graph and words. When the grammar rules decide that a node should be realised at a particular visit, the name of that node is used (with the necessary morphological variation). As discussed in section 3.9 this is a very strong assumption to make.

Secondly, the 'blithe babies' example, although leading to a long sentence, is produced from conceptual input which is a simple (at most binary) tree. In reality, for more complex sentences, the graph will be cyclic and the control of the utterance path will be more complicated. Sowa [Sowa, 1984] does say (page 234) that "if the graph has cycles, a concept that is reachable by two or more different paths will only be uttered once with all of its qualifiers. If syntactic rules would also express the concept at a visit reached by a different path, they must instead generate an anaphoric expression". Sowa does not however expand on this point.

Finally, Sowa's examples assume that a portion of the conceptual graph can be realised in one sentence. The realisation process has to try to cover all the input by finding an *utterance path* that visits every node. Again Sowa recognises the problem without giving a detailed solution: 'If a graph is complicated, rules of inference may break it into multiple simpler graphs before expressing it in a sentence'.

Nogier and Zock's work

Nogier and Zock [Nogier and Zock, 1992] describe a method of generation from conceptual graphs which is used in the information retrieval system Kalipsos.

The input to their generator is a CG called the *utterance graph* which is realised by incremental matching with lexical entries that are also represented using CGs (*word definition graphs*).

Words in the lexicon have three kinds of information: a word definition graph representing their meaning, a base form (lexeme) and their possible syntactic structure. Examples of lexical entries for the verbs 'to move', 'to run' and 'to drive' are given in figure 3.16. The generation procedure is as follows (using the utterance

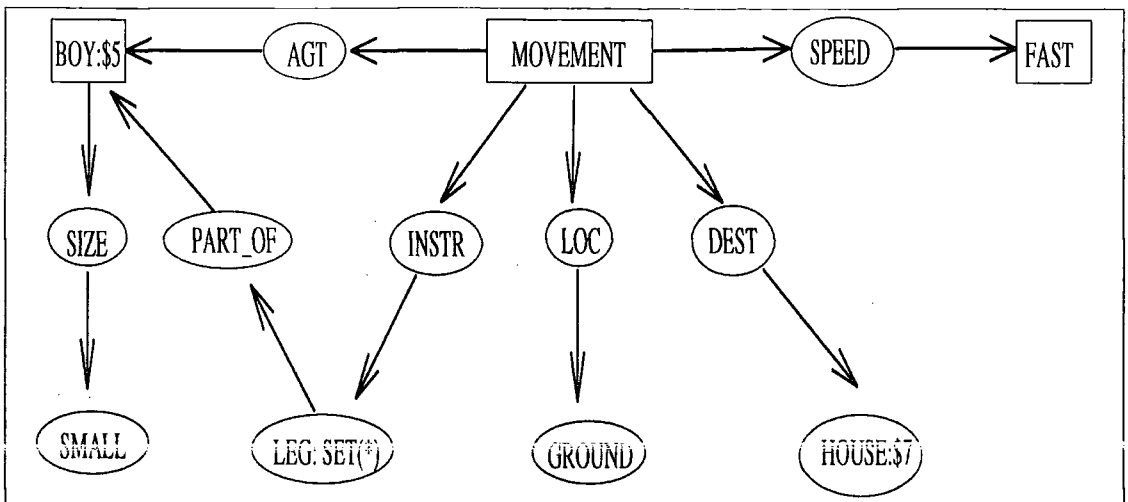


Figure 3.15: Example utterance graph input

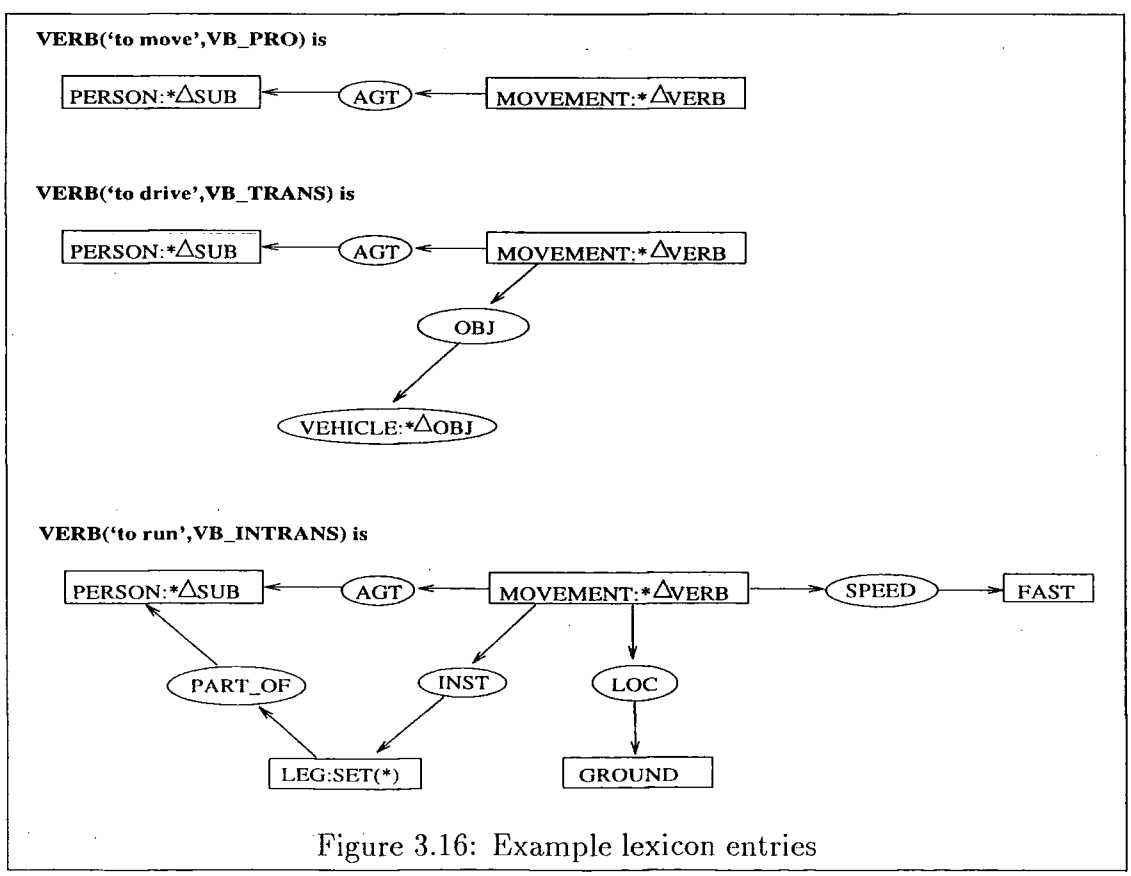


Figure 3.16: Example lexicon entries

graph shown in figure 3.15 as an example):-

Step 1: Preselection

Combinational explosion (due to the fact that the same conceptual input can be expressed in many ways) is avoided by making a rough choice for the most central concept of the conceptualisation. In this case the task is to find a word which conveys the central idea of MOVEMENT. As this type of concept typically maps onto a verb the task is now to find a verb which expresses movement. Thus lexical items that contain the concept [MOVEMENT: ΔVERB[4]] are selected, namely 'to walk' 'to drive' 'to move' 'to swim' and 'to run'.

Step 2: Choice of candidates by pattern matching

In order to eliminate all but one of the candidate lexical entries a pattern matching operation is carried out. The conceptual word definitions graphs are projected onto the utterance graph and the system chooses those definitions which are subgraphs of the input. In this example the entries for 'to walk', 'to move' and 'to run' will be successful and all others will fail (to drive will fail, for example, because the relation OBJ cannot be unified with LEG and INSTR).

Step 3: Selection of best candidate

Since there may be more than one lexical entry that passes the projection filter, the next stage is to chose the lexical item that best fits the utterance graph. A *correlation factor* is calculated for each candidate word definition which is a measure of the word's *appropriateness* or *accuracy* in expressing the input. The entry with the highest correlation factor is chosen as the root lexical node. In this case 'to run' is chosen as 'to walk' does not convey a speed and 'to move' does not convey information about the instrument(legs) or location(ground).

Step 4: Replacement of the conceptual structure with syntactic structure

Since the selected lexical item contains information about the syntactic struc-

[4]ΔVERB means the concept is to be expressed as a verb

ture which can carry it, the conceptual graph can now be partially substituted with syntactic information. In this case the verb 'to run' is expressed as an intransitive verb which associates the AGT(agent) of the conceptual action to the SUB(subject) of the verb. At this stage the representation is hybrid as it contains both conceptual and syntactic information. The lexical selection process will have to be repeated for the remaining concepts in the utterance graph. In this example the syntactic graph can be realised as the sentence 'the small boy ran to a house'.

Nogier and Zock argue that their method allows for a very natural way for creating paraphrases. By selecting words with varying values of correlation factor, different paraphrases will result. Because different words are associated with different syntactic structures these paraphrases will also differ in this respect.

Rather than trying to find a single path that visits all of the nodes of the input (as Sowa), the approach tries to match pieces of the input to lexical items incrementally (an analogy used is that of completing a jigsaw). This approach is called *incremental consumption.* However the *utterance path* and *incremental consumption* approaches are similar with respect to their goal of having to cover all of the semantic input.

Criticism of this approach

Although the authors describe a very 'neat' solution to the problem of lexical choice and generation there are a number of weak points. Although a system has been implemented which can generate sentences in real time, they only consider a lexicon of about 100 graphs. When this lexicon is expanded to a realistic size (which in itself will be a big task since each lexical entry has to be very rich) the search for entries which match onto input structures will become very inefficient. This problem is compounded as words which can belong to more than one syntactic category will have to have multiple lexical entries (for example if a verb allows both for active and for passive voice, then the lexicon must contain a graph for each of these forms).

The examples given in the paper start from a conceptual input that can be expressed as a simple sentence. What is more, the generation mechanism is verb

driven. This places a high level of responsibility on any high level planner which must be incorporated if more complicated sentences or multi-sentence utterances are to be produced.

Finally, the approach aims to cover all the semantic input. In the LOLITA generator (see chapter 5), the realiser takes the whole SemNet representation as input and such a complete coverage approach is therefore impossible.

Other CG generation work

Work at IBM, Rome, [Velardi *et al.*, 1988] concerns an Italian NL system used to analyse a database of press-agency releases on finances and economics. The system uses a conceptual graph representation from which short sentence length replies to queries are generated (e.g., 'a plan is the theme of an assembly of the delegates'). Concepts in the CG are related to syntactic structures using a table of relations between them. The system can produce passive or active sentences (controlled by the user) but the generation examples are simple and the grammar is small.

Dogru and Slagle [Dogru and Slagle, 1992] describe another system that generates English expressions for conceptual graphs. They say that this tool is useful as complex graphs can look ambiguous and English translations can be used to verify that they are correct. The authors claim that 'the implemented system can take arbitrarily complex graphs as input and produce a corresponding English translation'. However they do not give examples of complex graph translations as 'space limitations do not permit more complex graphs that show how various aspects combined can result in high quality translations'. What is more the English translations they do give seem rather unnatural as they are a very literal translation of the graphs (examples:- 'Persons John and Jane live in the city Minneapolis', 'There exists at least 2 persons among John, Jane and Jill', 'At least 100 students from the group cultural-diversity'). This system may well be useful for interpreting and checking CGs for development or reasoning purposes but its performance as a generator seems limited.

Rijn [van Rijn, 1992] describes a system which generates from a special kind of CG called a conceptual dependency graph (not to be confused with Schank's conceptual dependency). These graphs contain low-level primitives for which there is not necessarily a direct link to a word in the output NL. Again, only simple input graphs are assumed.

Harrius [Harrius, 1992] describes work (which seems in an early stage of development) to interface an RST based planner (see section 3.7.3) with the CG representation.

3.12.3 Generation from SNePS

Introduction to SNePS representation

The SNePs (Semantic Network Processing System) [Shapiro, 1979] [Shapiro and the SNePS Implementation Group, 1993] project has a long history beginning in the early 70's. The project, which aims to develop a computational cognitive agent, is based on a semantic network knowledge representation and reasoning system that "allows one to design, implement, and use specific knowledge-representation constructs and that easily supports nested beliefs meta-knowledge, and meta reasoning". Although the project has a long history and there is an abundance of published work, there is a surprising lack of information specifically concerned with NLG. This section will examine two exceptions: NLG work by the leader of the project, Shapiro, and work by Cline.

Shapiro's generation

Shapiro [Shapiro, 1982] describes a generalisation of the ATN formalism (see section 3.6.5) that supplies consistent semantics for both parsing and generating grammars. This allows an ATN grammar to be constructed so that the 'parse' of a NL question is the NL statement that answers it. The goal of the generation part of this process is, given a node, to express the concept represented by that node as a

user : Young Lucy saw a saw
SNePS: I understand that young Lucy saw a saw.
user : who saw a saw
SNePS: Young Lucy saw a saw
user : Lucy is sweet
SNePS: I understand that young Lucy is sweet
user : what was seen by Lucy
SNePS: A saw was seen by sweet young Lucy

Figure 3.17: Example SNePS interaction

NL surface string.

Figure 3.17 shows an example interaction using SNePSUL-the SNePS User Language and figure 3.18 shows the SNePS semantic network that is built as a result of this interaction.

KALOS

Cline's KALOS system [Cline, 1994], based on SNePS, generates descriptions of the M68000 processor. The system first generates very simple sentences and then a revision component looks at the surface output together with the representations that led to this output and passes revision suggestions back to both the deep generator (for conceptual revisions) and the surface realiser (for stylistic revisions). Cline argues for a uniform knowledge base. His system illustrates the flexibility of the SNePS representation as it is used for:-

- Domain Knowledge : a detailed taxonomy of relations in the domain of microprocessor operation. This knowledge largely comprises taxonomic links such as 'sub-classes' and 'part-of' links although it also contains some knowledge about the operation of instructions.

- Deep Generation : Kalos uses a simple schema approach (see section 3.7.2) but represents the schema (description, identification, constituency) rules for filling the slots and instantiated schema using SNePS.

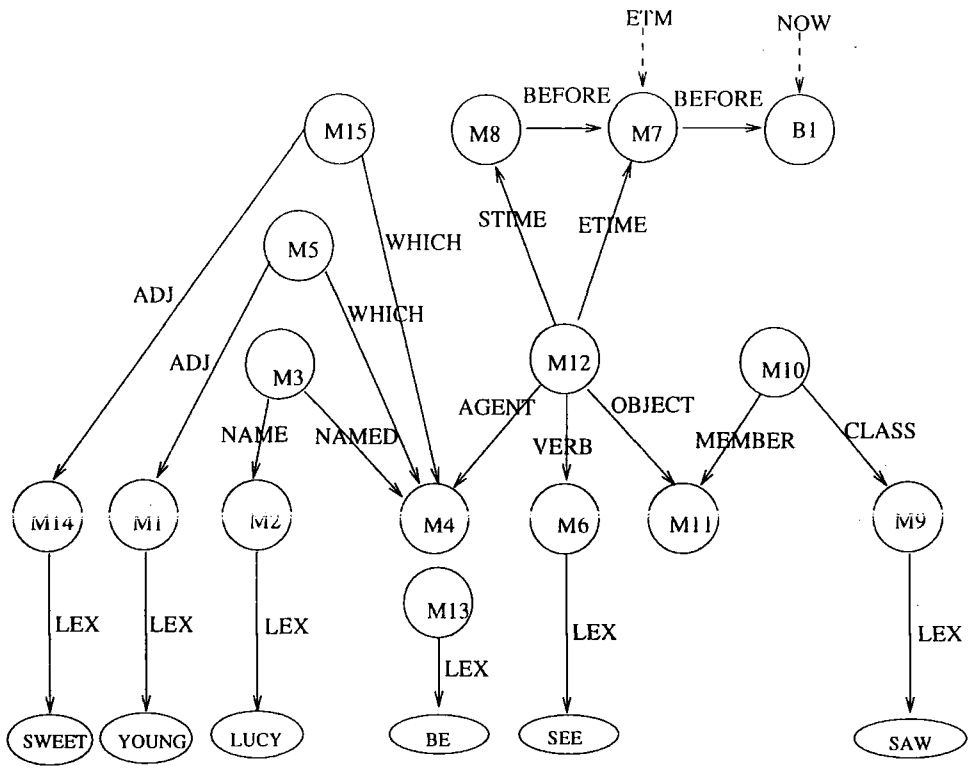


Figure 3.18: SNePS semantic network built by the example interaction

- Surface Generation: The system uses a unification based approach (see section 3.6.2) with the grammar rules being represented in SNePS.

- Conceptual Revisor: Knowledge encoded in SNePS is used to look for conceptual defects in the output and pass back suggestions to the deep generator (i.e the schema). Example conceptual revisions are removal of redundant information, application of domain preferred words and phrases, proper ordering of attributes and handling of inordinately long lists.

- Stylistic Revisor: Knowledge encoded in SNePS is used to look for stylistic defects in the output and pass back suggestions to the surface generator (i.e the grammar). Example stylistic revisions are to suggest the use of anaphora, compound phrases and sentences, use preferred words and phrases, suggest thematic progression and other cohesive constructions.

The initial output from the surface generator is very simple and comprises a long list of short sentences which just seem to describe the taxonomy of the domain

```
the M68000 is a microprocessor
the M68000 supports memory-mapped I/O
the M68000 address bus is an address bus
...
the M68000 address registers can be divided
           into pure address registers
the M68000 address registers can be divided
           into special address registers
...
etc (about 60 sentences)
```

the 16-bit M68000 microprocessor has an address space size of 16 megabytes and supports memory mapped I/O. It has 9 32 bit address registers and 8 32 bit data registers ... etc

Figure 3.19: Example output from Kalos before and after revision

(mainly 'is-a' links). After revision the text is much better but is still very simple (see figure 3.20).

The strengths of Cline's work are the use of the SNePS representation and the theory and implementation of a revision stage. However, as noted in section 3.9, 'after realisation' revision is usually best suited to those systems that organise clause sized chunks and this is clearly the case in the Kalos system. The domain and application of the Kalos system is very restricted.

3.12.4 Generation from MTM

Introduction to MTM

Mel'čuk's Meaning Text Theory (MTT) [Mel'čuk and Polguère, 1970] is a well-founded linguistic theory which has been used in generation.

A Meaning Text Model (MTM) describes the bidirectional mapping between linguistic meanings and texts which carry those meanings. Seven levels of descriptions are used [Iordanskaja *et al.*, 1991]:

1. semantic representations (SemR)

2. deep syntactic representations (DSyntR)
3. surface syntactic representations (SSyntR)
4. deep morphological representations (DMorphR)
5. surface morphological representations (SMorphR)
6. deep phonetic representations (DPhonR)
7. surface phonetic representations (SPhonR)

However, the MTM is meant to model a variety of languages as well as phonetic levels. For generating languages such as English, the 'interesting' representation levels are the first three [Iordanskaja *et al.*, 1991].

In the MTM, the generation process starts from a semantic network representation (see figure 3.20 for an example). Unlike other approaches which start from such a representation, nodes and arc labels in a MTM semantic network correspond directly to lexemes.

Realisation in MTM proceeds by performing a series of transformations that restructure the network and allow for the production of various paraphrases.

The MTM lexicon

The rich lexical information in a MTM is presented in an 'Explanatory Combinatorial Dictionary' (ECD) which aims to cover all possible linguistic knowledge governing the use of words in texts. Lexical information is split into three 'zones':-

- **The Semantic Zone**: specifies a semantic network which defines the meaning of the lexical entry in terms of the next simpler word meaning elements (semantemes).
- **The Syntactic Zone**: specifies the entry's syntactic class, syntactic features (to identify special constructions containing the lexeme) and government patterns which show how the semantic cases of the entry are represented in the two syntactic levels (DSyntR and SSyntR).

- **The Lexical Combinatorics Zone**: specifies related lexemes as the values of lexical functions. These functions can compute such things as synonyms, super-ordinate terms and converse terms.

The following section will describe how the information encoded in each lexical entry is used to generate NL and produce paraphrases.

GOSSIP

The GOSSIP (Generation of Operating System Summaries in Prolog) system described by [Iordanskaja *et al.*, 1991] is based on the MT model. It is a sentence length generator which takes as input sentence length portions of semantic network together with a communicative structure which marks the theme and rheme of the sentence to be produced. Generation comprises four transformation stages each of which can be a source of variation:-

- semantic network reductions,
- choice of root lexical node for the deep syntactic dependency tree,
- deep syntactic paraphrasing using lexical functions,
- alternative renderings of deep syntactic structure as surface syntactic structures.

These variation sources will be considered in turn:-

Incremental semantic network reductions

The semantic network input represents the literal meaning of sentences. Because the semantic elements on the nodes of the network are usually simple lexemes of English, these networks could be 'verbalised' directly. However, this direct verbalisation would lead to long and clumsy sentences which would not normally be acceptable (sentence 1 is the verbalisation of the network shown in figure 3.20).

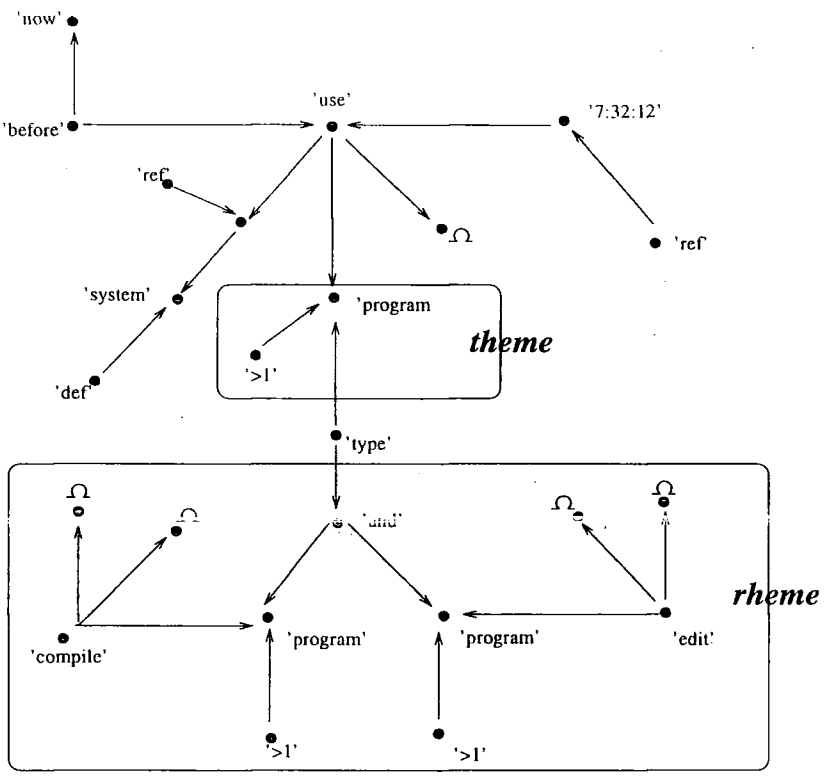


Figure 3.20: MTM SemR for sentence 1

(1) The referred-to user(s) of the system used (before now) during the referred-to period of 7 hours, 32 minutes and 12 seconds, more than one program of a type such that someone compiles something with these programs and someone edits something with these programs.

Instead, the semantic network can be incrementally reduced leading to shorter verbalisations. Subnetworks which are the meaning definitions of semantically more complex terms, can be replaced by the single node representing this term. In each case the defining semantic subnetwork is found for each lexeme in the *semantic zone* of the ECD lexicon (see above).

For example, the semantic subnetwork which leads to the verbalisation 'programs such that someone compiles something with these programs' in sentence (1) can be replaced by the node 'compilers'. Similarly the node 'editors' can be substituted. With a further substitution involving the lexeme type, the resulting semantic network would be verbalised as sentence (2).

(2) The aforementioned users of the system used compilers and editors during ..

This reduction process will eventually terminate but each reduction stage will lead to a different paraphrase when the network is verbalised. The authors note that sometimes it would be more appropriate to use longer paraphrases (i.e networks with fewer reductions) in order to meet certain stylistic constraints.

As well as variations caused by the amount of reduction, another source of paraphrase arises when there is more than one way of reducing a network. The authors say that this can lead to paraphrastic variant sentences with quite different lexicalisations. For example when a verb can incorporate some but not all of its manner modifier there may be two alternative reductions, each leaving part of the manner modification to another structure. An example of this is given in sentences 3a and b below.

(3a) Fred limped across the road quickly.

(3b) Fred hurried across the road with a limp.

Root lexical node choice

The second stage of the MTM generation process, and another source of paraphrase, is during the transition form the semantic representation (SemR) to the deep syntactic representation (DSyntR). This step involves choosing between different *entry nodes* of the semantic network which will determine the root verbal lexeme of the DSyntR. A semantic node can be an entry node [Iordanskaja *et al.*, 1991] if it is a predicate node and either

(i) is the dominant node of the theme or rheme,

(ii) directly governs the dominant node of the theme or

(iii) connects the dominant nodes of the theme and rheme.

In figure 3.17 the entry node could be the predicate 'use' which fulfils condition (ii), in which case the DSyntR would have the root lexeme 'use' and result in a

sentence such as 4a. However, a second possibility for an entry node would be 'type' which connects the dominant nodes of the theme and the rheme (condition iii) and could lead to sentence 4b (because the chosen predicate is non-verbal, the copula verb 'be' is the root).

(4a) System users ran compilers and editors during this time.

(4b) The types of programs that users ran during this time were compilers and editors.

The communicative structure (i.e the theme and rheme regions) marked on the semantic input can impose constraints on the *entry node* chosen. If, for example, there are two competing lexemes which could be chosen (for example send/receive) then the preferred lexicalisation would be the one whose first actant is in the theme region of the network.

The collection of sentences below (5abc) gives another example of variation created by choosing differing entry nodes. All three sentences are good paraphrases of each other however, while 5a and 5b have the same theme/rheme structure, this is inverted in 5c.

5(a) The user who ran editors is called Martin.

5(b) The name of the user who ran editors is Martin.

5(c) Martin is the name of the user who ran editors.

Deep structure paraphrasing

The ECD lexicon relates semantically related lexical items using a set of lexical functions (LFs)[5]. These functional expressions can be used to represent structural correspondences such as those between 6a,6b and 7a,7b below.

(6a) Martin used Emacs a lot.

(6b) Martin made heavy use of Emacs.

[5]Boyer and LaPalme [Boyer and Lapalme, 1985] also describe transformations using lexical functions.

(7a) Martin edited a large text file.

(7a) Martin did some editing of large text file.

In both example 6 and 7, the simple verb form is paraphrasable by a complex de-lexical verb phrase. Both illustrate the use of the lexical function *Oper1* below:-

$$X_{verb} ==> Oper_1(X) - -II- - > S_0(X)$$

This rule states that a verb node in DSyntR can be replaced by the two node dependency tree specified in the right side of the rule. The second syntactic argument of the new verb is the action nominal (S_0) of the old verb (X). The particular lexical values of the items $Oper_1(X)$ and $S_0(X)$ are contained in the ECD of the language. In example 6, $Oper_1(use) = make$, $S_0(use) = use$ and in 7, $Oper_1(edit) = do$, $S_0(edit) = editing$.

The transition from 6a to 6b also requires a change in the degree adverbial using another lexical function *Magn*. In the DSyntR representation of 6a the item 'a lot' is not explicit but labelled by the node 'Magn' attached to the verb node 'use' (the X verb). When the *Oper1* function is applied, all the dependents of the X node are carried over as dependents of the node $S_0(X)$. After the transition, therefore, the lexical function *Magn* will apply to the new verb 'make' and correspond to the value 'heavy'.

Surface realisation

The final source of paraphrase in the MT model is during surface syntactic realisation. Variations can be achieved by using a different grammatical ordering or, if applicable, by choosing between two or more possible values of the lexical functions described in the previous section.

MTM: Summary

The direct use of lexemes for nodes and arcs in the semantic input means that the granularity of concepts is the same as the granularity of words. The MTM takes the most basic words of the language as the set of primitives. The theory is thus language driven and assumes that these basic words are sufficient to define other words (see section 3.9).

Although some of the associations provided by lexical functions are useful to handle surface relationships between words (e.g., by providing exact synonyms, or giving an appropriate de-lexical construct) they seem to take this notion to an extreme by defining semantic relationships in the same way. For example taxonomic relations such as *part-of* are represented as in the same way as is information such as the actors and sub-events of verb entries (actor(to shoot) = gunman, marksman. Prepare(to shoot)= [to] charge (the gun)).

Another limitation to the approach is the assumption that the generator receives sentence sized portions of semantic representation as input. This puts more work on the planning component and is similar to the restriction to clause size predicates imposed by other systems.

The solution presented in this work (see chapter 5) depends on the plan-realiser taking the whole semantic network as input. The approach therefore of incrementally simplifying the network until a suitable utterance results, would be highly inefficient in this case.

3.12.5 Other Similar Work

KING

Jacobs' KING (Knowledge Intensive Natural language Generator)[Jacobs, 1987] is a sentence level, unification based generator which was developed from Jacobs' PHRED (a Unix consultancy system) [Jacobs, 1985].

Jacobs advocates a knowledge intensive approach which:-

> 'addresses the language generation task from within the broader context of the representation and application of conceptual and linguistic knowledge'

Jacobs makes the point that many other generation systems operate with a great deal of linguistic knowledge specific to the generation problem. They serve to illustrate the importance of the need for specialised constructs and the ability to use specialised knowledge in generation. He argues, however, that some of these systems tend to use knowledge which is too specialised leading to the coding of redundant knowledge so that the specification of linguistic choice is more convenient. Certain knowledge about the world and knowledge about language is treated instead as knowledge about generation. He claims:-

> "the problem is not only the lack of a parsimonious representation; more importantly, the representations fail to support the interaction of general and specialised knowledge required for a broadly applicable system. .....
>
> This difficulty proves to be a major handicap in building versatile generation systems; A key element is to facilitate the exploitation of generalisations while still providing for specialised uses.'

Jacobs, therefore, builds a lot of information into his representation so that these generalisations can be exploited. The representation based on ACE [Jacobs and Rau, 1985], defines events in a hierarchy with each event inheriting properties for those above and the roles of the events defined from a number of 'views'. Jacobs provides the example for the *commercial transfer* event. In this example :-

> "(figure 3.21) illustrates that the *commercial-transaction* is a complex event that consists of a transfer of *merchandise* and a transfer of *tender*. The *merchant* receives the *tender* from the *customer*, and the

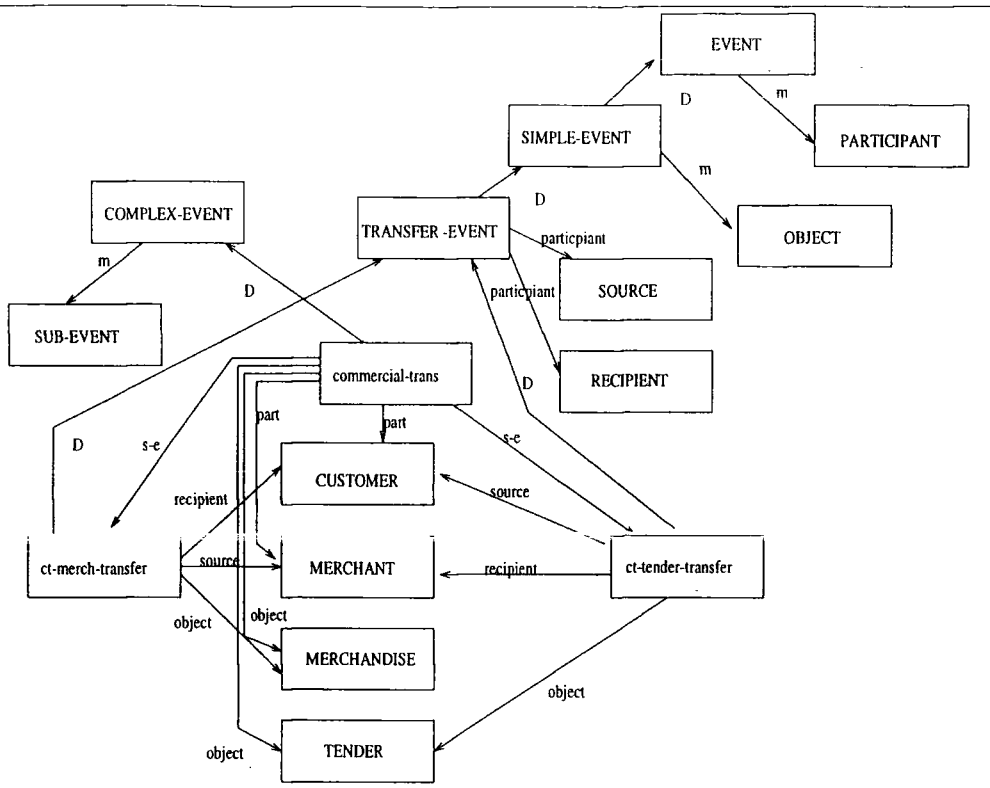


Figure 3.21: KING's commercial transfer event

customer receives the *merchandise* from the *merchant.* Concepts, such as *merchant*, *customer*, *merchandise* and *tender* are aspectuals of the *commercial transaction*; that is they are specific concepts whose meaning is un-detachable from the *commercial-transaction* event."

The rich representation forms an important core of knowledge about commercial transactions. For example, the knowledge that merchandise and tender play object roles is linked to knowledge about transitive verb forms so that phrases such as 'bought a book' and 'paid five dollars' conform to a general rule.

Moreover, Jacobs' system allows events to be 'VIEWed' in terms of other events using *structural associations.* For example, commercial transaction events would normally be realised using the verbs 'buy' and 'sell' but because of the hierarchical arrangement, and the use of VIEWS, these events can be related to 'giving' and 'taking' concepts and thus the verbs 'give' and 'take'. VIEWS are used to represent knowledge about concepts that may be used in expressing other concepts. (e.g. a transfer action may be VIEWed as a giving action or a taking action). Jacobs claims

that VIEWS and structural associations can also be used to create metaphorical expressions such as 'give permission' and 'give a hug' (it is disputable as whether these de-lexical verb phrases are metaphorical).

Unfortunately Jacobs does not give many more examples of the variations produced by the KING generator. He claims that the system can produce 200 structured associations on the representation but this comprises a total of only 150 concepts.

Jacobs concedes that the ability to apply structured associations leads to the problem of how to control their application. He says this control really depends on the context of the utterance (beliefs, intentions, situational knowledge and discourse knowledge) but can lead to good results using simple heuristics. The heuristics he uses include expression by super-category rather than by expressing a concept by describing a component, favouring a metaphorically related concept rather than one that is too specific or too general and those associations which produce linguistic structures directly.

Grammatical information is held in the same ACE representation (for example, a verb phrase hierarchy) and conceptual information is associated with the grammatical information using referential links. Thus, for example, verb phrases can be linked to conceptual role (for example, the recipient of a transfer event can be linked to the indirect object of a verb).

The work presented in this thesis also takes the view that a representation for generation needs to be knowledge intensive. Other systems have built this knowledge directly into the generator which reflects their building of a generator independently from a complete natural language processor. In a complete system (such as LOLITA) this type of knowledge has to be encoded for other components as well as the generator and should be generalised as much as possible. The small number of concepts in Jacobs' system mean that although intensive, the knowledge can only be extremely limited and only applicable for generation in a restricted domain.

Horacek's work

Horacek [Horacek, 1990] [Horacek, 1992] [Horacek, 1994] describes two systems, WEIBER and OFFICE-PLAN which also take comparable semantic network input.

WEIBER is a German NL consultation system. It covers 'the whole spectrum of NL processing tasks including analysis, response determination and generation' [Horacek, 1990] in the limited application domain of financial investment. OFFICE-PLAN is an expert system which provides an explanation to its problem solution. The system solves room assignment problems in offices, represented as a constraint-satisfaction problem. OFFICE-PLAN's generation module is called DIAMOND.

Like the work described in section 3.12.4 by Iordanskaja *et al.*, Horacek adopts an approach where intermediate representations are simplified in the generation process. However, as well as describing a simplification process at the lexicalisation level (see 'Terminological transformations' section below) he describes a similar process at the planning stage.

Integrated Planning

Horacek's transformations at the planning stage involve simplification of the content allowed by inference rules. His 'integrated'[6] view of text planning takes into account *conversational implicature*. The generation process starts with an argumentative structure which conveys the original and internal content of a system's communicative intentions. This representation is then augmented by adding (possibly redundant) supporting arguments to provide a source of variability in presenting the conceptual specifications. The final text structure is derived by successively modifying it to leave some parts implicit and thus making the utterance shorter. The content of the utterance which is made implicit has to be inferred in the following ways:-

- The inference is drawn by the hearer due to **world knowledge** attributed to him/her. For example (from [Reiter, 1990]) if a system wished to inform the

[6] not to be confused with the integration of planning and realisation components, section 3.5.2

user that a flight lands at La Guardia airport, it has, in addition to stating this fact explicitly, the option of simply calling the flight a shuttle and relying on the user's knowledge that shuttles land at La Guardia.

- The inference is carried out by the speaker. When trying to elicit information from a dialogue partner, the system, instead of asking a direct question, may ask another question the answer of which makes it possible to infer the information initially required. In the WEIBER domain for example, to determine whether the user wants to buy an asset with a high or low liquidity, the system may ask 'Do you want to have access to your money during the term of investment?'

- The inference is justified by the **context** and the hearer is supposed to draw it. This applies to newly established common knowledge and matters of coherence. In the office planning domain, for example, it would be adequate to answer the question 'Why is Smith assigned to group?' with 'Smith is a group leader' instead of citing the generic condition and class membership of the entities involved.

Terminological transformations

These transformations involve selecting an appropriate level of granularity on a conceptual basis by performing terminological equivalence operations. The aim of these transformations is to chose a concise but still comprehensible alternative according to the known or assumed knowledge of the user.

The transformations are carried out by a tool called FTRANSLATE which has two procedures with inverse functionality EXPAND and CONTRACT. Horacek only considers the former procedure as he assumes that the dialogue component passes only the most compact representation to the generator. (Note that this is the opposite of the assumption in the MTM work where semantic networks are replaced by more compact ones). The EXPAND operation substitutes subexpressions representing specialised concepts with more generalised ones augmented by additional roles and restricted fillers to maintain the terminological equivalence.

```
'Notgroschen' (money set aside for a rainy day)
EXPANDs to 'a savings account with more than two net months income'

'High liquidity associated with an investment'
EXPANDS to 'the possibility of its owner to have access to the money
            during the term of investment'
```

Figure 3.22: Examples of Horacek's terminological transformations

Examples Horacek gives of terminological transformations are given in figure 3.22, they seem to depend on very domain dependent rules.

The WEIBER system does not attempt to infer the user's experience from the course of the dialogue but performs terminological transformations based on a priori assumptions about the user (i.e the system assumes which terms the user is familiar with). Horacek mentions that there may be a choice of how to expand a specialised concept but this is not considered further.

Verbalisation

Verbalisation in Horacek's WEIBER system has already briefly been discussed in section 3.8.4. Horacek shows that when mapping from the conceptual representation to lexical structures there may be a choice of ZOOM schemata to apply at a particular point. Examples Horacek gives of variations resulting from differing applications of ZOOM Schemata are given in figure 3.23.

With respect to controlling which particular ZOOM schema to apply, Horacek says that decisions are made implicitly by favouring what is to be considered the locally best choice of ordering and by accepting the first legal solution. The ordering is governed by 'stylistic constraints' which, for example, comprise the avoidance of indefinite pronouns, production of complete sentences and a preference for concise verbalisations.

Horacek's work, whilst considering the whole generation process seems to still be in an experimental stage. Furthermore the constraints which guide the choice of

'name of the project'	'Bill's project'
'the name is WEIBER'	'Bill leads the project'
'the project has a name'	'Bill is the leader of the project'
'the name WEIBER'	
'bonds at a value of 40000DM'	'Bonds are recommendable'
'40000DM in the form of bonds'	'it is advisable to buy bonds'

Figure 3.23: Variations caused by application of differing ZOOM schema

variation seem rather simplified (e.g., the user model and the simple stylistic constraints described above) and the examples are given in rather restricted domains.

3.13 Conclusions

This chapter has presented an overview of the state of the art in Natural Language Generation. It has concentrated on the areas of research that are the most relevant to that which is presented in this work. Because the chapter has not been organised by system, Appendix A provides such an organisation by summarising some of the most important generation systems (with references to more details presented in this chapter). The Appendix also includes a system summary table which categorises them according to some of the aspects discussed in this chapter.

The latter part of the chapter has concentrated on systems which take semantic rich information similar to the SemNet representation used in the LOLITA system. The advantages of such input are as follows:-

- Such input can be a knowledge rich representation. This allows a knowledge intensive approach to generation [Jacobs, 1987] and means that information useful for ordering (for example temporal and causal information) is explicit.
- The input allows for a message directed control approach which is often more efficient (see section 3.4).
- The knowledge rich input can lead to the variation in the utterances produced. Furthermore, this variation can often be achieved separately from the

realisation process. The semantic representations can often be transformed in a separate process thus alleviating the realiser from extra burden.

Despite the advantages of these approaches, the systems which use such semantic network input have common limitations and disadvantages.

- The methods employed still rely on total coverage of the semantic input. This will cause a generation gap problem as the planner, which has to delimit semantic network portions, must know if these portions can be expressed in surface language.

- Because of this, the systems discussed comprise planners (or assume the future existence of planners) which are able to carefully delimit semantic network portions into sentence or clause sized chunks. Thus although the type of input has the potential to move away from the restrictions imposed by adopting other inputs (e.g., predicate calculus) this potential is not really exploited.

- The problem of lexicalisation (see section 3.9) still exists. The inputs considered comprise semantic network 'nodes' which are either primitive concepts (e.g CDT, CG) or actual lexical items (MTM). These assumptions are different to those imposed by the LOLITA SemNet representation (see section 1.5.3 and 4.3.2).

Chapters 5 and 6 will detail the solution adopted in the LOLITA generation system which takes LOLITA's semantic network representation, SemNet, as input.

Chapter 4

The LOLITA System

This work forms part of the LOLITA project. This project was briefly introduced in section 1.4 when the project's bearing on the methods adopted were discussed. This chapter will describe the LOLITA system in more detail with special attention to the aspects most relevant to NL generation.

4.1 History and Background

LOLITA (Large scale, Object-based, Linguistic Interactor, Translator and Analyser) has been under development at the University of Durham since 1986. It is a large project with an increasing number of researchers working on different areas and modules. Such a long term core project is rare in many areas of science including NLE and NLG (exceptions are the SNePS and Penman projects, for example). The project provides an exciting environment for research as work on individual sub-projects can contribute towards the total system (compared to other environments which only support isolated projects with short time spans).

4.2 Advantages of General Purpose Base Research

LOLITA is termed a *general purpose base* (see section 1.5.5) and forms a core platform upon which different NL applications can be built.

Although demonstration prototypes have been built using LOLITA for various tasks and domains (these will be introduced in section 4.4) no polished final application has yet been developed. This is because research resources have concentrated on the 'base' of the system rather than task-dependent development. The procedure has been to build prototypes for different tasks to see if these tasks are feasible and to concentrate on the *general purpose base* rather than specific application development.

One motivation for developing a *general purpose base* system is that of saving effort by using the same software for different tasks or applications. This is the *flexibility* aspect of NLE introduced in section 1.2.5.

Another advantage in *general purpose base* system research is that it forces development in general and fundamental terms. When designing the system components no particular task, domain or application was in mind. This generality often has unforseen benefits; for example, LOLITA was not specifically intended to be used for machine translation but because of the generality of the work, little effort was needed to produce a prototype Italian to English translator (see section 4.4).

4.3 System Overview

This section outlines LOLITA's architecture and introduces some of its important components. The information in this chapter is adapted from [Long and Garigliano, 1994] which itself was adapted from [Garigliano *et al.*, 1992]. Figure 4.1 shows a block diagram of LOLITA's components.

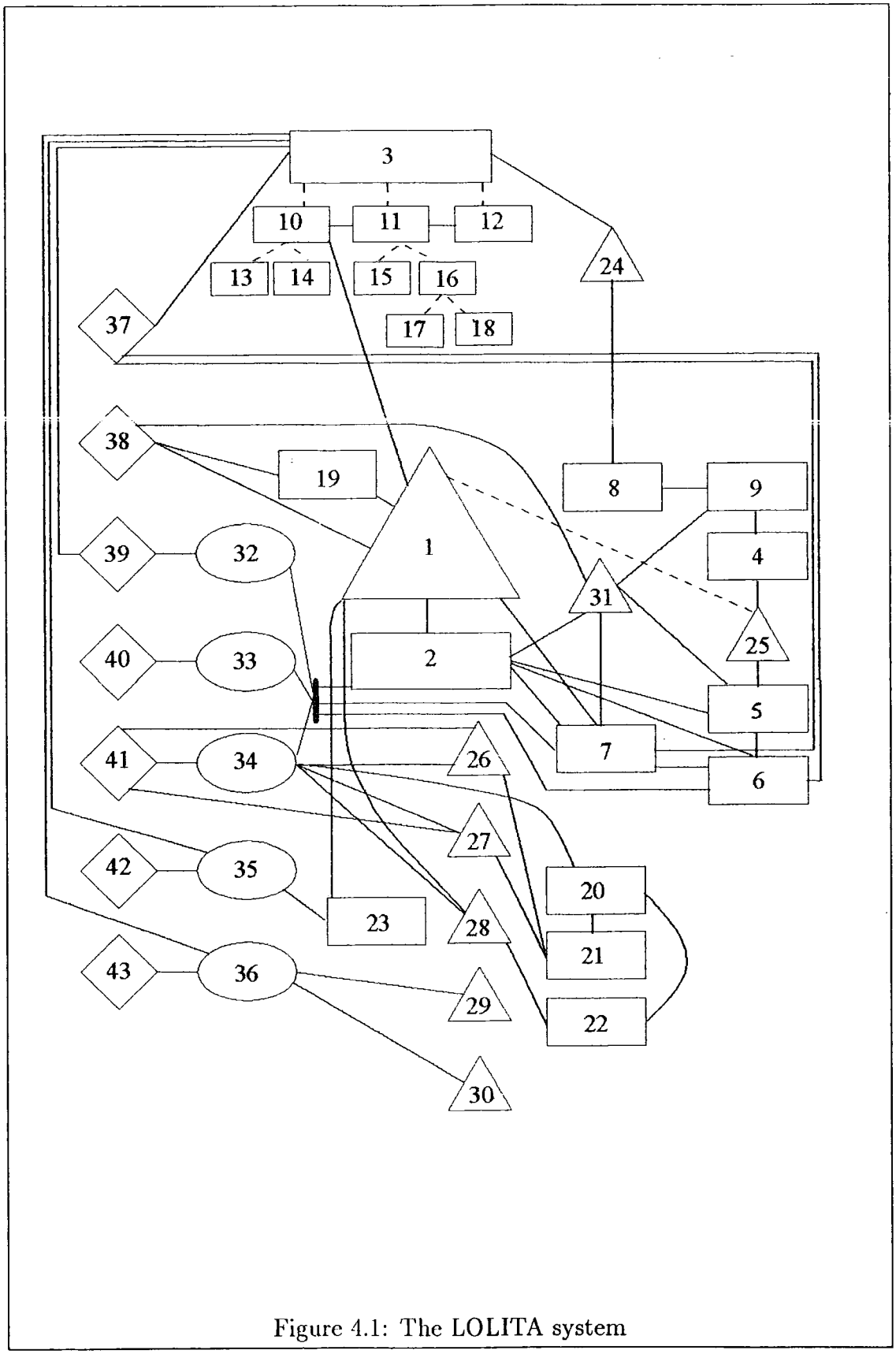


Figure 4.1: The LOLITA system

Legend for LOLITA diagram

- square = internal processing module
- diamond = interface module
- circle = application module
- triangle = data
- continuous line = module connection
- broken line = subpart relation

1 = semantic net
2 = inference engine
3 = syntax analysis
4 = semantic analysis
5 = pragmatics analysis
6 = discourse analysis
7 = natural language generation
8 = syntax tree normalisation
9 = pre-semantic normalisation
10 = morphological analysis
11 = grammatical analysis
12 = parsing search control
13 = misspelt words recovery
14 = new words guessing
15 = grammar structure handling
16 = grammar feature analysis
17 = structure analysis
18 = structure reconstruction
19 = consistency checks, data compression etc.
20 = dialogue planbox generation
21 = emotion analysis
22 = constraints analysis
23 = style analysis
24 = tree structure
25 = semantic net fragment
26 = LOLITA model
27 = user models
28 = dialogue structure models
29 = student models
30 = tutor models
31 = global switches
32 = query module
33 = template module
34 = dialogue module
35 = translation module
36 = chinese tutor module
37 = NLP interface
38 = semantic net interface
39 = query interface
40 = template interface
41 = dialogue interface
42 = translation interface
43 = chinese tutor interface

The following subsections will give details about the operation and main components of the LOLITA system.

4.3.1 Syntactic Analysis

Initial Preparation and Morphology

The first stage of syntactic analysis is to build a surface representation of the input string. Firstly, punctuation in the input is used to separate it into grammatical units and then spaces are used to separate these into individual words. Short hand words are replaced by their longer versions (e.g., 'I'll' to 'I Will') and when a word could relate to more than one concept node in the SemNet representation, all possibilities are included in the intermediate representation (e.g., 'bow' for a ship's bow or a violin bow)(see section on WordNet, section 4.3.2).

The morphological process extracts and labels the roots of the input words. Sometimes multiple extractions are required. For example, morphological analysis on the word 'unworthiness' will extract and label the word 'worth' by separating out the components 'ness' which makes an adjective into a noun, 'un' which indicates a negative, and 'y' which turns a noun into an adjective. There is also a facility at this preparation stage for recovering misspelt words [Parker, 1994] and guessing unknown ones.

Parsing

The prepared input representing the surface structure of the text is then ready for parsing where the words and constructs of the natural language input expression are grouped and labelled into a parse-tree. The parse-tree represents the grammatical structure of the text and the relationships between the component words.

The parser is based on the TOMITA algorithm [Tomita, 1986], a variant of the shift-reduce parser with a graph-based stack. This parser produces a large amount of possible parse trees (a parse 'forest'). LOLITA's grammar uses a set of

```
sen
 detph
   det THE
   comnoun COW [Sing,Female,Per3]
 auxphrase_advprepph
   compintransv JUMP [Past]
   prepp
     prep OVER
     detph
       det THE
       comnoun MOON [Sing,Neutral,Per3]
```

Figure 4.2: Example of parsing

features and penalties in order to discard unlikely parse trees. Lazy evaluation (see section 7.2.5) is carefully used so that only a minimal part of the parse forest is ever generated. This is a crucial step which allows us to use the TOMITA algorithm efficiently on long sentences (e.g., more than 30 words) which have a high degree of potential ambiguity.

The LOLITA parser produces the best parse tree (according to the penalties assigned by the grammar) or a list of possible parses representing the deep grammatical structure of the input. Each parse tree has all word features extracted (e.g., verb root rather than third person singular etc), errors (structural or feature caused) printed out, missing parts inferred and un-parseable parts isolated. Figure 4.2 shows the parsing of the sentence 'The cow jumped over the moon' whilst figure 4.3 shows an example of parsing the ungrammatical sentence 'and I likes him own'.

Normalisation

The next step in the analysis process is *normalisation*: equivalent parse trees are converted to a normalised form in order to reduce the number of mappings between parse-trees and the SemNet representation that the semantic translator must cope with. These *normalisations* include grammatical transformations (e.g., passive to

```
subsen_phrase
join AND
sen  * clash: Per3 *
defpronoun I [Sing,Sexed,Nom,Per1]
sentvbph
  sentverb LIKE [Pres,Per3]
  sen  * clash: NoPer3S *
    defpronoun HIM [Sing,Male,Nom,Per3] * clash: Acc *
    transvp
      comptransv OWN [Pres,NoPer3S]
      conjtermph * MISSING *
```

Figure 4.3: An example of parsing the grammatically incorrect sentence 'and I likes him own'

active, dative to non-dative), filling in missing phrases (e.g., 'John was kicked' normalised to 'John was kicked by someone'), rearrangement of prepositional phrases (e.g., raising prepositional phrases to the verb level) and removal of de-lexical constructs (see chapter 6).

During generation, the opposite process is useful: paraphrases should result from the same semantic input. This can be achieved using *abstract transformations*, an aspect of the solution presented in chapter 6.

4.3.2 Knowledge Representation

LOLITA's semantic network representation (SemNet) is important as it affects nearly all other parts of the system. This representation is especially important with respect to the work described in this thesis as it forms the input to the generator. Section 3.3 describes the importance the chosen input has on the design and operation of a generation module.

The structure

The LOLITA semantic network representation, SemNet, is a powerful representation scheme based upon a directed hyper-graph (i.e. it comprises links from a node

pointing to a set of nodes).

The nodes of the graph correspond to concepts (e.g., entities or events). Each node within the semantic net has a unique number by which it is identified (**the node-ref**).

Attached to each node is a set of control variables which contain basic information about the node (see section 4.3.2). The links between the nodes correspond to relationships between the nodes. A link is composed of a list of node-refs which are the range of the link, and an arc which identifies the type of the link, e.g., **subject_**, **object_**, **universal_**. A **subject_** link, for example, connects an event node with the nodes which correspond to the event's subjects, a **universal_** link connects a node to its universals etc.

Figure 4.4 shows an example event node together with an English description of what it represents (this description is produced by the LOLITA generator, the subject of this thesis).

The definitions of meaning, concepts and the relationship between concepts and language in SemNet have been discussed in section 1.5. Nodes (concepts) in SemNet are arranged in hierarchies with entities and events lower in the hierarchy inheriting properties from those higher up.

The LOLITA semantic network currently comprises in the order of 100,000 nodes.

WordNet

Although the LOLITA representation is concept rather than language driven (see section 1.5.3), it is often useful to use surface language information to build conceptual information. This assumption is based on the argument that language has evolved so as to provide a way of communicating important concepts: the existence of a word (especially common words) is a good indication of a useful concept.

The LOLITA SemNet representation has been built with the aid of WordNet

[Miller, 1990]. WordNet is a lexical reference system comprising lexical and semantic information about word forms in English (American English). Word meanings are represented by synonym sets - a list of synonymous word forms that are interchangeable in some context.

Control variables

As mentioned above, a list of *control variables* is associated with each node in SemNet. These variables contain standard information which is shared by a large number of nodes. This information is essential for many components of the system (including generation) and needs to be accessed often and quickly.

This information could be represented elsewhere in the network (e.g., as part of the hierarchy) but because it is required often it is stored explicitly with each node: this is a straight sacrifice of memory resource for faster execution speed. There are currently about sixty different types of control variables used in the LOLITA representation, three examples of which are discussed below:-

- **Rank**: The *rank* of a node gives the node's quantification and can have the following values :- individual, prototype, general, universal, bounded existential, named individual, framed universal or class. The system uses a multi-sorted logic representation to deal with the concept of rank. Instead of just having two kinds of entities, variables and constants, and quantifying over these variables, the LOLITA system just uses constants.

  There are various types of these constants, indicated by the rank, which obey different inferential rules. The complete set of inferential rules together with these constants is equivalent to a first order logic. This representation method has two advantages, efficiency and naturalness.

  From the point of view of efficiency, the multi-sorted representation allows a node to be considered on its own merit rather than (as in the case of a quantified variable) as part of an event. This limits the need to access such associated events and improves look-up efficiency.

Secondly, under the naturalness point of view, the multi-sorted deep representation is more similar to the surface structure of the language. For example, in the sentence "Every man owns a dog", the constructs 'Every man' and 'a dog' have the same grammatical type (i.e noun phrases) but have different quantifications ('Every man' is a universal quantifier, 'a dog' is an existential quantifier). These types can be directly represented by constants of different ranks.

- **Type**: A node can have the following values of *type* control :- entity, relation, typeless, event, fact, greeting, procedure, determiner, punctuation, attribute, mode, preposition, pronoun, conjunction or sub-conjunction. These control values are very similar to grammatical qualifications with a few exceptions and additions. For example the *relation* type mainly represents verbs, *attribute* represents adjectives and *entity* represents nouns.

- **Family**: This control classifies nodes into the semantic and pragmatic groups to which they belong. The values for this control are:- living, vegetal, animal, human, inanimate man-made, inanimate, animal or human, generic, inanimate organic, abstract, concrete, not human, temporal and location. Of course there are many ways in which a node could be categorised but these family values are used to discriminate between the possible meanings of verbs. For example consider the phrases 'drive a car' and 'drive sheep' which contain different meanings of the verb 'to drive'. Because a car is classified in the family *inanimate man-made* and sheep in the family *animal* the correct meaning of the verb 'to drive' can be identified in each case. If an alternative classification, for example invertebrate or vertebrate were to be used, the problem would be more difficult to handle.

Prototypical events

LOLITA's SemNet representation contains 'prototypical' events which define events by imposing selectional restrictions on the roles associated with that event.

A typical example of such an event is 'ownership' which can be represented by the surface utterance 'HUMAN OWNERS OWN THINGS'. This event encodes the following selectional restrictions: that a subject of the action 'own' has to be human (in fact it has to belong to a set of 'humans who are also owners') and that the object of 'own' must be non-human. Any event whose action will be 'own', will be represented in SemNet as a specialisation of this prototypical event. Consequently the same restrictions on the subject and object will apply to the more specific event.

This information can be used during pragmatic analysis for enrichment and disambiguation of meaning. For example, given the sentence 'He owns a motorbike' (and in the absence of any specific context) the semantics will produce an event whose subject is very general - i.e. 'a male creature'. The prototypical event associated with 'own' will allow the pragmatics to further specify the subject and determine that it is 'a man owner'.

Prototypical events, like entities, are arranged hierarchically with lower events inheriting properties of those above. It is important to note that actions themselves do not form a hierarchical structure: it is the prototypical events that define those actions that form such a hierarchy.

Prototypical events are important in generation as they allow knowledge intensive transformations leading to paraphrases (see chapter 6, abstract transformations).

4.3.3 Semantic and Pragmatic analysis

The purpose of the semantic analysis is to map the deep grammatical representation of the input (the information carried by the parse tree) onto nodes in the network. The analysis must determine whether a node already exists, if and how to build a new node and how to connect existing and new portions of the network. An existing node must be identified or a new node built for each object and event involved in the input text. For example, for the parse tree given in figure 4.2,

```
**********************************
    * event: 32035 *
 generalisation_:
   event - 7688 - rank: universal - definition_
 subject_:
   report - 32024 - rank: universal - suspended_
 action_:
   suggest - 3435 -
 time_:
   past_ - 20991 -
 date:
   31 October 1992
 source_:
   telegraph - 9994 - rank: named individual
 status_:
   suspended_ - 29025 -
 object_:
   explosion - 32011 - rank: individual - suspended_
**********************************
```

First reports suggested that at 9pm at night when a forceful person forced a driver to drive a black taxi to Whitehall, a bomb went off in it on a corner outside Cabinet Office and outside 10 Downing Street.

Figure 4.4: Example of an event in the LOLITA representation

nodes representing *the cow*, *the moon* and the event of *the cow jumping over the moon* must be created or identified. This process can be separated into different stages.

The first step is to make references absolute. There are many contextual referents in speech. The referent 'I', for example must be replaced with the person who is speaking and the referent 'you' with LOLITA. There are more complicated examples, consider the phrase "I'll do that tomorrow". The word 'tomorrow' cannot be directly represented as it is contextual: instead it must be represented by the concept for the day after the particular day when this input was uttered.

The second stage is to disambiguate the grammatical parse tree in order to decide on one of many possible interpretations. For example, take the sentence "I like the bottle of wine on the table because of its fruity taste". Here the referent 'it' refers to the bottle of wine (in fact not even to the bottle but to the wine inside it) and not the table. This ambiguity will be resolved by looking at the semantic network to find that taste is an attribute of wine and not tables. In fact such ambiguities are present in what could seem, at first thought, to be very simple words. The word 'the' for example has at least six possible semantic meanings (see section 5.4.3).

Once a new or modified portion of the semantic network has been built there is still a problem in checking that this portion is consistent with the existing network. Pragmatic and more semantic analysis is necessary to achieve this. Take for example, the sentence "I saw a pig flying", here the syntax is correct and the semantics might also be considered well formed. If this is the case then the pragmatic analysis must be able to conclude that there is a problem with the acceptability of this sentence. Alternatively, this sentence may be incorrect under the semantic point of view if there is some definition explicit in the semantic network saying that pigs do not fly.

There are many instances however, when there is no way in which the semantic analysis could find such an error and the problem is purely pragmatic. For example, in the sentence 'I bought a car from the Japanese manufacturer Ford', there may be

nothing in the semantic representation which says that Ford cars are American and not Japanese. The pragmatics will have to work this out using inference techniques. The role of pragmatics is, therefore, to adjust the semantic representation of any new or modified nodes so they can fit into the overall network. This may cause drastic actions such as attaching disbelief to events or assuming a misunderstanding and asking further questions for clarification.

One way to deal with a clash of pragmatics between new and old information in the network is using source control. This comprises deciding whether or not to accept or attach a certain degree of belief to a piece of information by looking not at the information itself but at where (or from whom) the information came from and the way in which this information was provided. Work on source control has been carried out at Durham for several years [Bokma and Garigliano, 1992] and a large model has been built which is currently being incorporated into the LOLITA system.

4.3.4 Dialogue

Many dialogue systems have been attempted, but due to the fact that no universal theory of dialogue has been defined, these have concentrated on particular applications. Such a general dialogue theory is being developed as part of the LOLITA project [Jones, 1994] and is loosely based on Schank's script theory [Schank and Abelson, 1977]. As we encounter different experiences in our lives, we come to expect certain types of dialogue in different situations, and we learn what is the 'norm'. If a computer system is to understand and then generate natural language in a dialogue, it must be capable of recognising what is appropriate within the dialogue. Because of the lack of a universal theory, it is impossible to describe all the possible dialogue situations (an infinite number) which may occur. The LOLITA dialogue project overcomes this problem by standardising dialogue using *Dialogue Structure Models* (or DSMs). The DSM is a schema which holds information about a stereotypical dialogue. The DSMs can be used not only to help in the understanding of a dialogue (by constraining what to expect) but can also aid the generation

of dialogue (by constraining how to respond or continue the conversation).

DSMs are composed of fundamental elements called *Dialogue Structure Elements* (DSEs). Each DSE describes one fundamental property of a dialogue: the presence, absence and, in some cases, strength of a DSE will affect the dialogue structure.

DSEs are classed into three categories:-

- *External elements* (EE) are those which can be observed with no other knowledge of the dialogue. *Number of participants* and *time limit* are examples of this category of DSE.
- *Motivational elements* (ME) are those which describe the purpose of the dialogue. As every dialogue is assumed to have a purpose, every DSM must possess a DSE of this type. Examples are persuasive, emotional exchange and information seeking.
- *Verbal elements* (VE) describe other verbal elements which may or may not be present in a dialogue. Rhythm and rigidity are examples.

By adding, deleting or changing the strength of a DSE in a DSM, the dialogue that the DSM models will become fundamentally different. A completed DSM will allow the system to know all it needs to know about what to expect from a dialogue situation. The model can be used to aid the generation of dialogue through a list of constraints which are associated with each DSE. These constraints are actions which a system may or may not carry out during a dialogue if a particular DSE is present. The constraints associated with each DSE are not fixed. Thus whilst the presence of a set of DSEs in a DSM will define the dialogue, the constraints associated with each DSE will provide more subtle information.

For example the following constraints may be associated with the *Dominance* DSE :-

- The right to initiate the dialogue.

- The right to terminate the dialogue.
- The right to interrupt the other participant(s).
- The right to choose the topic.
- The right to change the topic.
- The right to initiate sub-dialogues.

The dialogue theory is currently being implemented and interfaced with the LOLITA system and its the generator.

4.4 LOLITA Applications

4.4.1 Analysis of Text

The basic operation of the LOLITA NL general purpose base is to analyse text in order to build a representation of its meaning (i.e., the operation discussed above, sections 4.3.1 to 4.3.3). Information gleaned for input text is identified in or added to the SemNet representation. This is the general base operation for most other applications.

4.4.2 Query

This application allows a user to interactively provide information to LOLITA and interrogate the system using NL utterances. Once a piece of text has been analysed, for example, a user may ask NL questions about the information that LOLITA has gleaned from this article.

4.4.3 Translation

This prototype exemplifies the flexibility of LOLITA (see section 1.2.5): machine translation was not an original goal of the system but with only a small amount

of modification, a prototype has been developed. By adding a few rules to the LOLITA grammar, Italian text can now be interpreted and the information it contains added to SemNet. Since the generator can produce English from SemNet, a framework for Italian to English translation results. For more information see [Morgan *et al.*, 1994].

4.4.4 Database Front-end

A project has recently been initiated to use LOLITA as a NL front-end to a database. This project will involve translation of the SemNet representation of NL database requests into the database query language SQL [Smith *et al.*, 1995] [Garigliano *et al.*, 1995].

4.4.5 Contents Scanning

This application involves analysing input texts and filling domain dependent templates so as to summarise the content of the original text. This application is of particular interest as it is actually performed by a wide range of agencies (for example intelligence gathering organisations, communication centres etc). Until recently there were two existing methods for performing this task; manual and automated keyword search. Manual search is labour intensive, prone to errors, costly and time consuming. Keyword search on text can improve on this time problem but cannot cope with phenomenon such as negative, hypothetical or distributed information. More recently NL techniques have been applied to the problem with the aim of developing systems which are able to provide the accuracy of a manual search with the speed of keyword search.

Existing NL systems for content scanning fall into two types, those which attempt to perform a semantic analysis to arrive at a representation of the meaning of text (for example PROTEUS[Grishman and Sterling, 1993], TACITUS[Hobbs, 1991]) and those which focus on specific understanding tasks such as looking for patterns of particular words (for example JASPER[Andersen, 1992], SCISOR[Jacobs

and Rau, 1990]). The choice between the two methods is again a choice between accuracy and speed: the former group can build a much deeper understanding of the input while the latter can work at speeds which are adequate for real applications.

The LOLITA content scanning prototype [Garigliano *et al.*, 1993] falls into the former group of systems as it based on the LOLITA general purpose base. As this general purpose base is designed to be domain independent this should lead to advantages in portability compared to other systems (for example, some of those which have competed in the recent series of DARPA[1] sponsored MUC competitions, e.g. [DAR, 1993] see section 8.1.1) that have been designed to operate in one specific domain.

However, a domain and template dependent module is still required in the LOLITA scanner which is responsible for searching new portions of the SemNet representation in order to find relevant information with which to fill template slots. The requirement for such a hard-wired domain dependent search module is a disadvantage. There are plans to interface the contents scanning module with the dialogue module so that specification of template requirements can be initialised and modified using NL interaction.

Very recent work in collaboration with an industrial partner has concerned evaluating the LOLITA contents scanner with domain independent templates. Articles of average length of about 100 words from a wide variety of domains were analysed in order to fill templates with domain independent slots such as 'personal name','organisation', 'locations', 'animates', 'inanimates' etc. The system was found to be 100% robust (i.e it always managed to produce a template and never crashed) and resulted in recall and precision scores comparable with other state of the art systems. These results are extremely promising as these other state of the art systems were restricted to doing the template task in a single predefined domain. The LOLITA system is entered in the MUC6 competition (see section 8.1.1 and [DAR, 1993]) which comprises, among others, a similar template-filling evaluation.

[1]Defense Advanced Research Projects Agency

```
Parse Tree:
  chinese_sen
    en_prop_ph TRANSFER ERROR
      noun_per 教 授 (Prof.)
      proper_per 张 (a surname)
    en_prep_vp  TRANSFER ERROR
      trans_vp
        trans_v2 去 (go)
        proper_noun 伦 敦 (London)
      prep_ph
        prep 跟 (with)
        per_pronoun 我 们 (us)
```

Figure 4.5: Example of a Chinese parse tree

4.4.6 Chinese Tutoring

The wide variety of possible applications which can be built on the LOLITA base is exemplified by the Chinese tutoring prototype [Wang and Garigliano, 1992] [Wang, 1994]. The prototype involves helping students learning Chinese to overcome the problem of transfer errors caused by mother tongue influence.

By using the existing parser and adding Chinese words and grammar rules a tutoring module has been built which uses intelligent tutoring techniques (by using and updating various models of the situation, e.g., student model, expert model etc.) to ask users to perform English to Chinese translations. Users' translations are then parsed and the resulting parse tree is diagnosed for transfer errors.

An example Chinese parse tree showing an error caused by transfer is shown in figure 4.5.

4.5 The Role of Generation in LOLITA

Most of the applications that have been or are to be built 'on top' of the LOLITA general purpose base will need some generation capability. The generator described

in this thesis must provide the required capabilities and be as flexible as possible to allow its use in future applications (see Aim 3, section 2.2.3).

This section will describe the generation capabilities required by each of the prototype applications described above. It is important to note that these applications were not built without generation capabilities before this project was initiated. Rather, their development has been in parallel with the generation work described here. This has allowed the development of applications and the generator to influence each other (see the NLE principle of *integration*, section 1.2.6).

- To rebuild surface language expressions for SemNet. As the construction of semantic representation from input text is the base operation for the LOLITA system as a whole, re-building surface language expressions from this representation is the basic operation required for the generator. As well as forming the base for other applications, surface expressions are useful for checking the consistency of SemNet during, for example, development. As mentioned before (section 1.5.1), in theory a concept in SemNet is defined by the whole of the network. In practice however this is both unnecessary and un-practical for generation: depending on the application, the generator (and perhaps, application dependant controlling mechanisms) will have to decide on how best to produce an expression. This project is concerned with the generation of such surface expressions in English.

- Query. During the query application, a generator is required to produce NL utterances in response to questions. As in other applications this is achieved by passing a concept in the SemNet representation which corresponds to the desired response to the generator. During query sessions, it is also important (e.g., for context reasons) to represent the semantics of the question that was asked. It is therefore necessary for the generator to be able to re-build surface expressions for the semantic representations of these questions as well as the answers.

- Dialogue. Obviously, dialogue will require LOLITA to produce NL utterances. Again, this is to be handled by passing SemNet concepts to the gen-

erator. In this case the dialogue component (see section 4.3.4) will interface with the generator so as to drive it to produce appropriate responses. In dialogue, not only the content but also the method of presentation of the content is important. According to the constraints imposed by the presence of the different DSEs, the dialogue and utterance planning (see chapter 5) modules should be able to control the content and desired style of the utterance. The dialogue module currently uses a simple reactive planner, there are plans in the future to integrate the module with a more involved hierarchical abstraction planner based on the AbNLP planner [Long and Fox, 1995] [Fox and Long, 1995].

- Translation. By adding the ability to analyse languages other than English, a prototype machine translator has been built. By building semantic representations from foreign language input and then rebuilding the semantic representation in English, the content of the original expression can be translated. For a polished translation, not only content but some aspects of presentation and style have to be conveyed. Although this is beyond the current capabilities of the prototype translator, the generator should be built so that it can be controlled as much as possible in the future. Work has recently been initiated to build a generator for Spanish and there are plans to build generators in other target languages in the future.

- Contents Scanning. The template filling module will require the generator to build NL utterances in order to fill the various slots in the template. Again this is done by passing down relevant SemNet concepts. This time, due to the desired summarisation effect of the templates, instructions will also be passed down to favour brief utterances.

Chapter 5

Solution: The General Approach and the Plan-realiser

5.1 The General Approach

The solution to natural language generation (NLG) in the LOLITA system is based on two important principles which are different from those adopted in other generation systems. These factors are:-

1. Generator Input. The generator receives as input the whole of the semantic network (SemNet) and this is available throughout the generation process.

2. Generator Architecture. This input allows a novel architecture which avoids the 'generation gap' problem in traditional two-component architectures (see section 3.8). Although a two component approach is adopted (see section 3.5), the distribution of tasks between the **planner** and **plan-realiser** modules is different from that between traditional planner and realiser modules. More specifically:-

    - The role of the **planner**. The planner's job is to constrain the plan-realiser by passing it instructions.

- The role of the **plan-realiser**. The plan-realiser is autonomous. It tries to follow the instructions passed by the planner but in the absence of these instructions (or if these instructions are not achievable) it must be able to make default decisions of its own. The plan-realiser may sometimes perform some of the tasks (e.g., content selection) that are more traditionally undertaken by the planning component.

The rest of this section will expand on these factors and relate them to the more traditional approaches.

5.1.1 The Input

As discussed in the methodology chapter (section 1.4) the use of LOLITA is a starting assumption for this work: more specifically the generator must take input in the form of LOLITA's semantic network representation, SemNet. There are some systems which take comparable semantic based input (i.e., those described in section 3.12). However, these systems operate by delimiting content by 'cutting out' semantic network portions. These semantic portions, which can typically be expressed in one sentence, are then passed to a realisation module. The delimitation task is either assigned to a planning module (which may or may not be implemented), left to the underlying application or simply assumed to be achieved elsewhere.

A major decision in this work was not to base the solution on this 'cutting out' of portions approach. Rather, the generator and its subparts (i.e., the planner and plan-realiser) have access to the whole of SemNet. This approach conforms to that applied across the whole of the LOLITA system: every component of the system has access to the complete SemNet and is based on the philosophical assumption that the meaning of a node in SemNet is represented by the whole of that network (see section 1.5.1).

The enforced use of SemNet as input and the decision not to perform an explicit delimiting (or 'cutting out') process has a impact on the architecture and design of

the generator. Solutions applied to systems with vastly different input cannot be applied here. Many generation systems, for example take predicate calculus and other linear structures as their input. These inputs already contain explicit content and ordering: the ways in which they can be used for generation are different to those that have to be used for a non-linear input structure such as SemNet. Furthermore, methods adopted for systems which do take a similar non-linear input often rely on the fact that portions of this input are explicitly delimited or 'cut out'. Sowa's notion of an *utterance path* ([Sowa, 1984], section 3.12.2), for example, means that the realiser has to find a path which visits every node in the semantic input. Similarly, incremental reduction ([Nogier and Zock, 1992] [Nicolov *et al.*, 1995], section 3.12.2) aims to incrementally match portions of the semantic input to linguistic structure until all the semantic input is covered.

5.1.2 The Architecture: Introduction

Like the majority of NLG systems, the solution is modularised into two components: a planner and a plan-realiser (see section 3.5). The LOLITA generator therefore, adopts a *separated architecture*. However, to overcome the generation gap at the interface between traditional planning and realisation modules, the roles of the two components within the generator architecture are different from that of other approaches.

The *planner* provides a list of instructions specifying the content and the style of the utterance to be produced. At the very least this must be a reference (or list of references) to nodes in SemNet. The planner does NOT cut out portions of SemNet: rather it provides instructions on where to start in the network and indications of how to realise it. In general the planner makes decisions according to issues which are not surface language specific. A complete planner has not yet been implemented but its role will be discussed further in section 5.1.3.

The *plan-realiser* must act on the instructions of the planner. If no explicit instructions are passed except for content (which must be provided by either the planner or an underlying application) then default instructions are assumed. If the

planning instructions are vague, or even conflict, then the plan-realiser will have to perform some tasks which are more traditionally assumed by planning modules. For example, influenced greatly by the actual content of the SemNet input, the plan-realiser may have to make decisions on content delimitation and sentence organisation. The plan-realiser has been implemented and will be discussed in more detail in section 5.1.4 and in the bulk of the remainder of this chapter (sections 5.2 to 5.12).

An example illustrating the range of instructions which could be passed from the planner to the plan-realiser is given below:-

- The plan-realiser might get the instruction to say something about an explosion. All the planner will provide is the reference to the node representing the explosion in the network.
- The plan-realiser might get the instruction to produce 'a medium sized utterance about the explosion, mention the casualties and the damage but don't say anything about who was responsible for the explosion. Use short sentences with little colour and avoid using the passive voice'.[1]

5.1.3 The Architecture: The Role of the Planner

Although some LOLITA modules contain simple planning procedures (for example, dialogue, section 4.3.4) a full planner has not been implemented and is outside the scope of this work. However, in the interest of integration (see section 1.2.6), its future operation has to be described, particularly with respect to how it interfaces with the plan-realiser.

As introduced above, the role of the planner in this novel architecture is to provide instructions to the plan-realiser which constrain how SemNet is to be realised. Although it is not necessary to define exactly the type of instructions that the planner will provide, it is necessary to have an architecture in which the planner

[1] Of course the instructions will not be in the form of NL as used in this example !

has access to 'hooks' into the plan-realiser so that its instructions can be carried out. The plan-realiser should then be able to produce a wide variation of output according to these 'hooks'.

The NLE principle of integration also demands that unimplemented or missing modules should be achievable. The assumption that the planner described in this architecture is achievable is validated because:-

- The planner will not have to find the optimal solution. There is a general planning theory that suggests that a planner which simply provides a set of useful constraints is easier to develop than one which tries to find the optimal path. For example, it is not too difficult to develop a planner which plans good chess moves but very difficult to develop a planner which provides the best move. The NLG planner in the architecture described here does not have to produce the optimal path. In fact, it does not even have to provide a consistent set of instructions (see section 5.1.5) as the plan-realiser with its heuristics about how instructions can be linguistically realised, can chose between them. In effect, the proposed architecture has shifted responsibility away from the traditional planner to the plan-realiser.

- The demands on the planner need not necessarily be high. The planner will have to automatically choose relevant instructions to pass down to the plan-realiser but it is assumed that this task will be achievable, especially when compared to the aims of the more traditional planner (which must, for example, carefully delimit semantic content, order each clause and, to avoid the generation gap, ensure that the plan-realiser can cope with its results).

- An intermediate planner does already exist. The dialogue application (section 4.3.4, [Jones, 1994]), for example, uses a planner which comprises a *template* element and a *motivation based reactive* element. The *template* aspect of the dialogue planner defines the current situation in terms of dialogue structure elements which constrain the behaviour of the system. The *reactive* aspect models the 'individuality' in a dialogue situation: the characteristics of the speaker, her motivations and immediate emotions are used to con-

strain the next utterance produced. At the moment the reactive element of the planner only 'reacts' to the last utterance entered in the dialogue so currently no 'long term' plans can be executed.

- The planner does not have to know linguistic details. The architecture allows the planner to make decisions on a conceptual level without having to take into account linguistic details. The 'generation gap' problem is avoided as the plan-realiser can ultimately overrule suggestions made by the planner which cannot be realised in surface language.

- The planner will not operate in isolation. The planner will have access to the many other components of the LOLITA system. The rich information held in the SemNet representation and components such as the user model, the dialogue planner and source control (see chapter 4) will aid the planner in its task.

- Planner development is already under way. Substantial work on the development of the LOLITA NLG planner has already been achieved. The planner uses state of the art hierarchical abstraction planning methods [Long and Fox, 1995],[Fox and Long, 1995].

At the present time, and for the purposes of this work, the operation of the planner is simulated. A series of operation methods, commands and switches (termed realisation parameters) have been provided and will be discussed in the next subsections.

It is important to note that a planner is not always needed for the generation system. This is illustrated by the fact that although no planning module exists, the realiser is now able to be used to provide useful output. Some applications may bypass the planner even if it did exist. For example, for each slot in the template, the contents scanning application passes references to nodes in SemNet that it wants to be described. Instructions which constrain how the utterance for each slot is realised may also be provided (by setting the required realisation parameters, see below).

Node by Node

In this method of operation, a planner is not required, the plan-realiser is simply passed SemNet and a reference to a particular node inside it. The plan-realiser assumes that the realisation parameters (see below) are set to their default values. This is equivalent to the instruction 'say something about x', where x is a node in SemNet. This operation is utilised in LOLITA's basic semantic analysis task (see section 4.4.1) where a description of each node is produced for each new SemNet node created by the syntactic and semantic analysis. This generation has been extremely useful in the debugging of LOLITA as natural language utterances are much easier to understand than SemNet itself. Development is extremely important in NLE, and without the generator the development of LOLITA would be extremely slow, if not impossible. It is very difficult for a developer to progress when faced with an output of twenty or so nodes (which is typical for the paragraph length pieces of text that LOLITA can analyse). An alternative would be to develop a graphical interface that could be interpreted more easily than a textual output of the nodes. However, this type of graphical interface would only be useful for development where as the generation of NL utterances is needed for other purposes. The use of the plan-realiser in the development process is strong evidence to support its success even in the absence of a planner.

Realisation Parameters

Realisation parameters are switches which can be set by the planner (or simulated planner) or the underlying application to directly affect the way the plan-realiser produces utterances. Alternatively, to produce a variation in utterances, these realisation parameters can be set randomly. These parameters can be set globally to affect a complete utterance or more locally to affect individual sentences (see 'story' command below).

There are four categories of realisation parameter :-

- Grammatical: this type of realisation parameter can directly affect the gram-

matical style of the utterance. Examples are the self-explanatory Passive/ Active and Dative/ Non-Dative realisation switches.

- Style: this type of realisation parameter will affect the generic style of the utterance. For example, a **colour** parameter can be set to affect the use of colourful synonyms, the number of adjectives and punctuation. A **rhythm** parameter controls the lengths of individual sentences and clauses within the utterance.

- Content: as well as explicitly indicating information which should or should not be said in an utterance, there are more general content parameters. A **length** parameter, for example, can control the total length of an utterance and thus how much content it contains.

- Abstract Transformations: the fourth set of realisation parameters can determine which abstract transformations should be carried out. Abstract transformations (which will be discussed at length in chapter 5) can be used to produce variations and paraphrasing of utterances by modification of SemNet prior to realisation.

Realisation parameters are derived from work on style analysis by Emery [Emery, 1994]. This work takes as a starting point the work on style by Hovy [Hovy, 1988b] (see also section 3.12.1) and DiMarco and Hirst [DiMarco and Hirst, 1993] which suggests sets of high level classifications of style together with a set of low level rules for how they are manifested on the surface level. Emery performed extensive analysis of a wide variety of real life texts in order to identify parameters which can be used to connect these two levels. The existing realisation parameters were chosen according to the evidence provided by Emery. Although there is not a direct corelation between these parameters and the ones identified by Emery, the realisation parameters form a subset which is sufficient to show that they can be used in this architecture.

The 'story' Command

The story command has been built to allow a user to be able to mimic the planner and to interactively build paragraph length pieces of text. The user can formulate instructions to the plan-realiser by inputting a series of SemNet event node references and a list of realisation parameters to be associated with each of these nodes. For each node the user also provides information as to the importance of the node. The various options are :-

- **Must describe separately**: the concept must be described as a separate sentence or principle clause within a sentence.
- **Must describe**: the concept must be mentioned somewhere in the utterance but not necessarily as a separate sentence or as a principle clause.
- **May describe**: the concept can be described if it fits well into what is being said.
- **Do not describe**: the concept should not be described. In the case of events, the event should not be mentioned at all. In the case of an entity acting as a role of the event, then the event can be expressed without explicitly mentioning that role. For example if a subject of an event should not be mentioned, the event can be passified and the subject omitted (e.g., *'The dog was kicked'*).

Figure 5.1 shows an example of the story command in operation. After analysing input text, LOLITA displays a NL summary of each of the events contained in that input. The user, playing the part of the planner, can then enter instructions which the plan-realiser can follow to form an utterance.

5.1.4 The Architecture: The Role of the Plan-realiser

The aim of the plan-realiser is to produce surface English expressions for concepts represented in the LOLITA representation (SemNet). There are two important

```
96018:You were tired.
96020:You called a warm taxi.
96022:You own a home.
96023:You went to your home.
96024:A taxi was warm.
96028:A driver was cool.
96029:A driver was cool and a taxi was warm.
96068:You gave a driver a big tip.
96045: <96068> because <96029>.
96066: <96023> because <96018>.
96046: <96020> because <96018>.

Example 1.

Instructions :-
96018, separately, short rhythm then
96020, separately, short rhythm then
96023, separately, short rhythm then
96024, separately, short rhythm then
96028, separately, short rhythm
       (mark driver in scriptural context) then
96068, separately, short rhythm.

Output:-
You were tired. You called a taxi. You went home. The taxi was warm.
The driver was cool. You gave him a big tip.

Example 2.

Instructions :-
96045, separately long rhythm, colourful (mark driver in
       scriptural context) then
96046, separately.

Output:-
You gave the driver a big huge tip because he was cool and the cab
was warm! You called it because you were tired.
```

Figure 5.1: Example of the 'story' command

differences between the plan-realiser and more traditional realisation modules:-

- The plan-realiser has access to all information held in SemNet both linguistic and semantic.
- This allows the plan-realiser to be autonomous. Without sufficient instructions from the planner, or when these instructions cannot be realised, the plan-realiser may make planning decisions on its own.

Given the complete semantic network and a reference to a particular node in the network, the realiser will generate an English expression for that node and follow as many planning instructions as possible. The tasks of the plan-realiser therefore, can range from having to do a lot of work normally associated with a planner itself (for example content selection) to merely following detailed planning instructions.

Ultimately, the plan-realiser must relate concepts in SemNet to lexical items. However, as described in section 1.5.3, the granularity of concepts in the SemNet representation is much smaller than that of words. Only some concepts (i.e., nodes) in the semantic representation will have a link to a lexical entry which will be adequate to convey the meaning of that concept (a discussion about how a concept can be 'adequately' realised is presented in section 5.4.1). These concepts are termed 'language-isomorphic' concepts or nodes (see section 1.5.3). The plan-realiser must specify non language-isomorphic concepts in terms of other concepts. This happens recursively until language-isomorphic concepts are reached that can be expressed as single lexical items (when presented with the correct quantification, morphology and formative linking expressions).

The job of the plan-realiser, therefore, is to search the network in order to 'decompose' a concept into language-isomorphic concepts that can be used to describe the original concept. To adopt a term used elsewhere (e.g [Sowa, 1983]) this search process finds the *utterance path*. Sowa's utterance path, however, aims to visit every node in the semantic input. This is clearly not the case in the LOLITA plan-realiser as the semantic input comprises the whole of SemNet (in the order of 100,000 nodes). In LOLITA's terms the utterance path is the path the plan-realiser

must follow in order to produce an 'adequate' utterance for a particular concept. The search for this utterance path is critical and depends on the following three factors:-

- What is present in the network: The SemNet input, representing the knowledge from which the utterance is to be drawn, is the most important factor determining the plan-realiser's search. The search depends heavily on what is actually contained in the network (in terms of arcs and controls, see section 4.3.2). This property represents the procedural control in the plan-realiser (see section 3.4). Because the semantic network is a rich source of information it would be irrational to ignore it and adopt a declarative approach.

- The grammar: The plan-realiser itself contains grammar rules which constrain the search so that correctly formed utterances can be built in the surface language.

- The realisation parameters: These parameters represent instructions passed down from the planner and can affect the order in which arcs are followed.

5.1.5 The Architecture: The Interface between Planner and Realiser

The interface between planning and realisation has been the subject of much debate. The 'generation gap' problem (see section 3.8) concerns how a planner can make decisions without knowing if they can be carried out at the linguistic level. This section will discuss the interface between the planner and plan-realiser in the architecture adopted in this work and discuss its relation to the more traditional generation gap problem.

In traditional architectures, planners take on a lot of responsibility: they are responsible for such tasks as accurately delimiting content and ordering. The realiser's task is less complicated, it merely has to produce surface NL from the

detailed input provided by the planner. The 'generation gap' problem can be serious if the realiser cannot follow these instructions because of the nature of the surface language. In this case either the planner has to make sure that its decisions can be realised at the surface level (the approach used in pipelined systems, section 3.8) or it receives feedback from the realiser (the approach used in interleaved systems, section 3.8). The 'generation gap', therefore, often causes problems in either efficiency (due to complex interactions between the planner and realiser) or in overloading the responsibility of the planner (so it has to know about linguistic issues).

In the architecture described here, responsibility is shifted away from the planner to the plan-realiser. In the absence of detailed instructions, or in the case of conflicting instructions, the plan-realiser can still make decisions on its own and should always produce a correct utterance. Of course, there is still no guarantee that the plan-realiser will be able to carry out all of the planner's requests. In this case the plan-realiser will have the 'final say' in which instructions will override others. This decision is based on the philosophical assumption that the communication is ultimately controlled by what constructs and words are available in surface language[2].

There are still however two alternatives which can be developed at the interface either of which may be utilised once a more advanced planner is built:-

- Pipelined interface. The plan-realiser can try to follow as many of the planning instructions as possible. In the case of conflicting instructions it will have default rules to determine which are more important.

- Feedback interface. The plan-realiser can consider alternative utterances, each of which conform to different subsets of the planner's instructions. The planner may then chose between these options according to the list of satisfied constraints that the plan-realiser presents for each alternative. This approach is different from the more traditional interleaved approach where

[2]Although with respect to NLE, the validity of the assumption is not important.

after receiving information back from the realiser, the planner has to re-plan the utterance.

5.2 Solution Detail : The Plan-realiser

The rest of this chapter will provide further details about the plan-realiser. This component of the LOLITA generator has been the subject of the majority of the work in this project and it has been successfully implemented (see chapter 7 for implementation details). It is important to note that these sections do not aim to present all the heuristics present in the plan-realiser: instead the overview of a broad range of heuristics aims to give the reader a taste of the plan-realiser's operation. Throughout the discussion relevant examples produced by the LOLITA generator will be included *in this font.*

5.3 Generation of Language-Isomorphic Concepts

Language-isomorphic (LI) concepts are those which can be 'adequately' (see section 5.4.1) described by a single lexical entry: they have a link in SemNet from the conceptual level to the linguistic level. These links are obviously language dependent: for the English generator described here, it is the **English_** link which is used (currently, the LOLITA system also has Italian, Spanish, French and Chinese concept-to-language links). Whether or not a concept is LI is also dependent on language. For example, the concept for 'lawn' is LI in English (and will have an **English_** link to the lexical item 'lawn') but not in Italian (there is no exact word in Italian for the concept of lawn), the lexical item 'molle' (meaning 'disgustingly soft' in Italian) has no corresponding lexical item in English (thus the concept for 'molle' is LI in Italian but not in English). If a concept is not LI in the chosen language then it must be decomposed into concepts that are. This approach is in contrast to work which assumes that concepts have a larger grain size than words and have to decide between lexical items in order to realise a particular concept

(see sections 1.5.3 and 3.9). Lexicalisation is somewhat similar in an aspect of the PENMAN system explored by [Sondheimer *et al.*, 1989] where, "if a concept does not have an appropriate lexical association, the algorithm generates a phrase with a more general head term and restrictive modifiers." (Note that 'concept' here is knowledge-base concept rather than the definition we adopt, see section 1.5.2).

In the case of exact synonyms, concepts may have a larger grain size than words and can be linked to more than one lexical item in a particular language. This is the case when the only difference in meaning is that of style[3] (for example, the only difference between 'cab' and 'taxi' is formality). In such a situation the generator may chose between synonyms so as to conform to the imposed stylistic constraints (via the realisation parameters) or randomly.

One current weakness of SemNet is the representation of concepts that can be expressed using phrases rather than individual words. There are often many ways of describing a concept by using synonymous phrases (for example, 'to die', 'to pass away', 'to kick the bucket') but the linguistic level in the LOLITA system is largely restricted to single root lexical entries and therefore such phrases cannot be expressed as easily as in lexicons such as Becker's phrasal lexicon [Becker, 1975] (see section 3.6.6). This weakness comes from the fact that LOLITA's representation is semantic rather than surface-linguistic based (for example, compared to the use of a phrasal lexicon in PAULINE, section 3.12.1 and the approach used in MTM based systems, section 3.12.4, where concentration is shifted to the linguistic level). It would be easy to modify LOLITA's representation and generator to provide a 'temporary' solution which allows the use of such fixed phrases but as discussed in section 3.6.6 there are also problems associated with this phrasal approach. An exception to the single lexical item restriction in LOLITA is for compounded verbs (which compound a verb with a preposition, for example 'go off' or 'blow up').

[3]Some may argue that such differences in style manifest themselves on the conceptual level, e.g., inherent style factors in a word influence their meaning. This approach is not adopted in the LOLITA representation.

5.4 Generation of Entities

This section will describe how concepts representing entities are realised.

5.4.1 How can an Entity be Adequately Described?

The SemNet representation adopted in LOLITA comprises a hierarchy of concepts representing entities. Theoretically, each of these entity concepts is defined by the whole of SemNet (section 1.5.3). However, it is of course impractical and unnecessary to realise the whole semantic network each time an entity is to be expressed. Instead, it is necessary to generate an expression which defines a particular entity in sufficient detail. For example, if an expression for a particular motorbike is required it would be usually insufficient to say 'a vehicle' (although this is true) whereas, for example, something like 'the red motorbike which is in John's garage' could be sufficient. It is a difficult problem to determine what is required to uniquely define a entity as it is dependent on context (e.g., if the object has been mentioned before, either explicitly or implicitly, then less description is usually required) as well as the user model (what the reader can infer or already knows etc). This problem while not being ignored completely has not been directly tackled in this work. It is assumed that in future the planner will be able to pass down instructions to help the plan-realiser produce adequate entity descriptions. What is important in this work, however, is that the plan-realiser must be able to cope correctly with such instructions and, when they are not available, be able to adopt reasonable default heuristics.

A recent attempt at generating adequate descriptions of entities is described by Reiter[Reiter, 1990] and concerns customising object descriptions according to the extent of the users' domain and lexical knowledge. He formalises the process by defining three constraints that a utterance must satisfy (based on Grice's maxims [Grice, 1975]):-

- **accuracy**: the utterance should be truthful.

- **validity**: the utterance should trigger the desired inferences in the hearer.
- **freedom from false implicature**: the utterance should not lead the hearer to draw incorrect conversational implicature.

Reiter illustrates these constraints with the example below:-

1. There is a shark in the water.
2. There is a dangerous fish in the water.

Reiter argues that on reading sentence (1), the knowledgeable hearer would not just infer that a member of a certain fish species was present in the water but would access her domain knowledge about sharks and recall that they were large, carnivorous and possibly dangerous. If, however, the reader did not have such domain knowledge about sharks, she may decide, for example, that it was still safe to swim in the water. For such a naive hearer a more explicit utterance such as (2) would be better. If the hearer did know about sharks however, sentence (2) may seem rather odd and in fact the hearer might draw the conversational implicature that the animal in question was not a shark (because she would have thought that if the fish was a shark, the speaker would have said so explicitly).

The default in the LOLITA generator is to generate the most specific realisation of the concept if it exists (i.e. if it is LI). However, there are some instances when, even though a certain way of expressing a concept is adequate, a paraphrase may be more appropriate. As in Reiter's example it might be better to generate 'dangerous fish' and not 'shark'. Reiter's algorithm makes the large assumption on the availability of a very rich user model in which every concept is marked as understood or not for a particular user: this would be easy to implement for systems with a few concepts but would be difficult to scale up to larger systems.

The problem of deciding when to chose an alternative paraphrase for a concept has not yet been tackled in this work. The decision will ultimately be a task which is distributed between the planner and plan-realiser:-

- In some cases the planner may decide that a concept should not be described directly (for example in the **do_not_describe** list in the **story** environment, section 5.1.3).

- If the plan-realiser is asked to express a concept which has no lexical realisation in the surface language (i.e. it is non LI) then it will have to find an alternative expression (see section 5.4.2).

- In the general case, because the planner has no linguistic information (and therefore does not know whether a concept is language isomorphic), it cannot make decisions about exactly how concepts are expressed. Instead the planner will pass instructions to the plan-realiser which may indicate whether or not to express a particular concept directly, but the final decision must be the responsibility of the plan-realiser. In the 'shark/dangerous fish' example, the planner may pass the information that clarity is essential but that the listener has a poor control of the particular surface language (in this case English) then the plan-realiser may well use heuristics to decide to paraphrase 'shark'.

These aspects will be the subject of further work but in the meantime the plan-realiser must be able to create paraphrases once this decision has been assumed (see section 6.8).

5.4.2 Realising Entity Concepts that are non LI

If a concept node is language isomorphic, and therefore has a link to a lexical item, then the plan-realiser can use that lexical item with the correct quantification. This quantification will be indicated by the **rank** control (see section 4.3.2) and range from the universal of a set, through to a bounded existential number (explicitly numbered or not) of a set to an individual or named individual. If a concept represents a set of entities then the root lexical item has to be pluralised. Morphological rules produce standard pluralisations from the root of the word whereas words with irregular plurals will be present in the linguistic part of SemNet.

If, however, a concept node is non LI and therefore has no link to a lexical item in a particular language, then the plan-realiser has to search for an alternative expression. If the node has more than one universal then it is an intersection of universal sets: the plan-realiser must move across the universal link to find the lexical items for each of these sets. Heuristics can then be used to order these names (e.g., adjectives will come before nouns). For example, a universal node with universals of the node for the concept 'big' and the node for the concept 'motorbikes'[4] will be realised as *'big motorbikes'*. This process may be recursive as even the universals of a particular concept may not be LI, the plan-realiser will have to recursively decompose the concept until LI concepts are reached. Additionally, an entity may be involved in an event which can be used to define that entity more fully. This information may be described using a relative clause or a 'special' relative clause. This is described further in sections 5.4.4 and 5.4.5.

Even if an entity concept is LI, it may be desirable to express it differently (see section 5.4.1 above). This can be achieved using abstract transformations (see section 6.8).

5.4.3 Determiners and Quantifiers

It is not usually sufficient to just realise a concept using a suitable lexical root item with the required morphology: a determiner may also be required. This section will describe different determiners, and discuss how and when the plan-realiser may use them.

The use of determiners is a complex linguistic issue and there are often cases when more than one determiner could be correctly used for the same concept in a particular utterance. As in other areas the plan-realiser does not necessarily have to be able to generate every case: what is important is that a correct determiner is always used. However, the more cases the plan-realiser can cope with, the more powerful it will be.

[4] i.e. a node which represents the intersection of the set of 'big things' and the set of 'motorbikes'

1. *'Men like cats'*: both subject and object roles have implicit universal quantification i.e, 'all men like all cats'.
2. *'Men drink liquids'*: the subject role has implicit universal quantification whereas the object has implicit bounded existential quantification i.e., 'all men drink some liquids',
3. *'Men die in wars'*: both subject and object roles have implicit bounded existential quantification i.e, 'some men die in some wars'.

Figure 5.2: Examples of implicit quantification associated with verbs

The determiner 'some'

When describing a set of entities which is not the universal set, it is sometimes necessary to explicitly indicate this quantification by using the determiner 'some'. Verbs have implicit quantification rules that govern the 'default' quantification of their subject and object roles [Garigliano and Long, 1988] (see figure 5.2 for examples). If the quantification of the subject or object roles in the meaning to be expressed is different from the default quantifications associated with the verb (which will be marked by a control in SemNet, see section 3.4) then the quantifications will have to be made explicit. Bounded existential sets can be indicated explicitly using the quantifier 'some', universal sets can be indicated using quantifiers such as 'every' or 'each'.

The definite article

Although there has been extensive linguistic work concerning the use of the definite article (see for example [Kramsky, 1972]) there seems to be very little in the NLG literature which explicitly lists rules for generation of articles[5].

There are at least six uses of the definite article in English (see [Garigliano, 1992]). These uses will be discussed together with notes on how and when they can be generated.

[5]Perhaps this is because many generators start from predicate logic which makes quantification explicit.

- The definite article is used to refer to a unique element in the external world or at least to a unique element in the common knowledge of the writer and reader. For example, *'The Moon' 'the Government'*. Concepts that are always unique will be indicated as such within the SemNet representation. For context dependent 'uniqueness', it will be up to the planner to mark concepts as being uniquely defined by the context.

- The definite article is used to show the uniqueness of a concept when it is defined in the sentence. For example *"the motorbike that I keep in my garage"*. When building relative clauses (see section 5.4.4) the plan realiser can use the definite article[6].

- The definite article is used to refer to something that has been introduced before and is unique in the focus of discourse. For example, *"I met a dog, the dog bit me"*. This use of the definite article is linked to the use of anaphora: if the plan-realiser cannot refer to a previous entry with a pronoun then it can use the definite article and an appropriate referring expression (see section 5.11).

- The definite article is used to refer to something that is implicitly unique in the focus of discourse. For example "I went to a restaurant. The waitress was pretty". For generation of this type of definite article, the planner will have to mark concepts as being in context. This can be done using 'scripts' [Schank and Abelson, 1977] containing information about commonly occurring situations. In this example the 'restaurant script' may contain information about such things as 'the waitress', 'the bill', 'the table' and 'the menu': each of these concepts may be marked as implicitly in context and the definite article used to express them.

- The definite article is used as a determiner for universal sets. For example, 'The horse is a beautiful animal'. This use can only be used for highly

[6]NOTE: The SemNet representation is normalised so that concepts with the same properties are grouped together. If, for example there were 'two motorbikes in the garage', there would be a concept for 'the motorbikes that are in the garage' which would have two specific instances.

structured relations such as 'is_a' and 'has_part' relations (e.g., it is unnatural to say "The horse does not feel well'). Currently the plan-realiser only uses the definite article in this way for realising 'is_a' relations.

- Finally, the definite article can be used in a situation where there is no 'script' but it is used to trigger one. For example 'I was looking for Russell. The office was empty'. This complex use of the definite article has not been considered in this work.

The indefinite article

If a concept has a singular rank and the definite article (or another article such as a possessive noun phrase, see section 5.4.5) cannot be used, then the plan realiser must use the indefinite article. The plan-realiser will use 'a' or 'an' depending on whether the following noun phrase (not the head of the noun) starts with a vowel.

Other determiners

In some cases, no article is required. This is the case for 'fixed' or 'continuous' concepts (marked with a control, see section 4.3.2) such as 'rain', 'flour', 'sugar' etc.

There is a variety of other determiners which could be used such as 'each', 'every', 'each and every', 'all', 'many', 'most' etc. Although these determiners are not yet covered they could easily be incorporated. Their use will be dependent on the realisation parameters (e.g., the length and colour parameters could have an affect on determiner choice), the type and required precision of the information to be conveyed (e.g., if the information is a defining clause then determiners such as 'each' and 'every' will be common) and the grammar of the surface language (the position of the determiner within the sentence structure will affect how natural a determiner will be).

5.4.4 Describing Entities with Relative Clauses

When producing an utterance to describe entities, relative clauses can be used to describe events in which the entity is involved (i.e., events in which the entity plays a role). As discussed in section 5.4.1, the problem of when a relative clause is needed to adequately define a concept will ultimately be the job of the planner: the planner's instructions about which events can be mentioned will be passed down to the plan-realiser. It could be possible to include heuristics in the plan-realiser that, in the absence of planning instructions, determine an ordering of possible relative clauses according to how much they define the entity. This, however, would be an example of bad integration (see section 1.2.6) as the plan-realiser would be trying to achieve something that is best left to another module (i.e the planner).

In the absence of instructions from the planner, the default operation of the plan-realiser is to generate relative clauses for entities depending on the information held in the relevant part of the SemNet representation (the plan-realiser cannot generate relative clauses when no information exists), the grammar and the value of the rhythm realisation parameter. The number of relative clauses allowed is dictated by the grammar. In theory a grammar may allow an infinite amount of perhaps nested relative clauses but in practice this would lead to very confusing sentences. In practice it is necessary to limit the total number and number of nested clauses that are allowed. This is achieved using the rhythm parameter which constrains the number and nesting of clauses to zero, one or two. It has been found (informally) that a greater number leads to complex and incomprehensible utterances. If an event has to be mentioned (indicated by the planning instructions), but cannot be said as a relative clause (constrained by the rhythm parameter) then a separate sentence will have to be constructed. The realisation of relative clause events is very similar to the generation of normal event clauses (see section 5.5): more details will be left until section 5.6. Relative clauses may be suppressed so that they appear later in the utterance. This prevents 'front-loaded' sentences and leads to more natural utterances. This will be discussed further in section 5.6.

5.4.5 Describing Entities with 'Special' Relative Clauses

Some events in which entities are involved cannot be expressed using normal relative clauses. These events are often internal events and, as such, need special rules for their realisation. Examples of these special clauses follow:-

- Possessive clauses.

  If an entity node (O1) is the object of an event (E1) which has the action **'to own'** or the internal action **possrelate** then the plan-realiser can use the Saxon genitive to express the event. The internal action **possrelate** is used when a Saxon genitive or possessive pronoun in the input has not been completely disambiguated - although it may be equivalent to the 'to own' action, it may correspond to a different semantic relationship (for example in the sentences *'The King's executioner'*, *'my executioner'* etc). Even when semantics is unable to further disambiguate the **possrelate** relation, however, the realiser can use the Saxon genitive.

  The plan-realiser will generate the subject of E1 (by recursively calling the plan-realiser with this node), followed by ''s' followed by the realisation of O1. Figure 5.3 shows a portion of the semantic network which represents (and is realised by) *'John's motorbike'*.

  An event with the action **'to own'** can also be realised as a normal relative clause (in the above example *'The motorbike that John owns'*). This is not the case for the **possrelate** action however.

- Noun co-locations.

  There are a host of internal events that can be realised using a co-location of nouns. For example:-

  - **is_part_of**: If an entity is the subject of an event with the action **is_part_of** then the object of that event can be used as a co-location before the entity. For example *'car bomb'*,*'computer screen'*.

```
* motorbike: 19868 *
 universal_:
   motorbike - 19862 - rank: universal
 object_of:
   event - 19871 - rank: universal
 concept_:
   motorbike - 19868 - rank: individual
 english_:
   motorbike - 19868 - rank: individual
*****************************************
John's motorbike
*****************************************

    * event: 19871 *
 generalisation_:
   ownership - 20946 - rank: universal
 subject_:
   john - 19845 - rank: named individual
 action_:
   own - 16943 -
 object_:
   motorbike - 19868 - rank: individual
 time_:
   present_ - 20989 -
*****************************************
John owns a motorbike.
*****************************************
```

Figure 5.3: SemNet representation for 'John's motorbike'

- **controls_**: Similarly, if an entity is the subject of an event with the action **controls_** then the object of that event can be used as a co-location before the entity. For example *'train driver'*, *'car mechanic'*.
- **is_in**: If an entity is the subject of an event with the action **is_in** then the location of that event can be used as a co-location before the entity. For example *'town square'*.
- **relate_**: As for the case of Saxon genitives, the LOLITA system is sometimes unable to fully disambiguate noun co-locations. In these cases SemNet uses internal events with the internal action **relate_**. Even though the meaning of the input noun co-location has not been represented correctly, the plan-realiser may still use a co-location to realise such expressions.
- **is_a** clauses as adjectives: If an entity is the subject of an event with an *'isa'* action and the object of that event is marked to be an attribute (with the **type** control) then the object of this event can be expressed as an adjective instead of a normal relative clause (e.g.,*'I called the warm taxi'* instead of *'I called the taxi that was warm'*).

5.4.6 Proper Nouns

Entities with a **rank** control of value **Named Individual** are realised as proper nouns. As a default, the plan-realiser currently assumes that a named individual is adequately specified using its name alone: no further specification using relative clauses is required. This is a simplifying assumption as the context will have a bearing on whether the name alone will be sufficient. In the future it will be up to the planning module to dictate whether further information will be needed.

The plan-realiser uses heuristics to order any multiple universals a named individual concept has (see section 5.4.2) of the entity (e.g., surnames last, *John Smith*; titles first,*'Mr Jones'*; locations last, *Downing Street, Central Park* etc.) and uses capital letters.

5.5 Generation of Events

Once expressions for entities can be generated, the relationships between them (expressed in SemNet as events) can be realised by linking them together. The different roles present in an event are realised as different clauses. English grammar allows a variation in the order of these clauses: the only order that English grammar dictates is that the subject comes before the verb (except in passive sentences, see section 5.5.2).

The planner may provide instructions which limit the number of clauses to be expressed or dictate clause order. In the absence of such instructions, and depending on the length and rhythm realisation parameters, the plan-realiser will generate all clauses (although it is unlikely that every clause will be present in a particular event) in the following order (where only the subject and verb clauses are mandatory):- certainty, time, subject, verb, object, co-subject, origin, destination, instrument, location and goal.

5.5.1 Generation of Actions

The most important part of an event is its action which is usually (but see section 5.7.2) realised as a verb. However, the approach to the generation of events is not verb-driven as in other work (for example, the incremental consumption approach to realising conceptual graphs [Nogier and Zock, 1992], section 3.12.2).

Generation of Non Language Isomorphic Actions

As for the case of entities, not all action concepts will be language isomorphic. If an action is not connected to a lexical entry in the chosen language then the plan-realiser will have to search for a paraphrase that is expressible. Unlike entities however, actions themselves do not form a hierarchy. Actions only have a conceptualisation within the framework of an event: it is the definitions of prototypical events which form a hierarchy (see section 4.3.2).

If an action in a particular event is non LI, the plan-realiser will have to first find the prototypical event for that action and then find the first prototypical event above that event that does have a LI action. The plan-realiser may use this LI action but, in order to express the extra meaning conveyed by the original action, the roles in the original prototypical event which differ from those in the LI prototypical event must also be made explicit.

Sometimes, even if an action is LI, it may be desirable to paraphrase it using the verb from a higher action together with restricting clauses. This is achieved using an abstract transformation which will be discussed further in chapter 6.

Subject/Verb agreement

Once a language isomorphic action has been found, morphology rules must be applied so as to ensure the correct subject/verb agreement. The plan-realiser will find the grammatical number (e.g., first person singular, etc) of the subject of the event and generate the correct verb endings. The rules for realising regular verbs are incorporated within the plan-realiser: irregular verb information is held in SemNet (at the linguistic level).

Tenses and other aspects

Time and tense representation in the LOLITA system is currently under development [Short, forthcoming 1995]. However, by using either an explicit tense carried in the **time_** slot of an event, or by using an algorithm to determine a correct tense by looking at the relative times of events, the following tenses can currently be realised: present (e.g.,*I see*), future (e.g.,*I will see*), past (e.g.,*I saw*), future perfect (e.g.,*I will have seen*), and past perfect (e.g.,*I had seen*).

In addition the following aspects can be realised in combination with the above tenses and each other: non_action (e.g.,*I do not see*), conditional (e.g.,*I would see*), hypothetical (e.g.,*I may see*), infinitive (e.g.,*I went home in order to see my mother*) and continuous (e.g.,*I am seeing*).

In complex events (see section 5.7) it may be necessary to override information in the **time_** slot and force an event to have a particular tense or aspect.

Clause order

The action of the event may affect the presence and ordering of other clauses in the event. Two examples follow:-

- Transitive and intransitive actions: A transitive action (marked with the **relation_type** control) requires an object clause to be realised whereas an intransitive action does not. If a transitive action is present in an event which does not have an explicit object link, the object will have to be inherited from an higher event in the hierarchy.
- Dative and non-dative actions: If the action is dative (marked with the **dative** control) then the plan-realiser can produce a dative grammatical construct: the destination clause will be realised before the object clause. The default is that actions marked as dative will be realised in a dative construct but this may be overridden by the dative/non-dative realisation parameter. Example:- *'I gave the cab driver a big tip'* (dative); *'I gave a big tip to the cab driver'* (non-dative).

5.5.2 Generating Event Roles

This section will describe how some of the roles associated with events can be realised. The algorithm to realise these roles is an example of procedural control within the plan-realiser. The plan-realiser will only attempt to realise clauses corresponding to roles if these roles are actually present in the input event. There may be cases, however, when a clause is required for a role that is not explicit in a particular event. In this case the role has to be inherited from an event higher in the hierarchy. This is most common for the subject, action, and in the case of transitive verbs, object roles. However, according to planning instructions this inheritance could equally well apply to any of the other roles.

Some of the individual clauses are discussed below. Note that there may be alternative methods of realising each clause (using different linking phrases for example) which could easily be added. The choice of which alternative to chose could be made either randomly or set via the planning instructions (for whatever reason).

- Time clause (the **time_** link). The time slot, as well as determining the required tense of the event, may refer to an explicit time or a time relationship with other events. For explicit times, simple heuristics are used to generate the correct linking phrases, (for example '**On** *Monday ..*', '**Last** *night*', '**At** *9pm*', '**In** *1995*' etc)[7]. If an event appears in a time slot, then depending on whether the event is to be 'opened' or 'closed' (see section 5.7.2), the phrases 'when' or 'at the time of' will be used (e.g., *'When the bomb exploded, the taxi was destroyed'* or *'At the time of the bomb explosion, the taxi was destroyed'*). After the linking phrase the plan-realiser is called recursively to generate the noun or event phrase.

- Certainty clause (the **certainty_** link). The certainty link can be added to events by LOLITA's analysis process (e.g., using inference methods such as analogy [Long and Garigliano, 1994], or source control [Bokma and Garigliano, 1992]) and is a measure of LOLITA's acceptability of the truth of an event. The plan-realiser uses phrases (dependent on their value of certainty, and the planning instructions) such as *'there is a slight chance that..'*, *'it is probable that..'*, *'it is odds on that..'* etc.

- Co-subject clause (**co_subject_** link). Co-subjects are realised using 'with' and a recursive call to the plan-realiser. For example *'I went with John to the Supermarket'*. (See also abstract transformations, section 6.5.)

- Origin clause (**origin_** link). Origin clauses are realised using 'from' and a recursive call to the plan-realiser. For example *'I received a kiss from Mary'*.

[7]When the time representation of LOLITA is improved these heuristics will be more precise as more information about the 'kind of time' will be explicit in the semantic representation.

- Destination clause (**destination_** link). Destination clauses are realised using 'to' and a recursive call to the plan-realiser. For example *'I gave a kiss to Mary'*. If the verb is dative however, the word 'to' can be omitted and the destination clause generated before the object. If the destination has already been mentioned and pronominalisation (see section 5.11) is allowed then 'there' can be used.

- Instrument clause (**instrument_** link). If the instrument is not an event then it can be realised using 'with' and a recursive call to the realiser. For example *'Brutus stabbed Caesar with a dagger'*. (See also abstract transformations, section 6.8.1.) If the instrument is an event then it can be realised using 'by' with a recursive call to the plan-realiser with the action forced to be in a continuous tense. For example *'people can book rooms by calling the hotel'*.

- Location clause (**location_** link). Location clauses are produced by realising the location with the correct preposition. Section 5.10.3 will describe the algorithm for generating such locations from LOLITA's representation of positions. For example *'Bolzano Hotel is near a cathedral and in a town centre'*.

- Goal clause (**goal_** link). If the goal is not an event then it can be realised with 'for' and a recursive call to the plan-realiser. For example *'I robbed the bank for money'*. If the goal is an event then 'so that' and a recursive call to the plan-realiser can be used. For example *'John married Jill so that his children had a mother'*. However, if the subjects are the same for the event being realised and the goal event the construct 'in order' can be used and the event generated with a forced infinitive. For example *'John married Jill in order to get her money'*.

The Passive Voice

Although the default is to produce sentences in the active voice, the realiser also has the capability to generate sentences in the passive voice. The choice of when

to do this will be made by the planner in order to meet stylistic constraints, aid coherence or in order to avoid the necessity of explicitly describing the subject of the sentence.

The passive voice can be generated if:-

- The action of the event is transitive.
- There is an explicit object associated with an event (the passive voice could be used with inherited objects but this would lead to unnatural sentences).
- The event is not marked as a command (see section 5.8.1).
- The action of the event is not a sentential verb which requires an 'open' event (see section 5.7.2). For example *'reporters suggested that a bomb exploded in Whitehall'* cannot be realised as 'A bomb exploded in Whitehall was suggested by reporters'.

An event is realised in the passive voice by saying the object of the event followed by the correct form of the auxiliary verb 'to be' followed by the correct form of the verb. The subject may then also be realised (according to planning instructions) using the linking word 'by'. For examples: *'The dog was kicked by a postman'*, *'The Sheriff was shot'*.

5.6 Generation of Relative Clause Events

As described above (section 5.4.4) entities can be defined in more detail by using relative clauses to describe events in which they are involved. As discussed in section 5.4.4 the problem of whether and when to express an event as a relative clause will depend on the planning instructions. If these do not exist the plan-realiser will use simple default heuristics to limit the number of clauses and permitted the number of levels of embedded clauses (depending on the rhythm realisation parameter).

Another default adopted by the plan-realiser is to generate relative clauses when they are first encountered. This leads to front loaded sentences with relative clauses being attached to entities near the beginning of the sentences. However, some heuristics are included so that events that may be mentioned later in a sentence can be suppressed. For example, if a relative clause is to appear in the goal slot of an event it can be suppressed. This would lead, for example, to the sentence *'the man married the girl in order to get her money'* instead of the more awkward *'the man who may get a girl's money married her in order to do this'.*

When generating relative clauses, the plan-realiser must first generate a relative pronoun according to the role the entity plays in the event to be expressed. Some examples of these rules are:-

- If the entity is the subject of the relative clause event then say 'that' or if the entity is marked (by a control) as animate say 'who'. For example, *'The car that won the race','The man who owned a motorbike'.*

- If the entity is the object of the relative clause event then say 'that' or if the entity is marked (by a control) as animate say 'whom'. For example, *'The car that you drive','The man whom Jane loves'.*

- If the entity is a position in the relative clause say the correct preposition followed by 'which' followed by the rest of the event (see section 5.10.3 on how prepositions are realised).

- If the event is an 'is_a' event use 'of whom' (or 'of which') followed by the subject of the 'is_a' event followed by 'are one' (if the subject is singular) or 'are members' (otherwise). For example, *'Mad men of whom Rasputin is one.'* or *'Companies that manage their hotels and of which Sogno is one.'*

- Other examples:- for destinations say 'to which' (or 'to whom'); for instruments say 'with which'; for goals say 'for which' etc. In the absence of any other rule to produce a pronoun, 'that' is used.

After the relative pronoun, the plan-realiser generates the relative clause using

similar rules to those that generate normal events. Some exceptions are:-

- The plan-realiser must keep careful track of the events that are currently being described or that have already been described. This will prevent information being repeated or sentences of infinite length being produced. For example, the plan-realiser will not produce a sentence such as 'the cat that sat on the mat **on which the cat sat**' as the embedded relative clause event (marked in **bold**) will not be described as it is already being mentioned.
- The plan-realiser does not have to mention the entity that is being described in the relative clause event in whatever role it appears. For example: *'the cat that sat on a mat'* not 'the cat that a cat sat on a mat', *'the mouse that the cat chased'*, *'the charity to which you gave some money'*.
- However, if the relative clause event is reflexive then a reflexive pronoun is needed in the object slot: For example *'the cat that licked itself'*.
- If there is an embedded relative clause (i.e., a relative clause inside another) then the original entity must be at least pro-nominalised in this embedded clause. For example *'People who want to destroy things that they hate'*.

5.7 Complex Events

Events in SemNet are not isolated and are often related to each other. Events may be connected with causal or temporal links or can appear as roles in other events. This section will discuss these possibilities and how they are realised by the plan-realiser.

5.7.1 Causal Links

Events which are the cause of, or are caused by other events are linked by the arcs **cause** and its inverse **cause_of**. The realisation of such related events also depends on the status of the events in question, particularly if they are **hypothetical** events

meaning that they may have occurred in the past or might occur in the future. Example heuristics are presented below:-

- Cause: to produce an NL expression for event E1 which has cause E2 (i.e., E1 is linked via **cause_** to E2).
  - If E2 is hypothetical, the events can be generated using the structure 'E1 if E2' with E1 forced to be conditional (unless E1 is in the future tense when this subsumes the conditional aspect) and E2 forced not to be in the future tense (i.e., if E2 is in the future tense then it is realised in the present). For example, *'You would like the motorbike if you knew that Mary owned it', 'The pavement will get wet if it rains'.*
  - If E2 is not hypothetical then the plan-realiser can use the construct 'E1 because E2'. For example, *'You like the motorbike because you know that Mary owned it','The pavement will get wet because it will rain'.*
  - If an event has more than one cause and planning instructions dictate that only one should be made explicit (via, for example, a Short-Length realisation parameter) but does not specify which one, then the plan-realiser will choose hypothetical causes in preference. This is because the hypothetical causes will usually convey the most important information. For example, taking E1 the hypothetical event *'I may go to London'* with cause E2 *'I want to see the Queen'* and the hypothetical cause E3 *'there may be a train'.* If no length restrictions are present the plan-realiser will be able to realise all the events with hypothetical causes before non-hypothetical ones, i.e., *'I will go to London if there is a train because I want to see the Queen'.* If, however, there are length restrictions but the planner does not give any instructions as to which events should be realised, *'I will go to London if there is a train'* is prefered (as the default) to *'I may go to London because I want to see the Queen'.* The planner will be able to override this default operation by indicating that the non-hypothetical event should be expressed in preference to the hypothetical one.

- Cause_of: to produce a NL expression for event E2 which has a **cause_of** E1:
  - If the event E2 is hypothetical then the plan-realiser uses the structure 'if E2 then E1' with E2 forced not to be in the future tense and E1 forced to be conditional or future. For example, *'if it rains then the pavement will get wet', 'if you knew that Mary owned the motorbike then you would like it'.*
  - If E2 is not hypothetical then the plan-realiser can use the construct 'E2 so E1'. For example, *'it will rain so the pavement will get wet'.*

5.7.2 Events within Events

In the *frames and slots* representation adopted by LOLITA (SemNet), events can appear in the role slots of other events (e.g., the object of an event could itself be an event). Events, and specifically these embedded events, should sometimes be expressed as a noun phrase (i.e, they should be kept 'closed', e.g., *'I saw the explosion'*), or sometimes as an event clause (i.e they should be 'opened' up, e.g., *'I saw the car bomb explode'*). It is not only the generation of the embedded event that changes: different linking phrases may also be required (e.g., in the case of generating phrases for the time slot, section 5.5.2. For example, **'At the time of** *the bomb explosion, ..'* compared to **'When** *the bomb exploded, ..'*).

The heuristics adopted to cope with this problem depend on the context of the generation, properties of the action in the main 'surrounding' event and properties of the embedded event itself:-

- The context: The context of the utterance can affect whether the default is to 'open' or 'close' events. This is relevant to the realisation of events in general rather than specifically to embedded events. If the utterance is part of a dialogue the default may be to keep events closed. For example, if the utterance is in answer to a question it would be more natural to express a closed noun phrase (e.g., user: 'what did you see ?', LOLITA: *'a bomb explosion'*). This default operation for a particular utterance will be set either by

the planner using a realisation parameter, or by whatever application is calling the plan-realiser (e.g., a particular slot filler in the template application, section 4.4.5, may prefer events to be closed rather than open).

- The action of the main event. Events which contain events in the object slot are signalled by the action being marked (using a control variable, section 4.3.2) as **sentential** or **infinitive**.

  There are three types of sentential verb (also distinguished with controls[8]):-

  - Those that require the event objects they take to be open (e.g., 'to know', 'to understand', 'to suggest', 'to think', 'to hear' in the sense of understanding). For example *'I know that*[9] *the man hit the girl'*, *'I heard that the bomb exploded'*, *'I think that the man died'*.
  - Those that require the event objects they take to be closed (e.g., 'to describe'). For example *'I described the hitting of the girl by the man'*, *'I described the bomb explosion'*, *'I described the man's death'*.
  - Those that are indifferent if the event object they take is transitive but require embedded intransitive events to be closed (e.g., 'to watch', to smell', 'to see', 'to hear' in the physical sense). For example *'I watched the man hit the girl'*, *'I watched the hitting of the girl by the man*, *'I watched the bomb explosion'*.

  Infinitive verbs require the object event to be open and the verb in the embedded event to be forced to be infinitive. For example, the verb 'to force': *'The policeman forced the crowd to go home'*.

- The embedded events themselves will sometimes be more naturally expressed as a noun (closed) or an event (opened). This will largely depend on the kind and type of the roles in the event. For example, if the event inherits most of its roles from the prototypical event for that action, then a noun phrase is usually prefered (e.g., *'the explosion'* compared to *'explosive devices exploded'*).

[8]It is important to note that these controls are not generation specific but are required in other areas of analysis (e.g., syntactic parsing).

[9]Note the word 'that' is optional in this case

On the other hand if an event contains explicit roles then it is often more natural to realise these using 'opened' events (e.g., *'In 1963, Oswald murdered Kennedy in Dallas'* is better than *'The 1963 murder of Kennedy by Oswald in Dallas'*).

If there is a conflict between what the verb of the main enclosing event requires and the naturalness of the embedded event then the following heuristics are used:-

- If the verb of the enclosing event requires an open event but the embedded event would normally be closed then the embedded event can be 'opened' using the correct form (i.e., tense and subject/verb agreement) of the verb 'to happen'. For example, *'The reported suggested that the explosion happened'.*
- If the verb of the enclosing event has chosen a closed event but the embedded event would normally be open then the realiser uses the continuous form of the embedded verb. For example, *'I described the man hitting the girl'*, *'I watched the bomb exploding'.*

5.7.3 Temporal Links

Events can be related to each other temporally. The plan-realiser must be able to make such relations explicit in the utterances it produces. At the present time however, SemNet's representation of temporal relationships is being improved [Short, forthcoming 1995]. For this reason related heuristics that are currently employed by the plan-realiser will not be discussed.

5.8 The Realisation of Commands, Questions and Answers

Applications may require that commands, questions and answers are generated by LOLITA (for example the 'query' application, see section 4.4.2). These types of

events, which are distinguished using the **status_** slot, require different realisation rules.

5.8.1 Commands

Command events are represented using normal events with the subject being the person LOLITA is talking to (i.e. the user) and a **status_** of **command**. These are realised by simply omitting the subject of the event. For example, *'give me a kiss'*, *'take the rubbish outside'*. According to realisation parameters the realiser could make the utterance more polite by adding 'please' or 'could you' etc.

5.8.2 Answers

Answers and, more generally, any response to an utterance (e.g., *'I do not understand'*, *'I do not know'*) are stored as explicit events in SemNet. Among other reasons, this is to allow the context of the dialogue to be kept for future reference. The node representing an utterance for an unknown answer, for example, would be represented as an event with LOLITA as a subject (which will be realised as 'I'), a non-action of 'to know' and the original question as the object (see section 5.8.3 below for discussion of embedded question realisation). A successful answer to a question is represented using the internal action **answer_of** with a **subject_** which represents the answer, an **object_** which represents the original question and a **cause_** which represents the evidence for the answer. There are a variety of ways in which these answers can be realised. One of which is simply to generate the subject followed by the **cause_** events. For example, in answer to the question 'Do I own a vehicle ?', *'Yes, you own a big fast motorbike.'*.

5.8.3 Questions

The plan-realiser must be able to produce utterances for various types of questions as described below. The type of question is obtained using the **status_** slot.

'To be' or 'to do' questions

These questions are represented as normal events with a **status_** of **question**. If the question is an 'is_a' question then the subject and the action are inverted (e.g.,*'Is the cat black?'*), otherwise an auxiliary word ('do', 'did', 'will' etc. depending on the required tense) is used before the normal realisation of the event with the tense forced to be present (e.g.,*'Did the cat chase the mouse?'*).

wh-questions

'Wh' questions (i.e., 'who, what, where, when, why' etc) are represented using normal events with the role to which the question relates marked (with a **status_** of **Unknown**). Depending on the question type, they are realised using a wh-word, followed by the correct auxiliary (i.e., 'to be', or 'to do', see above), followed by the normal realisation of the event with the omission of the role which dictates the type of question. For example, if it is the **cause_** that is marked as being **Unknown**, 'why' questions are realised (e.g.,*'Why did John hit Jill ?'*); if it is the location that is marked, 'Where' questions (e.g.,*'Where is the black cat'*); if it is the subject then 'What' or 'Who' (depending on the type of subject) is used (e.g., *'Who shot the Sheriff?','What is the meaning of life?'* etc.).

Recursive questions

Just as the rules for realising embedded events are different from those of normal events (section 5.7.2), so embedded questions have to be realised differently from normal questions:-

- Embedded 'Wh' questions are realised in the same way as normal 'Wh' questions except the auxiliary ('do','did','will' etc.) is not repeated. For example, *'Why did you ask why the man gave me a car?'* and not 'Why did you ask why did the man give me a car?'.

- For other (i.e., non-wh) embedded questions the linking word 'if' has to be used and the embedded question generated as a normal event. For example, *'why did you ask if the man gave me a car?'* instead of 'why did you ask did the man gave me a car?'.

5.9 Punctuation

There is little in the NLG literature concerning how utterances are realised with the correct punctuation. The LOLITA plan-realiser is able to cope with full stops, question marks and exclamation marks (useful in generating colour) at the end of sentences. Similarly, when producing lists of more than two clauses (ranging from simple list of nouns to a list of events) the plan-realiser is able to realise appropriate commas. More sophisticated punctuation, for example colons and semi-colons is not yet covered. A simple post-processor is used to check the output in order, for example, to ensure that full stops do not immediately follow other punctuation.

5.10 Generation of Special Portions of Semantic Input

In order to represent complicated aspects such as positions, time and other special relationships, SemNet adopts representations that cannot be realised in the normal way. The plan-realiser has to have special rules and heuristics in order to cope with these special SemNet constructs. The following subsections will give examples.

5.10.1 Internal Events

Some events in SemNet are represented using events which are not directly expressible in any language. They usually arise when input text has not been fully disambiguated and are distinguished by having an internal action role which does not correspond to a verb in the surface language (such as **is_a**, **relate_**, **poss_relate**,

has_part, controls_ etc.). These internal events can still be used to convey information during generation. The most common way in which these internal events are used is to generate special relative clauses (see section 5.4.5) such as possessive and noun co-location expressions. Normally the internal events themselves are not realised in surface language (when asked to produce an expression for these events the plan-realiser responds with *'internal event'*). Exceptions are:-

- Internal events with the actions **is_a** , **exist_**, and **is_in_state** can be expressed using the verb 'to be'. For example *'Rasputin is a mad man'*, *'The fan is off.'*
- Internal events with the action **poss_relate** which are usually used to realise a Saxon genitive (see section 5.4.5) can also be realised using the verb 'to have'. Although the exact relationship in these events is ambiguous the verb 'to have' conveys the same ambiguity. For example *'The King has an executioner'*, *'The man has two legs'* .
- Internal events with the action **is_in** are realised differently according to the type of the event's subject. If this subject is an event the verb 'to happen' is used (e.g., *'The explosion happened near Downing Street'*). If the subject is an entity then the verb 'to be' is used (e.g., *'The bomb was near Downing Street'*).

5.10.2 Time Representation

As mentioned previously (sections 5.5.1 and 5.7.3) the representation of time and the way in which events are temporally related is under development and has not been fully implemented. The SemNet representation of time will incorporate internal events which will require special realisation rules. Because these aspects are the subject of ongoing work [Short, forthcoming 1995] they will not be discussed further.

5.10.3 Positions

The representation of positions is another area which requires special SemNet constructs and rules for their realisation. SemNet builds explicit position nodes [Short and Garigliano, 1993][10] which can be realised in isolation, (e.g., *'On the wall near the fireplace', 'On a mat'*) as a relative clause (e.g., *'The picture on the wall that John painted'*) or as a location role in an event (e.g., *'The picture is on the wall', 'the cat sat on a mat'*). Position nodes (e.g., node 95999 in figure 5.4, representing *'on a mat'*) are the subjects of special internal events which have the internal_action refer_to_loc (event node 96000 in figure 5.4). The object associated with the position (in this example 'a mat') is held in the object slot of the event and the relative position of this object is held in special slots which indicate the range, the direction and the horizontal and lateral positioning between the position and the object with which it is referenced. The realiser must use the values of these measurements to generate an appropriate preposition: each combination of values has a list of prepositions which can be used to express that combination (examples of prepositions the SemNet can represent and the generator realise are 'between, in, on, upon, in front of, on the back of, on top of, around, near, next to, beside, outside, by, over, under, behind, far below, far above' etc). The generator can also realise more complex positions represented by SemNet such as *'Roberto placed a stone every 100 metres'*. For more details see Short and Garigliano [1993].

5.11 Generation of Anaphora and Referring Expressions

When an entity has already been mentioned either explicitly or implicitly (i.e., it is in context, for example due to a 'script') the rules for producing an adequate description (see section 5.4.1) will be different. Sometimes a simple pronoun will suffice and in other cases a shorter noun phrase will be required. There has been a

[10]This allows LOLITA to reason about positions explicitly

```
    * event: 96001 *
 universal_:
   event - 7688 - rank: universal - definition_
 subject_:
   cat - 95993 - rank: individual - suspended_
 action_:
   sit - 27678 -
 location_:
   position_ - 95999 - rank: individual - suspended_
 time_:
   past_ - 20991 -
**************************************
The cat sat on a mat
**************************************

  * position_: 95999 *
 universal_:
   position_ - 11456 - rank: universal
 status_:
   suspended_ - 29025 -
 subject_of:
   event - 96000 - rank: individual - suspended_
 location_of:
   event - 96001 - rank: individual - suspended_
**********************************
  On a mat.
**********************************

    * event: 96000 *
 universal_:
   event - 7688 - rank: universal - definition_
 subject_:
   position_ - 95999 - rank: individual - suspended_
 action_:
   refer_to_loc - 11497 -
 object_:
   mat - 95998 - rank: individual - suspended_
 status_:
   suspended_ - 29025 -
 range_:
   range_1_1 - 11472 -
**********************************
  Internal event.
**********************************
```

Figure 5.4: An Example of the SemNet representation of positions

great deal of work on handling anaphora (e.g., an overview [Hirst, 1981], referring expressions [Dale, 1990], the use of focus [Grosz, 1977]): however, this has largely concentrated on anaphora used for interpretation rather than generation. In generation, there is always the option of generating full references to events and entities without the use of anaphora and pronominalisation. This would lead to unnatural text, but it illustrates that the use of anaphora in generation is different to that in interpretation: in the latter case all possible uses of anaphora and pronominalisation will have to be interpreted while a generator can use anaphora only if the reader will be able to decipher it unambiguously.

Complex handling of anaphora at the paragraph level using context, focus, scripts etc. will require the help of the planner. However, in the absence of these instructions the plan-realiser must provide its own default heuristics in order to produce anaphora at the 'few' sentence level. The plan-realiser keeps a record (in the form of a stack) of all the entities and events to which it has referred (the planner may also place entities on this stack representing entities and events in context). If an entity is to be referred to again then:-

- If the entity is the only one of its pronoun class (i.e., singular male 'he', singular female 'her', singular un-sexed 'it', event 'this' and plural 'they') on the referred-to stack, then it can be pronominalised.

- If there is more than one entity on the stack with the same pronoun class then a pronoun cannot be used. Instead a shortened noun phrase is realised. At the moment the plan-realiser simply generates the head of the original noun phrase.

These heuristics are very oversimplified and can lead to ambiguous utterances and utterances where it would be more natural to use pronouns. For example, 'The red car and the blue car raced. The blue car won' would be better than *'The red car and the blue car raced. The car won'*, and 'The dog chased the cat. It barked' would be better than *'The dog chased the cat. The dog barked'*.

However, the simple heuristics are usually sufficient at this level and it would

be unwise to include more involved heuristics in the plan-realiser when they would be better left to the planner (see section on integration 1.2.6).

5.12 Conclusion

This chapter has discussed the architecture of the LOLITA NL generator and given details of some of the heuristics involved in the plan-realiser component. The implementation of these heuristics, together with many more that have not been detailed have resulted in a plan-realiser component that can successfully produce NL utterances from the SemNet representation.

Chapter 7 will discuss some of the issues involved in the implementation of the plan-realiser and the solution will be evaluated with respect to the project aims in chapter 8.

Chapter 6

The Solution: Abstract Transformations

Chapter 5 detailed the architecture of the solution to NLG adopted in the LOLITA system. It described the organisation of the *planner* and *plan-realiser* components before detailing aspects of the *plan-realiser*. This chapter will discuss another aspect of the solution: the use of *Abstract Transformations*. These transformations act on the SemNet input before it is passed to the plan-realiser. They can be invoked (for example, by planning instructions) between sentences in an utterance or between clauses in a sentence, to alter the SemNet input to the realiser and therefore change the utterance produced. Abstract transformations move from *normal forms* of SemNet representation to alternative forms which represent the same or very similar meaning. The application of such transformations leads to the ability to express events in different ways. Besides being more natural and 'human-like' abstract transformations can help satisfy stylistic constraints.

The ability to produce such paraphrases could be incorporated in the plan-realiser module itself (for example, by adding heuristics to search the SemNet input differently). However, abstracting these transformations away from the plan-realiser takes away some of the responsibility from the plan-realiser and leads to a more modularised and better integrated solution (section 1.2.6).

The *normal forms* used in LOLITA's SemNet representation are not as restricted as other normalised forms (for example Schank's CDT, see section 3.12.1, where a very restricted set of about twenty primitive actions are employed). However, as there are many ways of expressing an event without significantly changing the meaning, it is advantageous to have a normal form from which the generator can produce more than one utterance. This work, although approached from a generation viewpoint, is also applicable to NL understanding when normalisation (see section 4.3.1) concerns mapping different SemNet representations onto a normal form.

This chapter describes various abstract transformations, with discussions on why particular normal forms are chosen, the rules which allow us to move away from these normal forms and what effect the transform has on the final utterance (apart from the obvious reason that variations are more natural). Before discussing these abstract transformations, a summary of other relevant work in this area is provided.

6.1 Other Work at this Level

Various other researchers have included a similar level of manipulation in their solutions to generation.

The CYC Project

Work by [Barnett and Mani, 1990] on generation in the MCC (Microelectronics and Computer Technology Corporation) 'Large Common Sense Knowledge Base' project (CYC) includes a manipulation process called *goal revision* which involves the modification of semantic representation before it is realised. Although other aspects of generation are very different from those adopted in LOLITA (CYC for example is predicate calculus based and its generation uses a unification grammar, see section 3.6.2), *goal revision* is similar to the use of abstract transformations.

Barnett also recognises that such a process should allow for bi-directionality to allow goal revision to be used in both interpretation and generation. Whereas the abstract transformations described in this chapter concentrate on generation aspects, Barnett's work has concentrated on interpretation. Moreover, the type of transformations Barnett considers are different (and more restricted) to those presented in this chapter. Barnett concentrates on problems of noun compounding (e.g., the representation of a 'Lisp machine') and that of metonymy (for example 'I read Shakespeare'). Noun co-location in LOLITA is represented by different relate actions (see section 5.4.5) which are generated as special relative clauses. LOLITA does not yet consider figurative methods such as metonymy. However such metonymy could be achieved using an abstract transformation which, for example, allows the substitution of an artist's (e.g., composer, painter, author etc.) work by the artist's name (e.g., 'I like listening to Beethoven', 'I studied Shakespeare').

Barnett argues that other researchers solve some of these problems (i.e. noun compounding, metonymy, use of de-lexical verbs) by having separate entries in the lexicon and that this would cause the problem of lexical explosion and the inability to cope with novel cases. This argument is specially valid for large-scale NLE systems where it would be infeasible to contain lexical entries for every possible noun compound, use of metonymy, or de-lexical structure.

The KING Generator

Many abstract transformations rely on the depth of knowledge about concepts held in SemNet. Jacobs also advocates such a 'knowledge intensive' approach in his KING generator [Jacobs, 1987] (see section 3.12.5). Jacobs' paper, however, concentrates on how this knowledge is represented rather than how it is used to generate variation. His intensive knowledge about different events allow them to be expressed or *'VIEWED'* from different angles. Section 3.12.5 has shown KING's representation of commercial transfer events (i.e. the representation of 'buying' and 'selling' etc). Cline's KALOS system [Cline, 1994] (see section 3.12.3) also advocates such a knowledge intensive representation but this is used in a final

I like hot curries → I do not dislike hot curries
(or:- I really do not dislike hot curries. I do not dislike curries at all.)

Figure 6.1: Example of antonym substitution abstract transformation

revision process rather than transformations that occur before realisation.

MTM

The use of lexical functions in systems which are based on MTM (see section 3.12.4) also lead to similar paraphrases. Lexical functions, as their name implies, work purely on the lexical or surface level: even semantic information such as the 'actors' in particular events are stored as lexemes. Furthermore, the use of lexical functions still require the lexical information to be made explicit for every case. One lexical function (*Oper1*), for example, returns a de-lexical structure for a particular verb (see 3.12.4) but the possible de-lexical structures for each verb must still be explicitly stored in the 'Explanatory Combinatorial Dictionary' (ECD). Again this information rich lexicon may work well for small systems with a limited vocabulary but would lead to lexical explosion for larger-scale systems.

6.2 Substitution of an Antonym Action

The first type of abstract transformation to be considered can be performed when the action concept of an event is deemed by the semantics to have an antonym. The action link in such an event can be negated (i.e. actions made into non-actions and vice-versa) and the action replaced by its antonym (see figure 6.1 for an example). The normalised form has been chosen to be events which have actions rather than non-actions. The effect of the generator choosing to perform such a transformation is often to under-exaggerate a particular event by negating its opposite. If however, mode modifiers such as 'at all' or 'really' are added to the negation of the antonym, the effect is the opposite and events can be exaggerated (Hovy [Hovy, 1988b] also considers 'imputing' affect or bias on text in this manner).

Copula verbs:- to be, to feel, to appear, to become, to look, to seem, to smell , to taste etc
Kennedy is dead → Kennedy is not alive
Velvet feels smooth →Velvet does not feel rough
(or:- Velvet does not feel at all rough)

Figure 6.2: Examples of copula verbs and copula action abstract transforms

Some confusion may arise when it is unclear if an antonym pair forms a partition of all possible states. It could be argued, for example, that between the areas of 'dislike' and 'like' is an area of neutrality. Whether this transformation preserves meaning will depend on the semantics: if a complete partition is deemed to exist then this abstract transformation may always be applied. If not, the decision on whether to perform such a transform will depend on the level of precision required.

6.3 Transformations on Copula Actions

Another very similar transformation can be carried out on events with copula actions. Copula actions are those which take complements, the most common example being the verb 'to be'. Other examples of copula verbs are give in figure 6.2, together with examples of transformations of this type. If the complement these actions take (which in SemNet are held in the object slot) have an antonym, then the complement can be replaced by this antonym and the action of the event negated (i.e. actions made into non-actions and vice-versa). As previously stated, the normalised SemNet will have events with actions rather than non-actions and the effect of performing this type of transformation can again be under-exaggeration or, with the use of modifiers, exaggeration. The problem of having neutral states (as discussed in section 6.2, above) is also applicable here.

6.4 Transformations on Complemented Verb Pairs

Some actions which describe a transfer from an origin or to a destination have a complement which can be used to describe the same event in the different (i.e.

(1) John bought a car from the salesman → The salesman sold a car to John.
(2) I gave the dog a bone → The dog received a bone from me
(3) * I gave the dog a bone → The dog took a bone from me
(4) * The vicar gave five pounds to charity → The charity took five pounds from the vicar

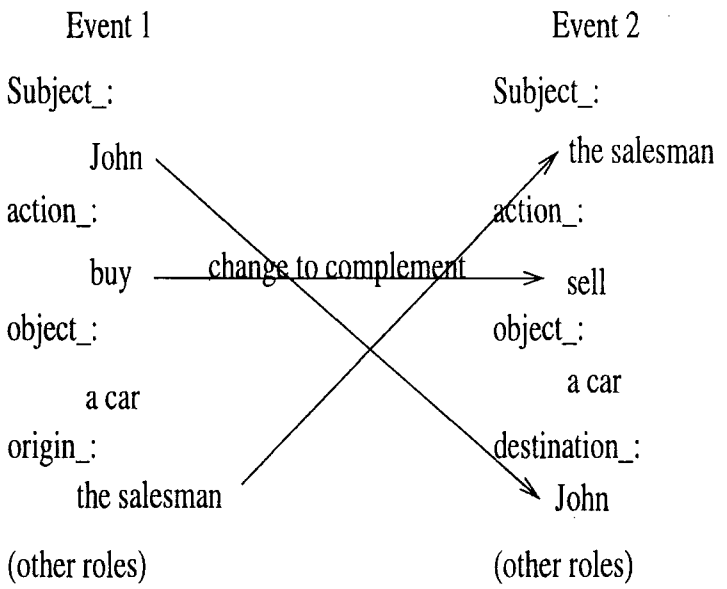


Figure 6.3: Examples of complemented action pair transformations

opposite) direction. An abstract transformation can be performed by changing the action (or non-action) of an event to its complement and swapping the various role arcs in the following ways:-

- If the original event describes a transfer from an origin role, make this origin the subject role and change the original subject role to a destination role (see figure 6.3).
- If the original event describes a transfer to a destination role, make this destination the subject role and change the original subject to a origin role.

It has been chosen (arbitrarily) to have those events with origin links as the normal form held in SemNet. This means that the generator needs only to consider transformations in one direction. Apart from creating variety in its output, the generator may chose to make such a transform in order to stress the original origin (or give less stress to the subject).

Examples of action complement pairs (which are linked by a **complement_** link) are 'to buy'/'to sell' and 'to give'/'to receive'. Examples of different sentences resulting from a complement pair transform are given in figure 6.3.

This transformation is only strictly valid if the 'family' (marked by a control, see section 4.3.2) of the subject and destination or origin roles are of a compatible type. Otherwise the meaning of the transformed sentence may be slightly different or even incorrect (for example *'I bought some food from the supermarket'* $\rightarrow$ *'The supermarket sold me some food'*). Again, a precision flag set by the planner will indicate if these less precise transformations are allowed. In some instances it would seem that two actions form a complement pair when, in fact, they do not. For example, it may seem that the pair 'to give'/'to take' in example (3) would be more natural than 'to give'/'to receive' in example (2). However 'to take' usually implies that its subject has an active part in obtaining the object. So although, example (3) seems a valid abstract transform, example (4) shows that these actions do not form a strict complement pair.

John, Mary and Sue went to the supermarket →
John and Mary went to the supermarket with Sue →
John went to the supermarket with Sue and Mary

Figure 6.4: Example of a multi-subject transformation

6.5 Transformations on Multi-subject Events

When an event has more than one subject, any combination of these subjects may be expressed as a normal subject or as a co-subject. The plan-realiser normally (i.e. in the active voice, see section 5.5.2) generates subjects before the verb of an event and the co-subjects after the verb (using the word 'with', see section 5.5.2). Thus by changing some of the subject links to co-subject links, different emphasis can be placed on each of the original subject roles. The only constraint on the combinations of subjects which can be transformed in this way is that at least one original subject must remain as a subject (so the resulting event does not have an empty subject). The normalised version of the semantic network contains multiple subjects rather than co-subjects: thus the transform must only decide which of these subjects to make into co-subjects rather than vice-versa. See figure 6.4 for an example of a multi-subject transformation.

6.6 'Give' Related Transformations

A further group of transformations can be applied to events which can be expressed as the transfer of an object or of an event. In these events the object or event which has been transferred can be mentioned explicitly, or be inferred by the action (see figure 6.5 for examples).

Unlike the other abstract transformations described, there is not one universal normal form for these type of events; the normal form for a particular event depends on the object or event which is transferred. If this object or event only has a meaning when bound to this particular event, then the normal form is that which implicitly describes the object or event in the action. This is because a separate

(1) John kissed Mary (NF) → John gave Mary a kiss
(2) John punched Mary (NF) → John gave Mary a punch
(3) John beat Mary (NF) → John gave Mary a beating
(4) John promised Mary → John gave Mary a promise (NF)
(5) John lied to Mary → John told Mary a lie (NF)
(6) * John poisoned Mary → John gave Mary some poison
(7) John poisoned Mary → John administered poison to Mary (NF)

Figure 6.5: Examples of 'give' related transformations

node representing the transferred object or event is not required in the SemNet representation. If however an object is transferred which can be referred to and have a meaning independent of the original event then it must have a separate node in the semantic network. The normal forms of the example sentences in figure 6.5 are marked with (NF). In the first three pairs of sentences, 'the kiss', 'the punch' and 'the beating' that 'John gave to Mary' are bound uniquely to these events whereas 'the promise' and 'the lie' can be conceptualised without the original events (e.g., 'the promise' or 'the lie' could be described as another event). The last example is of interest as although the sentence 'John gave Mary some poison' seems to fit into the pattern described above, the semantics must differentiate between actual poisoning events and events which involve the transfer of poison (e.g., 'John gave Mary some rat poison'). To conceptualise an actual poisoning, therefore, the semantics must make explicit the fact that poison is administered. This normal form may be then generated as (the more natural) 'John poisoned Mary'. Jacobs [Jacobs, 1987] (see section 3.12.5 and 6.1) also describes how similar 'give' variations can arise in his KING system (the example he uses is 'to give a hug'). However a special entry in KING's representation for the concept of 'hug giving' is required and thus the problem of lexical explosion and scale-up is not really solved.

Because there are two alternative normal forms, there are two kinds of transformation which work in opposite directions from the normal form to the alternative form. The algorithm for one of these transformations, where the transferred object is not explicit in SemNet, is discussed in the next subsection. The algorithm which works in the opposite direction is similar.

6.6.1 Making an Implicit Object Explicit

The transformation can take place if the following conditions exist:-

- The event(E) has an action that can be represented with a transitive verb.
- The prototypical event(PE) in which this action appears either has a sub-event(SE)[1] (e.g., in figure 6.5, example (1), the node representing 'kisses') or is identified by a word ending in 'ing' (e.g., 'beating' in (3)).

The transformation steps are as follows:-

- Make an entity node(N) with its universal being either the sub-event(SE) of the prototypical event(PE) (e.g., 'a kiss') or the prototypical event itself (e.g., 'a beating').
- Change the action of the event to a 'give' related action (see below).
- Make a destination link in the original event to the original object of that event.
- Change the object of the original event to the newly created node (N).

Examples of other actions which can be used in similar ways are:- 'to administer','to apply', 'to deal', 'to deliver', 'to hand over' etc.

Further variations can be achieved if the original event has **mode_** roles which can be expressed as adverbs. In this case, the information contained in this role can either be left as an adverb to the new 'give related' verb (i.e. left as a mode link in the new event) or made into an adjective of the explicit noun (by making it a universal of the new node in the semantic network). For example, the semantics representing the sentence 'John quickly kissed Mary' can be transformed into either 'John quickly gave Mary a kiss' or 'John gave Mary a quick kiss'.

[1]Note: a sub-event link is required in SemNet for other areas of processing: it does not appear specifically for this kind of abstract transformation

6.7 Other De-lexical Transformations

'To give' is an example of a verb which can be used in a de-lexical construct. De-lexical verbs are those that add very little meaning in themselves so that most of the meaning is given by the noun which is the object of the verb. The function of de-lexical verbs is therefore to provide a verb for the structure and very little else; there is often an equivalent verb which can be used instead. Other examples of de-lexical verbs are 'to have' (e.g., 'to have a bath'), 'to take' (e.g., 'to take a shower'), 'to do' (e.g., 'to do some shopping') and 'to make' (e.g., 'to make a decision'). Although the number of these verbs are small, they contain some of the most common verbs in English and de-lexical structures are very abundant in everyday English.

Actions used in their de-lexical capacity will never appear in SemNet when there is an alternative. That is, the SemNet normal form is the form which does not contain actions which are realised using de-lexical verbs[2]. It is important to note, however, that de-lexical verbs always have another meaning which is not de-lexical (e.g., 'have' in the sense of ownership, 'make' in the sense of construct or produce). For example, in the phrase 'to make a noise' the verb 'to make' does not fall in the adopted definition of de-lexical verb so it can appear as a normal form in SemNet.

Some phrases which use de-lexical verbs have adjectives attached to the object phrase which cannot be used as adverbs (e.g., 'Mary had a hot shower'). In these cases however, there is always an alternative way of semantically conceptualising the event in normal form without the use of de-lexical action (in this case 'Mary showered with hot water'). In fact, any portion of semantics which uses a de-lexical action will not be sufficient to conceptualise the required meaning.

Other approaches to the handling of the interpretation or generation of de-

[2]However the normalisation process in LOLITA is not yet good enough to convert all de-lexical structures into a normal form. This work on generation of de-lexical verbs aims to be bi-directional and thus will be beneficial to the normalisation process. In the meantime, the realiser can cope with de-lexical actions in the SemNet input by treating them as normal events.

lexical verbs involve either incorporating de-lexical structures in the lexicon (e.g., the phrase "to have a walk' is present in the lexicon as well as 'to walk') or marking each verb as suitable for specific de-lexical structures. The first solution will quickly lead to lexical explosion as the scale of a system is increased; the alternative 'marking' process will be laborious and unable to deal with novel or previously unseen constructs. The methods of Natural Language Engineering (section 1.2.9) advocate the use of rules wherever possible. No matter how ad-hoc and contrived these rules may appear, if they are applicable to a wide range of cases then they are valid. Section 6.7.3 will provide an informal cost-benefit analysis to support such a rule based approach compared to that of explicitly marking individual verbs according to the de-lexical structures they can use.

Apart from creating stylistic utterances or producing variations, it appears that de-lexical verbs are often used either to single out a specific event or to emphasise that the subject is taking part in the event. Using this hypothesis we can find rules which allow or disallow the use of de-lexical verbs in different cases. The next subsections discuss possible abstract transformations for the de-lexical verbs 'to have' and 'to make' and how the above hypothesis is used to govern when they can be used. Rules for other de-lexical constructs are in various stages of development and implementation and are a possible subject of further research. In the meantime however, the temporary solution of explicitly marking verbs with possible de-lexical structures could be utilised (especially if an application is to work in a limited domain and requires a small lexicon).

6.7.1 Example of De-lexical Rules for 'to have'

The verb 'to have' is perhaps the most commonly used de-lexical verb. Its uses can be categorised into two classes depending on whether the action it replaces is intransitive or transitive: only the former case is discussed here.

If the event to be expressed has an intransitive action and the prototypical event for the action is language isomorphic (see section 1.5.3) then the de-lexical verb 'to have' can often be used. The 'naturalness' of the resulting utterance will depend

(1) I walked to the shops → I had a walk to the shops
(2) I showered → I had a shower
(3) I laughed at the clown → I had a laugh at the clown
(4) John died → John had a death (*)
(5) The prisoner escaped from gaol → The prisoner had an escape from gaol (*)
(6) The prisoner escaped from gaol luckily → The prisoner had a lucky escape from gaol

Figure 6.6: Examples of natural and unnatural uses of the de-lexical verb 'to have' with intransitive verbs

on the repeatability of the event[3]. If the event is highly repeatable (for example, strolls, baths, showers, walks, laughs etc) then the use of the de-lexical verb is common. Unrepeatable events or those which are usually unrepeatable and 'one-off' lead to very unnatural de-lexical phrases (for example, 'deaths' , 'weddings', 'births', 'escapes' etc.) and should not normally transformed in this way. The hypothesis above claims that one use of a de-lexical verb is to single out specific events. If however an event is unique and un-repeatable it is not necessary to single it out and the use of the de-lexical construct is not natural.

There are two exceptions to this general rule. Firstly the semantics may indicate that a normally un-repeatable event is repeatable in the current context (for example an escapologist may talk about 'having escapes'). Secondly, if the event is modified with an adjective then natural expressions such as 'to have a lucky escape', 'to have a gruesome death' will result. These constructs are in fact often more natural than their corresponding non-de-lexical forms (e.g., 'I escaped luckily' or 'I escaped with luck'). Examples of the 'to have' de-lexical transform are given in figure 6.6.

[3]The repeatability of an event will be marked using a control on the prototypical event. Again this control is not added specifically for this type of transform: it is needed for other areas of the LOLITA system

(1) Scott attempted to reach the South Pole → Scott made an attempt to reach the South Pole
(2) Columbus claimed that the Earth was a globe → Columbus made the claim that the Earth was a globe
(3) Jack arranged to meet Jill at the top of the hill → Jack made an arrangement to meet Jill at the top of the hill
(4) I thought that the woman was beautiful → I made a thought that the woman was beautiful (*)
(5) I did not believe that ghosts existed before I saw one! → I did not make a belief that ghosts existed before I saw one ! (*)

Figure 6.7: Examples of natural and unnatural uses of the de-lexical verb 'to make' with sentential verbs

6.7.2 Example of De-lexical Rules for 'to make'

If the action of an event is sentential (i.e., it takes an event as an object) and the object event it takes is language isomorphic[4], then the de-lexical verb 'to make' may be used. There are again exceptions which can be treated by returning to the hypothesis that a de-lexical structure can be used to emphasise the subject of a particular event. There is a class of actions which are in themselves very personal and already emphasise that the subject of the action is the one doing the action (e.g 'to hope', 'to dream', 'to think', 'to believe'). This means it is unnecessary and unnatural to further emphasise the subject of these verbs by using the de-lexical 'to make' construct. However for such personal verbs, the de-lexical verb 'to have' can often be used instead (e.g., 'I had a dream that ..').

If applicable, the abstract transformation can be applied by constructing a *de-lexical event* in the semantics with the same subjects as the original event, the action 'to make' (and relevant time information so as the correct tense results) and the original event as the object. It is this newly constructed event which can then be passed to the realiser resulting in transformations exemplified by the sentences in figure 6.7.

[4]Note: the name of the event must be language isomorphic, not the action of that event.

6.7.3 Cost-Benefit Analysis of the Rule Based Approach for Handling De-Lexical verbs

Cost-benefit analysis is an important aspect of NLE (see section 1.2.10). Although extensive formal cost-benefit is not always necessary or practical, a required criterion for success is that informal investigations into possible solution alternatives is performed. This section will give an example of such analysis by comparing the costs and benefits of adopting a rule-based approach to the generation of de-lexical structures (as discussed in the previous sections) with the alternative of explicitly adding information about which verbs can be used with which de-lexical structure into SemNet.

The values of the parameters used in this analysis are estimated. To determine accurate values would in itself require a good deal of work. This is an example of the cost of cost-benefit analysis itself (see section 1.2.10).

Marking Individual Verbs

Costs:-

- Data Entry. Every verb in SemNet will have to be analysed by hand to see if it can be used with any de-lexical structure. For each of these verbs, further analysis will be required to ascertain exactly which de-lexical structure it can use. For each of these possible de-lexical structures a link will have to be added to SemNet. A estimation of the time costs is as follows:-
  - Initial analysis of every verb node in SemNet. Approximately 10,000 nodes at about 1 minute a node = **approx. 4 person weeks**[5]
  - Analysis of which structures can be used with each suitable verb. Estimation of 1000 verbs which can take de-lexical structures at about 5 minutes a node = **approx. 2 person weeks**

[5] 1 person, 8 hours a day, 5 days a week

  - Adding the links to the data. Estimate that each of the 1000 verbs can take an average of 3 different de-lexical structures. Therefore 3000 links at about 1 minute a link = **approx. 1 person weeks**
  - **Total time cost = 7 person weeks**

- Memory Cost: The memory cost of adding 3000 links = **approximately 200k.**

- Search Cost: The extra links in SemNet will be an extra burden on the various search algorithms employed in LOLITA. When the new links are encountered they may have to be followed or at least checked to see if they should be ignored. This cost may well be significant under a complexity point of view, especially as de-lexical verbs are very abundant in NL.

- Robustness Cost. The significant data entry task described above will be prone to mistakes which will cause LOLITA to generate unnatural de-lexical structures or prevent it from generating valid ones. To ensure the data is correct it would have to be either checked and rechecked or corrected for each individual verb as errors were found. Furthermore the method will not allow for new or novel cases.

Benefits:-

- Development Benefits: No time will be required to design, implement and improve rule-based algorithms.

- Execution Benefits: the generation of a de-lexical structure will require little run-time cost. A simple lookup process will be needed to determine which de-lexical structures can be used for a particular verb.

Rule Based Approach

Costs:-

- Design Cost: The cost of designing and implementing algorithms for each de-lexical structure estimated at 3 days. For the de-lexical verbs 'give, have, take, do and make' development time will be less that the explicit verb marking approach = **3 person weeks**.

- Memory Cost: No extra links will be require in SemNet. The controls and links required for the algorithms are required for other aspects of LOLITA.

- Search Costs: because no new links are added, various search algorithms in LOLITA will not be affected.

- Robustness Costs: It is unlikely that the initial algorithms will cover all cases. As exceptions are discovered the algorithms will have to be developed so as to cover these cases. Although initially the rule-based approach may not cover as many cases as the explicit data approach, rule development when a individual exception is found will cover more than one case.

- Execution Cost: the cost of applying the de-lexical rules will be slightly more than in the data approach. However as can be seen in the example algorithms (sections 6.6 and 6.7.1), only 2 or 3 inexpensive look up processes will be required.

Benefits:-

- Preparation Benefits: no tedious and time consuming data entry and checking will be required.

- Theoretical Benefits: the development of rules will give a better theoretical understanding about the use of de-lexical structures.

- Development Benefits: as mentioned above, when a missing case is discovered rather than just adding data to cover this particular case, rules can be built so as to cover other similar cases.

Conclusion

Informal investigations into the cost and benefits of each approach together with estimations of the parameters involved have clearly shown that the rule-based approach is favoured. Although the rule-based approach is slightly more expensive at runtime and initially will not cover as many cases, it is cheaper with respect to preparation time, memory and search times. Furthermore, the rule-based approach will provide an insight into the use of de-lexical verbs which the simple one-off data entry approach would not.

6.8 Generalisation or Specialisation of Concepts

Another way of producing variation or generating style is to paraphrase the utterance by describing entities and actions in different ways.

Because of the nature of the SemNet input (more particularly the relationship between concepts and words, see section 1.5.3) and the architecture of the solution (see chapter 5), the plan-realiser already has the ability to find paraphrases. All that the abstract transformation is required to do is to temporarily mark an entity or action as being non-language isomorphic (non-LI) by removing the relevant language link. As the plan-realiser will not be able to find a lexical item for that concept, it must find a paraphrase (see sections 5.4.2 and 5.5.1). The plan-realiser will find a more general concept which is LI and realise this concept together with defining information which conveys the difference in meaning between the original and more general concept. Examples of such paraphrases are shown in figure 6.8. The success of these transformations is dependent on the amount and quality of data in SemNet: information which defines the differences between a general and more specialised concept has to be present.

The normal form in SemNet is to use the most specialised concept. However, the process of obtaining the normal form (normalisation, see section 4.3.1) is not often straightforward. It is useful therefore, to investigate how to move in the opposite

(1) remove LI link to 'motorbike' → 'motor vehicle with two wheels'
(2) remove LI link to 'murder' → 'unlawfully kill'
(3) remove LI link to 'sleet' → 'rainy snow'

Figure 6.8: Example generalisation paraphrases

(1a) I wounded you with a gun → I shot you
(1b) I wounded you with a hand gun → I shot you with a hand gun
(1c) I wounded you with a knife → I stabbed you
(1d) I wounded you with a knitting needle → I stabbed you with a knitting needle

(2a) I stuck the wood with glue → I glued the wood
(2b) I joined the wood with Bostick → I glued the wood with Bostick
(2c) I attached the wood with nails → I nailed the wood

(3a) I called you on the 'phone → I 'phoned you
(3b) I called you on the radio → I radioed you

Figure 6.9: Examples of verb specialisation by instrument clause

direction and consider abstract transformations which lead to the replacement of concepts with more specialised ones. These abstract transformations can be used in either the normalisation stage of interpretation, or in generation to produce a 'normalised' utterance from a portion of SemNet which is not already normalised. An event, for example, may contain enough information to allow a move further down the event hierarchy (see section 4.3.2) to find a more specialised action. The information which allowed this action substitution may then be dropped. A subset of this kind of transformation are those which involve instrument roles; these are discussed in the following subsection.

6.8.1 Action Specialisation

An instrument role in an event can often allow the action of the event to be substituted by a more specialised one. Example sentences resulting from this transform are shown in figure 6.9.

The algorithm for this transform is now described using figure 6.10 as an example of simplified SemNet input:-

- Find the prototypical event for the original event (e.g., for the event 'I wounded Jill with a gun'(E1), the prototypical event for woundings (PE1) will be found).

- Find any specialisations of this prototypical event (in the example these will be events representing 'woundings' by, for example, 'stabbing' and 'thumping' as well as woundings by 'shooting' shown on the diagram, E2).

- Find the instrument slots in these events. These will most likely have to be inherited from the prototypical event for the new specialised action as the specialised event will not necessary be a prototypical event itself. In this example the instrument 'firearms' will be inherited from the prototypical event for shooting (PE2). E2 is not a prototypical event itself as shootings are not necessarily specialisations of woundings (e.g., 'shooting at a target').

- If the instrument in the original event is a specialisation of (i.e., somewhere below in the hierarchy) one of these prototypical instruments, then the action of the original sentence can be replaced by the relevant, more specialised, action. In this example, the node representing 'the gun with which Jack wounded Jill' is below the node representing 'firearms' so the action 'to wound' can be substituted by the action 'to shoot'.

Sometimes it is possible to drop the instrument clause from the original event once this type of transform has been performed. This can be done when the instrument is the same as the prototypical instrument or the *most common specialisation* of this instrument. This most common specialisation is what would most usually be inferred to be the instrument of the event if it were not present. For example, 'a gun' could be inferred to be the instrument of a shooting event in the presence of no other information, similarly 'a pen' could be the most common specialisation of writing instrument. However, the most common specialisation link in SemNet must be dynamic as it is context dependent. In some circumstances the most common specialisation could differ dramatically from the norm. This area of context and semantics has not yet been fully developed and so, for the time being, the static most common specialisation link will be used unless a special context flag is

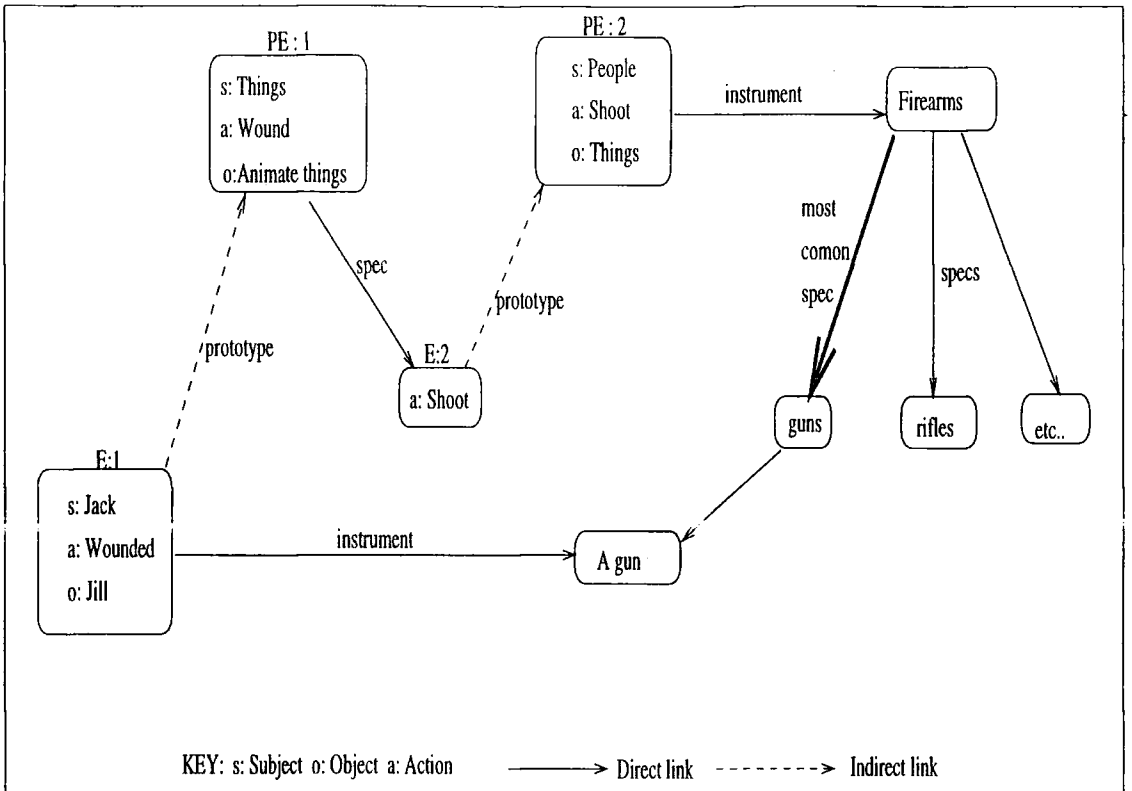


Figure 6.10: Simplified SemNet portion showing how instrument/action transforms can be made

present. If the original instrument node has any additional information to the most common instrument specialisation then it cannot be dropped from the transformed event without losing information (e.g., 'hand gun', 'Bostick glue' in figure 6.9).

There is a problem when the action derived from the instrument of the event conveys some directionality. For example the transform between the sentences 'I talked to you with a phone' and 'I phoned you' is not a valid abstract transformation as the second sentence contains more information; the directionality of the event.

6.9 Multiple Transformations

Many of the transformations described in the above sections can be carried out in combination with each other. Figure 6.11 shows how four transformations can be carried out on one original event. The types of transformation carried out are as follows :-

- Sentence 1 is transformed in to sentence 2 by generalisation of the action ' to murder' into 'to kill' and the addition of the information which determines the difference between these two actions; that murder is 'unlawful killing'. In fact, if the special context flag is not set, the information about unlawful killing can be dropped as 'murder' is normally the most common specialisation of 'killing'.

- Sentence 3 is created using another generalisation transform. This time 'to kill' is transformed to 'wound causing death'. This sentence is rather unnatural but as well as being a transitional sentence for transforms producing more natural sentences, there are instances when this unnatural style maybe useful.

- A specialisation of an action using instrument information results in the fourth sentence. This is the same example as described in section 6.8.1. It is assumed that there is no special context flag and so the most common instrument of a shooting, 'a gun' can be dropped from the sentence.

- The final sentence does not in fact result from an abstract transform but from a grammatical transform operating in the realisation stage. If an action of a subject on an object causes the object to change state and there is a concept describing that state which can be realised as an adjective, then a substitution is possible. In this case 'Jack causes Jill' to change state from 'being alive to being dead' and there is an adjective 'dead' which describes this state.

6.10 Conclusion

Abstract transformations are a novel way of allowing the LOLITA generator to produce variation and paraphrasing. By performing transformations on the SemNet input before realisation (i.e. before the input is passed to the plan-realiser) or during realisation (e.g., before the realisation of each clause) the power of the generator is increased while avoiding over-complication at the realisation stage.

1. Jack murdered Jill with a gun
2. Jack (unlawfully) killed Jill with a gun
3. Jack (unlawfully) wounded Jill with a gun causing Jill to die
4. Jack (unlawfully) shot Jill causing Jill to die
5. Jack (unlawfully) shot Jill dead

Figure 6.11: Example of a multiple transformation

Other systems have capabilities for similar paraphrasing (see section 3.10): however, the need for explicit lexical entries or annotations to indicate that various alternatives are possible would cause problems if these systems were to be scaled up. The rule-based approach adopted here avoids such problems and allows novel cases to be handled.

One possible criticism is that the addition of such variation capabilities to the generator without careful concern for how they will be controlled (by the planner) is dangerous. There is the opposing view, however, that (even without detailed consideration of control) adding the ability of variation is a good step forward. Indeed, until generators have been given such capabilities, the problem of control will not arise and may be ignored (see section 3.10.1).

This chapter has introduced examples of different abstract transformations: there are many more possibilities which could be the subject of further research. This stage of processing is also relevant to the process of normalisation during NL interpretation.

Chapter 7

Implementation

The first part of this chapter (section 7.1) will provide an overview of the implementation of the plan-realiser described in chapter 5. Some of the most important datatypes used in the implementation are described before a simplified portion of code is presented and explained.

One of the most significant decisions which must be made in the design of a system is which programming language to use in its implementation. An appropriate language can minimise the difficulties in code design; make the program more readable and therefore easier to maintain, and reduce the amount of testing required [Sommerville, 1992]. The choice of the functional programming language Haskell [Hudak *et al.*, 1994] was a starting point assumption for this work (see section 1.4). However an explicit aim of the project is to examine the suitability of Haskell for NLG (section 2.2.6). The second part of the chapter (section 7.2) will fulfil this aim by examining the most important features of Haskell and discussing how it has affected the system implementation and development.

7.1 Implementation Overview

7.1.1 Some Important Types

This section will briefly introduce the important types (see also section 7.2.4) used in the implementation of the LOLITA natural language generator.

- **Global**: The **Global** datatype is perhaps the most important of those used in the LOLITA system. Functional programming languages do not allow *side effects* (see section 7.2.1) so the 'state' of the system has to be made explicit and passed round between functions rather than leaving it implicit (as in imperative languages). The **Global** datatype corresponds to this overall system state. It holds, among other things, the whole of LOLITA's SemNet representation (both its conceptual and linguistic levels) together with information on how the SemNet has been most recently changed. The **Global** also holds all the SemNet information from which to generate and the various planning instructions set by the planner or the underlying application (see chapter 5).

- **Noderef**: A **Noderef** is simply a reference to a particular unique node within the SemNet representation. A **Noderef** together with the complete SemNet held in the **Global**, defines the meaning of that node (see section 1.5.1).

- **Meaning**: The **Meaning** datatype is simply a 'repackaging' of information held in the Global and Noderef datatypes to make it more suitable for generation. A **Meaning** holds the meaning of a particular node in the SemNet by combining a starting point node and the complete SemNet representation.

- **Generator**: The **Generator** is a datatype which acts as a 'building block' during the generation process. As an utterance is built, generators representing different parts of the utterance are composed together to form a more complete generator. This generator is then applied to the input instructions from the planner to produce a NL utterance. The generator comprises the utterance generated so far as well as planning instructions and switches set by the planner.

```
  * event: 95979 *
universal_:
  event - 7688 - rank: universal - definition_
subject_:
  charity - 95973 - rank: individual
action_:
  receive - 78714 -
object_:
  money - 95975 - rank: individual
origin_:
  john - 95977 - rank: named individual
time_:
  past_ - 20991 -


---> SAY Event 95979, Active, Short Rhythm ... etc
```

Figure 7.1: Simple input event and instruction passed to the plan-realiser

- **GenVals**: The **GenVals** datatype is a collection of flags set by the generator which can affect the future choices that the generator must make. Examples are a flag to force embedded events to be open or closed (see section 5.7.2) and a flag to force nouns to be singular or plural.

7.1.2 General Operation

This section will introduce some basic details about how the generation process is implemented using the datatypes described above.

Figure 7.1 shows a simple (and simplified) representation of an event node in LOLITA's SemNet together with simple planning instructions that could be provided to the plan realiser. A possible utterance describing this event would be *'A charity received money from John'*.

Figure 7.2 shows a much simplified portion of the generator code. It is not necessary for the reader to understand every aspect of this sample code, nor for this section to describe each aspect in great detail. The code and the following commentary are intended to give the reader a taste of how the generation process

is implemented. Some of the aspects of the sample code which are specific to functional languages (and more specifically Haskell) will be discussed further in section 7.2. One important aspect which needs to be explained at this point is that function application, mathematically expressed in the notation *f(x,y)*, is expressed in Haskell as *f x y* (see section 7.2.3).

The type declaration (1)[1] defines say_meaning to be a function which takes a parameter of type **GenVals** and a parameter of type **Meaning** and returns a result of type **Generator**. The code fragment first checks (2) using the **if_**[2] function and the function **is_event_m** to see if the node for which a description is to be generated is an event. Because (in this example) node 95979 is an event, the embedded condition (3) will be reached. This condition depends on the function **forced_close_event_gv** which 'queries' the GenVals parameter to see if a switch has been previously set to force the event to be expressed as a noun. Assuming that this is not the case the function will call the **say_event** function (4).

The fragment of the function **say_event** shows the simplified code to decide whether an event should be expressed in the passive or active voice (see section 5.5.2). The query function **if_gen** (5) queries the hidden parameter passed by Generator type (as mentioned above the generator type incorporates the utterance so far as well as the planner's instructions, see also section 7.2.3). In this case the **is_style** function determines whether the planner has requested an active or passive event. In this example the active voice is required so the function **say_active_event** (6) is called.

The simplified function **say_active_event** controls the generation of active events. The function calls other functions to generate phrases for each of the roles in the event (say_subject, say_action, say_object etc). Each of these functions returns a **Generator** and the **before_** (9) function is used to compose or merge

[1]The general type signature of a function is given as, for example:-
$f :: a- > b- > o$
meaning the function 'f' takes a parameter of type 'a' and a parameter of type 'b' and returns a parameter of type 'o'

[2]the occurences of **if_ before_**, **if_gen** *etc.* are not Haskell constructs but functions in their own right (see section 7.2.3)

the generators together. So for example, the **say_action** (10) function will have access to information produced by the **say_subject** (8), and **before_** (9) will add together the utterance strings produced by each.

There are two important simplifications in the example presented above:-

- The example code presented is in a 'grammar directed' rather than a 'message directed' style (see section 3.4 and 5.5.2) as control appears to be 'hard-wired' into the code. The function calls functions such as **say_origin** (11) even if the input event has no such link. In the 'unsimplified code' control is passed to the SemNet input and functions to realise particular roles are only called when they are present in the input event. The unsimplified code checks that arcs are present in the input before calling functions that produce utterances for them.

- Throughout the example the value of the **GenVals** parameter *gv* and the **meaning** parameter *e* have not been changed. In a more complex example, the GenVals parameter may be changed by adding or changing flags in order to constrain or direct future generation. The generation process will also need to generate utterances for different Meanings. The **say_subject** function, for example, will contain a recursive call to the **say_meaning** function with the meaning parameter representing the subject of the event (e.g., the meaning for node 95973 in the example in figure 7.1).

7.2 Features of Haskell

The following subsections will discuss the properties of functional programming (and more specifically Haskell) which are different to more common programming languages (e.g., imperative languages). The effects of each of these properties on the implementation of the LOLITA NL generator will be discussed.

```
say_meaning:: GenVals -> Meaning -> Generator          (1)
say_meaning gv n
  = if_ (is_event_m n)                                 (2)
        (if_ (forced_closed_event_gv gv)               (3)
             say_event_as_noun gv n
         `or_else`
             say_event gv n                            (4)
        )
    `or_else`
        say_entity gv n
        .
        .
        .
        etc

say_event:: GenVals -> Meaning -> Generator
say_event gv e
  = if_gen (is_style Active)                           (5)
       say_active_event gv e                           (6)
    `or_else`
    if_gen (is_style Passive)                          (7)
       say_active_event gv e
        .
        .
        .
        etc

say_active_event:: GenVals -> Meaning -> Generator
say_active_event gv e
  =   say_subject gv e                                 (8)
    `before_`                                          (9)
      say_action gv e a                                (10)
    `before_`
      say_object gv e
    `before_`
      say_origin gv e                                  (11)
         where
            a = action_m e
```

Figure 7.2: Simplified portion of the NLG Haskell code

7.2.1 Referential Transparency

Pure functional programming languages such as Haskell have the mathematical property of *referential transparency* which prohibit *side-effects* such as assignments which are rife in imperative languages (e.g., Pascal, C etc). The value of a function or expression in Haskell is dependent solely on the values of its sub-expressions and not dependent on 'hidden' values. Unlike imperative languages where a variable may be assigned several different values within an expression, different occurences of the same variable name in a Haskell function always have the same value. The properties of referential transparency can contribute to the ease of understanding of a program written in a functional language such as Haskell [Hazan *et al.*, 1993]. Ease of understanding is of course, closely related to ease of development.

Function declarations in Haskell clearly define the 'interface' of each function. The declaration states what the function takes as arguments and what it returns to the function that called it. No other hidden side effects can occur.

For example the top level NLG function has the following type [3]

```
nlg::  Global -> Noderef -> [Char]
```

The function nlg takes as parameters the Global (see above), a reference to a particular node in the SemNet input and returns a list of characters which form a NL utterance describing that node. No other parameters or values will be changed. In particular, in this example, the state (i.e the Global) of the system will not change after generation.

7.2.2 Higher-order Functions

A higher-order function is a function which takes another function as an argument or delivers one as a result[Bird and Wadler, 1988]. The use of such higher-order functions is an important feature of functional languages such as Haskell. They

[3]This example, as other examples in this chapter, maybe somewhat simplified.

allow concise forms of expression [Turner, 1987] and together with lazy evaluation allow for new levels of modularity to be attained: this enables programs to be more easily read and understood [Hughes, 1989]. Functional programming languages treat functions as 'first-class' citizens allowing them to be used in abstract datatype representations.

The example code fragments presented in section 7.1.2 show the use of higher order functions. The abstract datatype Generator is represented by a function which takes a datatype which comprises planning instructions and details of the generated utterance so far, and returns a datatype which includes a new longer utterance.

7.2.3 Currying

Haskell and other functional languages support *currying* where structured arguments can be replaced by a sequence of simpler ones [Bird and Wadler, 1988]. A function **f** applied to two arguments **x** and **y** is represented in Haskell as **f x y**, meaning that the result of applying **f** to **x** is a function which is then applied to **y**. For example the function **add** could be defined by :-

```
add x y =  x + y
```

and the expression **add 2 3** is interpreted as **(add 2) 3** where **(add 2)** is a function which takes a single argument and adds the value 2 to it. Currying makes it possible for functions to be greatly simplified merely by leaving out unnecessary arguments, thus aiding readability and abstraction [Hazan *et al.*, 1993].

The example code fragments presented earlier give an example of currying although this may not be immediately obvious because of its use in combination with the use of abstract datatypes and higher order functions. The functions (for example **say_meaning**) return a result of type **Generator** which is in itself a function. The use of this datatype could be replaced by an explicit reference to the functions

type, for example (1) could be replaced by[4]:-

```
say_meaning:: GenVals -> Meaning -> GenTypeA -> GenTypeB
say_meaning gv n
.
etc
```

The signature of the function **say_meaning** now indicates that the function should expect 3 input parameters but using currying, only two (*gv* and *n*) are made explicit in the function definition.

7.2.4 Abstract Types

An abstract datatype is a portion of code which appears to the programmer as independent of any particular representation. Values of an abstract datatype can only be processed using functions specifically provided to access the type. This means that the 'concrete' representation of an abstract type can be altered without any effect on other portions of code that use it: the type is completely determined by the provided 'access' functions and their behaviour [Holyer, 1991]. To implement an abstract type, a programmer needs to provide a representation of its values and define operations on the type in terms of this representation. Apart from these obligations a programmer is free to choose between different representation on the grounds of efficiency and simplicity [Bird and Wadler, 1988].

Section 7.1.1 introduced some of the abstract datatypes used in the LOLITA generator and section 7.1.2 gave examples of how they are used. The examples have shown how abstract datatypes can aid in abstraction: it is possible to show how the generator process is carried out in abstract terms without having to describe representation details. The effective use of abstract datatypes and the careful definition of functions used to manipulate them can be used to define a high level 'language' useful for specific tasks. In the code fragments provided in section 7.1.2,

[4]That is we replace 'Generator' by 'GenTypeA − > GenTypeB'

for example, most of the constructs and functions used (e.g., **if_**, **if_gen**, **before_or_else** *etc.*) have been defined as part of an abstract datatype and are not standard Haskell.

7.2.5 Lazy Evaluation

Lazy evaluation allows unevaluated expressions to be passed to a function leaving the function to be responsible for evaluating them as and when their values are needed [Holyer, 1991]. Lazy evaluation allows programs to manipulate extremely complicated and large values (potentially infinite values) whose complete evaluation would otherwise be time-consuming or even impossible. Lazy evaluation can also aid abstraction and program clarity by allowing the programmer to separate different aspects of a solution that would otherwise (i.e., in an imperative language) need to be combined. For example, in a search problem Haskell would allow a set of functions to be built which generated possible solutions (perhaps an infinite number) and a separate set of functions to decide which of these solutions should be chosen. In an imperative language all the possible solutions would have to be generated before being passed to a selection function: if the chosen solution is in fact the first one to be generated then the cost of building all the other solutions would not be necessary. The ability to separate such components of an algorithm can also improve the modularity of algorithms.

Lazy evaluation is used extensively in the implementation of the LOLITA system and its generator. The **if_** and **if_gen** functions rely on lazy evaluation (again, these are not Haskell constructs but functions in their own right). When using the **if_gen** function, for example, only one of the two branches is required to be evaluated. In figure 7.2 the **if_gen** function in line (5) takes two arguments, one being the function **say_active_event** (6), the other being the result of the embedded **if_gen** statement in line (7). It is desirable that only one of these branches is evaluated (branch (6) in our example).

7.2.6 The Haskell Type System

In imperative languages, type-checking ensures that types are consistent within each program statement, but they rely on the sequence of statements being correct; the type-checker cannot detect errors at this level. In functional languages, however, the type-checker checks the program at the level of function application. This means not only that a greater proportion of errors are 'caught' by the Haskell type-checker but also that the type specification gives more information about what the function does than in an imperative language [Hazan *et al.*, 1993].

This type-checking feature greatly aids development as most logical and typographical errors are caught during compilation: once a function successfully compiles most of the work has been done.

7.2.7 Data Structures and Management

Low level operations such as allocating sufficient memory for datatypes (e.g., *malloc* in the 'C' language) do not have to be performed in Haskell. What is more, dynamic datatypes (such as stacks, lists or trees) can quickly and easily be implemented without the use of, for example, pointers.

These features relieve a burden from the programmer and both aid program comprehension and shorten program development time.

7.2.8 Prototyping

The properties of Haskell (and other functional languages) presented above mean that they are highly suited for rapid prototyping:-

- referential transparency allows the clear definition of the interface between functions.
- the use of lazy evaluation allows complex data structures to be passed between functions without the worry of memory allocation or efficiency. Lazy

evaluation can also improve the modularity of programs.

- the use of abstract datatypes and manipulation functions allow high level abstract problem specific programming languages to be defined.
- it has been estimated [Turner, 1982] that each line of code in a functional program is equivalent to about 10 lines written in an imperative language such as 'C'. Prototypes can therfore be written more quickly in a functional language than an imperative one and the whole software development process is shorter as less time needs to be spent in the debugging and maintenance of the program [Holyer, 1991].

7.2.9 Suitability for Parallel Execution

Referential transparency and the lack of side effects in functional programs makes them suitable for parallel execution as the problem of propagating side effects between processors is avoided. The problem of correctness in parallel functional programs is the same for normal non-parallel functional programs: there may be no difference in code at all between parallel and non-parallel functional programs. This is in great contrast to the difficulty of solving the correctness problem for imperative parallel programs [Peyton Jones, 1989].

The LOLITA system code (including that for the generator) can be compiled using the Glasgow Haskell Compiler [The AQUA Team, 94]. This compiler has been designed to produce executable code which can be run on parallel machines and work is in progress both on LOLITA and GHC (e.g [Garigliano *et al.*, 1995] [Peyton Jones, 1989]) to produce a parallel version of LOLITA. Many of the problems associated with NLE (for example, concurrent searches, the handling of ambiguity) lend themselves very well to parallel solutions. Running LOLITA on a parallel platform will be greatly beneficial with respect to execution time.

7.2.10 Disadvantages of Haskell

The section will discuss the disadvantages and problems with the Haskell programming language :-

- Despite the use of Haskell being relatively wide spread in academia, there is a lack of general acceptance and a good deal of scepticism as to whether Haskell can be used in the commercial environment. Recently, however, this situation seems to have improved (e.g., this is reflected in a recent conference dedicated to the use of functional programming in the 'real' world [Giegerich and Hughes, 1994]).

- This lack of acceptance means that there is little support available in terms of support tools and standard libraries. However, other research at Durham is concerned with providing debugging tools[Hazan and Morgan, 1992] and profiling techniques [Morgan and Jarvis, 1995] for Haskell. Libraries for Haskell and other functional languages are also becoming available (e.g., the LML fudget library [Carlsson and Hallgren, 1993]) although there are currently no standards.

- Although the lack of side effects in functional languages (see section 7.2.1) leads to a variety of advantages, it also has disadvantages. Some algorithms are most easily expressed in terms of side effects. However, monads [Wadler, 1992] can be used to express such algorithms in such a side effect style.

- Some operations such as input and output (which are of course important in NLE) are more difficult to handle using Haskell and other functional languages compared to imperative languages. However, using abstraction techniques these problems can be 'hidden away' from the programmers of other modules.

- Another important drawback with the use of Haskell for large-scale systems such as LOLITA is its inefficiency. The programmer does not have to worry about low level aspects such as storage allocation but this leads to inefficiency

at runtime. Memory requirements are often high and the frequent process of *garbage collection* leads to slower execution times. However, although Haskell is not yet as efficient as the more common imperative languages, its efficiency has increased dramatically recently.

7.3 Other Implementation Details

The LOLITA system as a whole comprises about 37,000 lines of Haskell (estimated to be equivalent to about 400,000 lines of imperative code) and 2000 lines of C in about 200 modules. LOLITA is probably the largest application (i.e., non-complier) in the world to be written in Haskell. The generation component consists of about 7500 lines of Haskell (equivalent to 75,000 lines of imperative code) in 20 modules (although the generator also makes heavy use of shared code).

The LOLITA system and thus its generator currently runs on a Unix Sun Sparc workstation with 80Mb of memory.

The LOLITA generator runs in real time unless the system performs a garbage collection during generation (in this case output takes a couple of seconds).

7.4 Conclusions

This chapter has given a brief discussion of the operation of the LOLITA generator and, following an explicit aim of the project, discussed the effects of the chosen implementation language Haskell on the generator's development.

Section 7.2 highlighted the properties of Haskell (and functional languages in general) and discussed their impact on the implementation and development. In particular, the features of such languages make them particularly suitable for developing large-scale prototypes. Whether Haskell will be suitable for real commercial applications remains to be seen but if its development (particularly development which will tackle the problems listed in section 7.2.10) continues at the same rate

as recently, then the future is promising.

Chapter 8

Evaluation and Results

This section aims to evaluate the LOLITA NLG system with special attention to the adopted methodological approach (chapter 1) and the problem specific aims (chapter 2). Unlike other sciences (including other branches of computer science), the evaluation of NLP systems and especially NLG systems is not well documented or developed. An explicit aim of the project (aim 7, section 2.2.7) is that the discussion and suggestions concerning evaluation of NLG presented in this chapter should be useful in themselves.

The first part of the chapter (section 8.1) presents a brief survey of NLP evaluation methods including comments about a particular method suggested by Galliers and Sparck Jones [1993] (section 8.1.2) and information about evaluation specific to NLG. The chapter then details a particular evaluation of the LOLITA NL generator following the Galliers and Sparck Jones method (section 8.2) before giving conclusions about the current state of the art in NLG evaluation (section 8.3). Finally, the chapter turns to the evaluation of each of the project aims (section 8.4) against their criteria for success detailed in chapter 2.

8.1 Evaluation of Natural Language Systems: A survey

8.1.1 Competitions

One method of evaluation is by competition. In the field of NL, there have been many competitions in areas such as machine translation, message understanding, speech recognition, database interfaces and parsing.

The MUC series of evaluations [DAR, 1993] have involved the evaluation of information extraction systems applied to common tasks in order to measure and foster progress in information extraction. For the next MUC competition (MUC-6, to be held in the autumn of 1995) the objectives have been increased in order to push information extraction systems towards greater portability to new domains, and to encourage more basic work on natural language analysis by providing evaluations of some basic language analysis technologies. The areas of evaluation are now co-reference identification, named entity and 'mini-MUC' template filling [Grishman, 1994]. The LOLITA system is entered for all three areas of the MUC-6 competition.

There have been no such competitions specifically for NL generation evaluation.

8.1.2 Galliers and Sparck Jones

This report on NLP evaluation [Galliers and Sparck Jones, 1993] is presented in three parts. The first discusses the concepts which are important to NLP evaluation and uses the authors' experience in the field of Information Retrieval evaluation to define an extensive terminology and framework for NLP system evaluation. Secondly, a report on the state of the art for NLP evaluation is presented. Details of previous and current evaluation methods, competitions and workshops are discussed, particularly in the areas of machine translation, message understanding, speech recognition, and database query. The difficult problem of evaluating sys-

tems which have not been designed for a specific setup (which are termed *generic* systems, see section 1.5.5), is also discussed.

Finally, part three of the report presents a general approach to NLP evaluation which is "aimed at methodologically-sound strategies for test and evaluation motivated by comprehensive performance factor identification" (page 140). This method is illustrated mainly through the use of examples. It is important to note that these examples, although precisely defined, are not evaluations on real existing systems but on hypothetical systems in hypothetical situations. The recommendations associated with the examples, together with the terminology and framework defined in the first part of the report, form the most developed general NL evaluation methodology at present and will be investigated further in the following subsections and used to build an evaluation for the LOLITA NL generator in section 8.2.

The Evaluation Framework

The first, perhaps obvious, conclusion made by Galliers and Sparck Jones [1993] is that due to the variety of systems and tasks in the area of NLP, there can be no 'magic', all-encompassing evaluation method. They say:-

> "We cannot offer instructions along the line 'Take 14 texts consisting of 14 messages from the UP wires ..'. Just hypothesising concrete instructions of this sort shows what a mistaken idea this would be, even if some of the evaluation literature suggests that it might be both desirable and feasible." (page 140)

Instead, evaluations have to be designed for each individual case. It is paramount to define carefully the environment of the system, subsystem or component under evaluation: the entity under evaluation operates within a larger envelope and cannot be evaluated in isolation. An evaluation has to apply to both the system and the setting which together comprise the *setup*. In order to systematically identify the important setup factors, Galliers and Sparck Jones provide a framework of

relevant questions which can be used to decompose the evaluation subject. These questions and their answers are used to build an *evaluation remit* and an *evaluation design*. The pro-formas for a remit and design are shown in figure 8.1, these will be discussed further as a particular example is built (section 8.2).

Extending the Evaluation Framework

Although Galliers and Sparck Jones give detailed descriptions of how to design evaluation experiments, they pay little attention to the procedure after the evaluation has taken place. This procedure will obviously comprise the presentation of the evaluation results but it would also be useful if the evaluation review should also contain a criticism, with the benefit of hind-sight, of the evaluation methods. Thus the **Evaluation Review** incorporates the evaluation **Results**, a review of the evaluation **Methods** and the evaluation **Conclusions**. Comments on the mistakes made in the evaluation, explanations as to why the results were perhaps unexpected and ideas for improvements could be useful for other researchers both to understand the evaluation fully and to be able to design better evaluations.

8.1.3 NLG Evaluation

The previous sections have been concerned with evaluation of NLP systems in general rather than the more specific problem of evaluating generation subcomponents (although the general framework defined by Galliers and Sparck Jones, can be applied to the NLG module as will be done in section 8.2).

In fact, there has been very little work on evaluation of NLG. Perhaps this is because it is still a relatively young field and researchers have concentrated resources on development rather than evaluation. Alternatively, researchers may have thought that since generation leads to actual readable output, evaluation can be done informally. It is interesting to note that although it might be expected that papers on generation would be littered with examples of output, this is not the case !

EVALUATION REMIT

- **Motivation:** Why evaluate ?
  - Perspective: task/financial/administrative/scientific ...
  - Interest: developer/funder ...
  - Consumer: manager/user/researcher ..
- **Goal:** What to discover ?
- **Orientation** intrinsic/extrinsic
- **Kind:** investigation/experiment
- **Type:** black box/ glass box
- **Form (of yardstick):** ideal/attainable/exemplar/given/judged
- **Style:** suggestive/indicative/exhaustive
- **Mode:** quantitative/qualitative/hybrid

EVALUATION DESIGN

- **To identify:**
  - **Subject's ends:** What is subject for ?
  - **Subject's context:** What is in it ?
  - **Subject's constitution:** What is it of ?
- **To determine:**
  - **Performance factors**
    - environment variables
    - 'system parameters'
  - **Performance criteria**
    - performance measures
    - application methods
  - **Evaluation data**
  - **Evaluation procedure**

Figure 8.1: Framework for building an evaluation remit and design

Galliers and Sparck Jones

In their extensive report on NLP evaluation, Galliers and Sparck Jones [1993] include only a small subsection on the evaluation of NLG. This is a reflection of the small amount of work achieved in this area. They say :-

> " Evaluation for NLG remains at the discussion stage. Evaluating generation is difficult; it is hard to define what the input to a generator should be and it is hard to objectively judge the output." (page 98).

They suggest one solution is to evaluate NLG in the context of a specific application by evaluating task performance. Of course, this requires that a specific task exists: this is not the case when evaluating the NLG component of a general purpose base as in the LOLITA system (see section 1.5.5). They also report that Moore suggests a task-orientated evaluation of NLG by assessing the impact of the generated utterances on a user's behaviour. Again this assumes an actual sophisticated substantiating application.

The Seventh International Generation Workshop

NLG evaluation was the subject of a panel discussion at the recent workshop in Maine [NLG94, 1994]. Experts from other NL fields (such as speech recognition and machine translation) were present to give their experience on how evaluation in their fields had evolved. The consensus was that, because of the state of current generation systems, the adoption of certain specific metrics for evaluation would not be possible and would perhaps even be detrimental. However it was concluded that generation evaluation is extremely important and that it should be up to each research group to include information about how their systems are evaluated.

8.2 Example Evaluation

8.2.1 Introduction

The following sections present an evaluation of the LOLITA generator using the method described by Galliers and Sparck Jones [1993] together with the suggested extensions (section 8.1.2).

The aim of this evaluation exercise is two-fold:-

- To help evaluate the project's aim to build a generator which can produce NL descriptions of nodes in LOLITA's SemNet (aim 2, see section 2.2.2).
- To evaluate the evaluation method presented by Galliers and Sparck Jones [1993]. The example evaluations presented in Galliers and Spark Jones concern hypothetical systems in hypothetical environments. The evaluation presented here aims to apply the theoretical evaluation methods to a real system and environment.

In this evaluation method the 'setup' is paramount: the system's task and its relationship with the environment must be precisely defined. In the examples given by Galliers and Sparck Jones [1993], evaluations are given for complete application systems (although hypothetical). The following example evaluation, however, will not concern such a final application for two reasons:-

- The LOLITA generator is not part of such an application and so no natural setup exists in the LOLITA case: instead a more 'artifical' setup has to be chosen and it is this setup that has to be defined. It is important that this 'setup' is really evaluating the generator module rather than other LOLITA modules (e.g., the syntactic or semantic analysis).
- Evaluation of a final application rather than a more generic subcomponent will be easier and result in more precise results but may actually measure much less. A generation system which provided instructions on how to operate an appliance, for example, could easily be evaluated by seeing if users

could work out how to use that appliance. It could not however measure any of the generators capability in any other task. Since the LOLITA generator is part of a general purpose base, its evaluation needs to be more generic.

The general idea of this evaluation is to compare human generated descriptions with those produced by LOLITA and measure the acceptability of the computer generated utterances. There are many different variations that could be achieved using this basic procedure. The rest of the section will go through the process of building a remit and design for one particular evaluation. As this evaluation is being built, alternatives which would lead to different experiments will also be discussed.

8.2.2 The Evaluation Remit

This section describes the evaluation remit presented in figure 8.2. Each slot is now discussed:-

- **Motivation**: The Perspective of the evaluation is *scientific* both for the specific evaluation task itself and, at a meta-level, for evaluating the evaluation process itself. In this case the interest prompting the evaluation and the consumers of the results are likely to be the same: either developers of the LOLITA system or other researchers in the field. Once again, the interest and use of results could be associated with the evaluation task itself or the evaluation procedure.

- **Goal**: This slot should summarise what the evaluation is intended to achieve. In this case 'to indicate the effectiveness of the LOLITA NL generator within the general semantic analysis operation of LOLITA (in comparison with humans performing the same task)'. If the **Kind** of evaluation is to be an experiment instead of an investigation (see below), a further goal could be to indicate weaknesses in the generator.

- **Orientation**: An evaluation can either be intrinsic and relate to a system's objective or extrinsic and relate to its function (i.e. to its role in relation to its setup's purpose). In this example evaluation the orientation is extrinsic: the performance of how well the generator can do in a particular task.

- **Kind**: The evaluation is primarily an investigation to determine the performance of generator in the semantic analysis setup (see section 4.4.1). With slight modifications the evaluation could also be an experiment: it could indicate areas in which the generator is poor and show where resources should be channelled.

- **Type**: The evaluation will be black box as this is an input/output only evaluation.

- **Form**: There are no recognised benchmarks that can be employed to evaluate the generator's performance as there are no 'correct answers'. The evaluation must therefore be judged: the generator's results will be compared with human performance and judged by humans.

- **Style**: This example evaluation can only be suggestive rather than indicative or exhaustive.

- **Mode**: Hybrid. Results will be both quantitative and qualitative, as both the quantity of acceptability and quality of output will be measured.

8.2.3 The Evaluation Design

Identification

The identification part of evaluation design 'defines the evaluation subject at the level of detail necessary to conduct the evaluation' (page 141, [Galliers and Sparck Jones, 1993]). In this case, the identification is difficult because it is an artifical setup: instead of merely identifying parameters from an environment, it is necessary to first define that environment.

EVALUATION REMIT

- **Motivation:**
  - Perspective: scientific
  - Interest: developer/ other researchers
  - Consumer: developer/ other researchers
- **Goal:** To indicate the effectiveness of the LOLITA NL generator within the general semantic analysis operation of LOLITA (in comparison with humans performing the same task).
- **Orientation** extrinsic
- **Kind:** Mainly investigation (some aspects of experiment)
- **Type:** black box
- **Form (of yardstick):** judged
- **Style:** suggestive
- **Mode:** hybrid

Figure 8.2: Evaluation remit for the LOLITA NLG evaluation experiment

- Subject's Ends: (i.e. the subjects objectives/function): To generate descriptive utterances in the general text analysis operation (section 4.4.1).
- Subject's Context. As part of a complete system which produces 'correct' results. As the realiser is part of a system that is assumed to be capable of other tasks we have to choose examples where these other areas are correct. If this were not the case we would be evaluating other areas of the system and not the generator in isolation.
- Subject's Constitution: SemNet input (comprising both conceptual and lexical information), Grammar rules, realisation parameters.

Performance factors

- Environment Factors: Domain of input (terrorist incident). Length of input (paragraph length). The rest of the LOLITA system (i.e syntactic, semantic analysis). Human judges. Ideally many articles of differing lengths and domains should be used: however as this is just an suggestive experiment this ideal is relaxed.
- System Parameters: Realisation parameters (set randomly, or by hand to get a variety of system utterances)

Evaluation Criteria

The *evaluation criteria* indicate what is measurable in the evaluation (*measures*) and how these measures are made (*methods*) :-

- Measures: a quantitative measure of success of the generator (in this setup) based on judged qualitative results. There are four scales of generator acceptability (*Unacceptable*, *OK*, *Good* and *Best* of the group), the latter three indicate success. There is also a comparative measure as the judges were asked to mark utterances according to whether they thought they were computer or human generated.

- Methods: questionnaires filled in by human 'judges'. The judges were from a wide variety of ages, both sexes and included native and non-native speakers of English.

Evaluation Data

The data used in the evaluation was as follows (see appendix C for more details):- One article (five sentences, seventy-two words) which comprised information on fifteen events and entities. Five descriptive utterances were collected for each of these entities and events (this data was collected from LOLITA generator output and a sample of ten humans doing the same task). There were thus a total of seventy-five utterances involved, seventeen computer generated and fifty-eight human generated.

These utterances were then analysed by a different group of ten people. Comments as to why utterances were marked as unacceptable or as computer generated were also collected.

There is no scientific reason why the sizes of these data sets (i.e., 17 computer generated utterances, 58 human generated utterances etc.) nor the domain or article type were chosen. At the evaluation design stage, when there is no information about prior experiments, there is no way of knowing the required size of the data set. On a particular run of the LOLITA system analysing a article for which the rest of the LOLITA system (e.g., syntactic and semantic analysis), 17 utterances of reasonable length were produced. For the purposes of this investigative evaluation experiment this was deemed to be sufficient.

Evaluation procedure

The evaluation procedure was as follows (again the exact instructions given to participants are presented in appendix C):-

1. Give the first set of participants the paragraph and ask them to generate utterances about each event and entity described.

2. Run the same paragraph through LOLITA and collect the generated sentences.

3. Give a selection of computer and human generated utterances to different set of people and ask them to mark each utterance according to its acceptability and as also to whether or not they were human generated. The judges were also asked for comments as to how their decisions were made.

4. Analyse results.

8.2.4 The Evaluation Review

Evaluation Results

The results of the evaluation were as follows:-

1. Acceptability

Total of 750 utterances marked:-

Totals :	55% good or best	29% Okay	16% Unacceptable
Computer:	33% good or best	50% Okay	17% Unacceptable

Human utterances marked as best of group = 138/580 = 23%

Computer utterances marked as best of group = 12/170 = 7%

These results indicate that human utterances were of better quality than computer generated ones but that the level of acceptability (i.e., proportion marked Okay, good and best) was very similar (84% total, 83% computer).

2. Human or Computer

506 (67%) of utterances were marked as being human or computer
244 (33%) of utterances were not marked i.e., the judges could not tell if the utterance was human or computer generated
103 (41%) of the ones marked as computer generated were correctly assigned
235 (92%) of the ones marked as human generated were correctly assigned
506 (67%) of all utterances marked to be either human or computer generated were correctly assigned

This indicates that the judges were very good at deciding if an utterance was generated by a human but poor at identifying computer utterances.

3. The judge's comments

Comments were also collected as to why certain utterances were marked as being unacceptable. Those comments that applied to computer utterances can be used for future generator improvement. For example, one commonly occurring comment was that of 'over-generation'. Some people found sentences with redundant information unacceptable (e.g *'a forceful person forced the driver to drive to ..'*). This over-generation results from the way the semantics are built (for inference purposes the semantics may need this information) but suggests that some of the SemNet information could be left out of an improved generator.

Evaluation method

This section describes how the evaluation experiment could be improved (using the benefit of hindsight).

Primarily, the experiment was over complicated: judges were asked to do too many different things. The reason for the over complication was because the experiment was not only an evaluation of the generator but a investigation into evaluation techniques. The experiment tried to collect different types of data so as to be useful

for this latter purpose. If the experiment was to be done purely as a evaluation of the LOLITA generator, it should be both simplified and expanded, for example :-

- The different grades of acceptability reduced to just two (i.e. acceptable, unacceptable).
- The experiment to ascertain whether or not each utterance was computer generated should be separated out from the acceptability part (i.e. run as a separate evaluation).
- The number of utterances each judge had to evaluate should be lessened. This is because it was obvious that people began to recognise patterns in grammar and style which helped them ascertain whether they where generated by LOLITA or a human and this might have influenced utterance acceptability.
- In other respects the experiment should be expanded: more articles should be used as well as more participants (i.e. writers and judges).

Evaluation Conclusion

The evaluation has been useful both in indicating the success of the LOLITA generator and in judging the usefulness of the evaluation method adopted when applied to a real working system. Meta-comments about the use of this sort of evaluation technique are left to the following section.

8.3 Evaluation of Natural Language Systems: Conclusions

As can be seen from the previous sections, evaluation of NL systems and particularly generation systems, is still at a discussion stage.

One of the most extensive pieces of work in this area is that by Galliers and Sparck Jones [1993]: at the very least the careful definitions of the terms used in

evaluation that they adopt will be useful. Resulting from the study and application of their 'divide and conquer' method of evaluation to a working system rather than hypothetical situations, an extension has been suggested to include an evaluation review (see section 8.1.2). This is especially useful at the present time when NLG evaluation is at a young stage: there are no benchmarks for NLG evaluation so it is important that people can understand the evaluations others have performed and, if necessary, learn from other people's mistakes.

The investigation into this evaluation method has suggested that it is more easily applicable to final application systems rather than more generic environments. In application systems, the existence of a real set-up, environment and end-users would help to identify the evaluation goals, parameters and variables. In these cases the evaluation method will probably be able to be used to define comparative evaluations which can be applied to more than one system (although the systems would have to be perform very similar tasks). The process is more difficult when this setup is not real because, before the evaluation parameters can be identified, an artificial setup has to be defined. This process can be tedious and liable to misuse as the definition of the setup can be altered so as to ensure that the evaluation of the system is successful. Similarly, the evaluation method is prone to difficulties when a subcomponent is to be evaluated. In this case it is very difficult to define a setup so as only the subsystem in question is evaluated. Mistakes in other parts of the system could be accredited to the subsystem. This is particularly the case for generation as this is the final manifestation of many other sub-processes (including parsing, semantics, pragmatics, inference *etc*).

Although the results of applying the evaluation method to a final application system may be more precise, they may not be as useful as the results from an evaluation of a generic system or component. Specific application system evaluation will only provide results of the performance of the system or component for that limited task. In order to learn something about the generic capabilities or portability of the system/component, it must be modified to run for different applications and the evaluation process repeated.

In conclusion, there is no well defined generation evaluation method which can be utilised. However, the suggestions made by Galliers and Sparck Jones [1993] (together with the extensions presented here) form a promising framework for defining such evaluations in the future. In the meantime, it is important for researchers to bear in mind the problem of evaluation and include information about how their systems have been evaluated. At the very least it would be helpful if NLG researchers include examples of generated utterances in their literature.

8.4 Results Versus Criteria for Success

This section will discuss the results and findings of the project with respect to the criteria for success described in section 2.2.

8.4.1 Aim 1

To follow the AI Goal and principles of NLE.

- The AI goal. The criteria for success for the AI goal was to aim for behaviour that mimicked but not modelled human behaviour. No claim is made that the solution to the NLG problem presented in chapters 5 and 6 is similar to processes in the brain. The production of grammatically well formed English output, however, does mimic human behaviour. The example evaluation presented in section 8.1.3 indicates that for a particular task the human-judged level of acceptability is similar for human and computer generated utterances. What is more, human judges found it very difficult to correctly ascertain if utterances were computer generated or not (this is a small subset of the Turing test).

- Scale. The LOLITA system is a very large-scale system no matter how this scale is measured:- the system has been developed over a period of eight years and the group now exceeds 20 members, SemNet comprises over 100,000 highly connected concepts, the interpretation grammar comprises over 1600 rules, the program comprises somewhere in the order of 40,000 lines of Haskell code (equivalent to many more lines of code in an imperative language).

  As a highly integrated subsystem, the generator automatically inherits many of the LOLITA system's large-scale properties (e.g., the size of the lexicon, the knowledge contained in prototypical events etc.). One aspect of the generator that may not appear large-scale is its grammatical coverage (especially compared to those generators which have been built specifically for large coverage such as those based on systemic linguistic principles. E.g., NIGEL and

GENESYS, see section 3.6.3). Because the LOLITA generator does not comprise separate grammatical rules and is based on a highly procedural control mechanism (i.e. control is highly dependent on the SemNet input), the grammatical coverage is difficult to measure. Of course, grammatical coverage in generation is not as critical as in interpretation: a generative grammar's coverage must be sufficient to produce utterances adequate for its tasks. This adequacy has been confirmed as part of other project aims (e.g., Aim 2, below).

- Robustness The criteria with respect to the acceptability of results has been measured as part of other aims.

  As the LOLITA NLG module develops, it is tested using a variety of methods. As a change is made to the module, *regression testing* is used to ensure that the change does not have detrimental effects on both other aspects of the generation and the LOLITA system as a whole. After a change *white box* knowledge is used to test the generator using relevant examples. Before adding modified code[1] to the revision control mechanism a set of automatic tests (currently about 100) have to be carried out. These *black box* tests cover all aspects of the LOLITA system. The generation specific tests comprise SemNet nodes and the NL utterances which should result from their generation under specified realisation parameters. In addition, extensive testing is carried out each night (when around 15 test articles are semantically analysed by LOLITA, section 4.4.1) and over weekends (around 30 articles). Many other people are regularly using the LOLITA system and, more specifically, the generation module is used to aid other areas of development (section 4.5). Thus the generation module has been the subject of extensive *acceptance testing*. Any errors reported by the 'users' (i.e., developers of other areas) are recorded and corrected. *Defect testing* is then used to show that the error has been corrected and to try and expose similar errors.

[1] from any module of LOLITA, not just the generation module

Robustness under a software engineering point of view is also critical. Even if the performance of the system is not as expected, the system should never unexpectedly terminate or enter an infinite loop (i.e., 'crash'). If the input to the system (i.e., the SemNet) is erroneous or requires behaviour which is beyond the scope of the generator (i.e., it requires an unimplemented grammatical structure), the generator tries to produce something sensible. Useful error messages are also produced so that any problems are brought to the attention of the developer (e.g., an error in the input SemNet might lead to the generator producing the error message *'Warning: no object found for a transitive verb'* and an utterance with a 'generic' object *'The man kissed somebody yesterday'*). Robustness in this sense has been rigourously tested as the LOLITA system is often run continuously for long periods of time. As well as the regular overnight and weekend evaluations described above, other batch jobs are run which require LOLITA to analyse (and generate from the resulting SemNet, see section 4.4.1) many (e.g., around fifty at a time) previously unseen pieces of text (typically one or two paragraph length newswire bulletins): the LOLITA system and the generator it incorporates, very rarely crashes.

- Maintainability. In general, many people have worked on the LOLITA system in order to add functionality and find and remove existing bugs. The LOLITA project uses a strict revision control system which allows simultaneous system development.

  The generation system, however, has been largely developed and maintained by one researcher only (i.e. the author of this thesis). Thus it is difficult to ascertain maintainability with respect to how other people can understand the code. A good indication however, comes from the recent work on generation of Spanish: a person new to the group has been able to understand the English generator and begin to modify it to enable generation in another language [Fernandez, forthcoming 1995].

  Maintainability with respect to adding additional coverage and to cope with changes in the input SemNet structure has been successful. This is indicated

by fact that it has only required one person to perform this task. Furthermore, when producing deliverables, maintenance of the generation modules to cope with enhancements in other parts of the system, has typically been achieved in a short time.

- Flexibility. The LOLITA system as a whole has proved to be flexible. Proportion of time and code spent on domain and task specific development compared to general system development has been extremely low. The code required specifically for the template application, for example, comprises 1% of the total code and required a similar proportion of development time.

  Aim 3, below, will show that the generator has been used by different prototypes. In fact, there has been no generation development which has been specific to any one particular prototype: all the prototype applications which use the generator interface to it using the same function.

- Integration. The generation module is highly integrated both in respect to the LOLITA system as a whole and internally (i.e. the relationship between the generator's subcomponents).

  The generation module has a clear place in LOLITA's design: prototype applications have been able to use the generator using the same general function (see flexibility). The generator has been developed in tandem with the development of the SemNet representation: experience from generation development has influenced the development of the SemNet representation as well as the other way around.

  Internally, the generation has been designed to be integrated. Each component of the generator (i.e. the planner and plan-realiser) has specific roles and does not depend on unlikely assumptions made about each other. The use of realisation parameters has provided a mechanism for integrating these subcomponents both with each other and with the LOLITA system in general.

  The generator has benefited from code re-use. Where code to achieve the required functionality was already present in other modules, it was used (for

example functions to cope with SemNet inheritance were already used in reasoning modules). Furthermore, when new code was required, it was often built in a more general way than was actually needed for the generator so that it could be used by other modules.

- Feasibility. The LOLITA system currently operates on a Sparc workstation with 80Mb of memory. Full semantic and pragmatic analysis of paragraph length pieces of text (e.g., Teletext articles) takes in the order of a few minutes.

  The generation module requires less resources than that of the rest of the LOLITA system (although of course it is impossible to run the generator as a separate system). It produces utterances in real time, certainly faster than can be achieved by humans doing the same task (e.g., in the case of the example evaluation).

  Figures 8.3 and 8.4 show examples of results obtained by profiling the LOLITA system and its generator. In these examples LOLITA was given a paragraph-length piece of text[2] to analyse and profiled for heap (figure 8.3), time and memory usage (figure 8.4). The heap usage diagram shows that the the generator requires negligible heap storage compared to the rest of the system (the generator's heap is represented by one of the thin lines at the bottom of the graph). The time/memory profile shows that for that particular article which involved the generation of about 30 SemNet node descriptions, the generator took under 10 percent of the total time and 15 percent of the memory. The total time spent generating therefore was under 15 seconds (8.5% of 174 seconds) or less than 1/2 a second an utterance.

  Because the feasibility of the generation component of LOLITA is clearly not a problem (at least compared to the rest of the LOLITA system) formal complexity analysis has not been undertaken.

[2]This text was in fact the same shown in the contents scanning example, figure 3.9 and the evaluation experiment, Appendix C

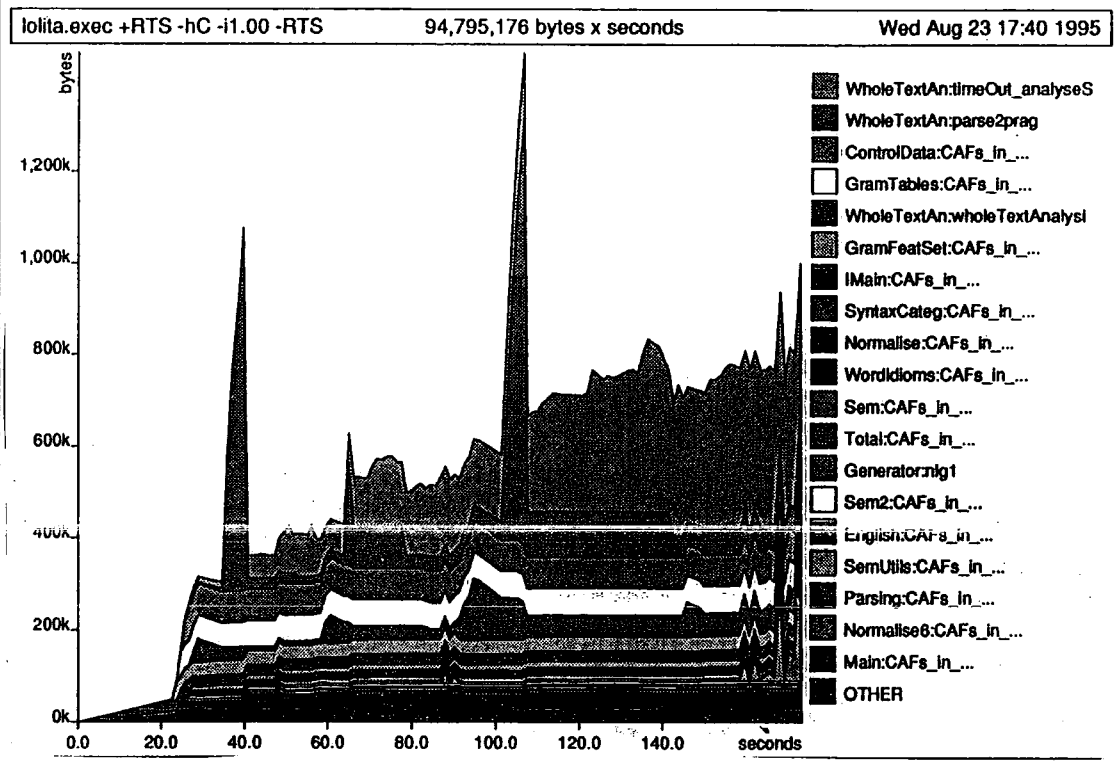


Figure 8.3: Example Heap Profile

```
Wed Aug 23 17:40 1995 Time and Allocation Profiling Report

          lolita.exec +RTS -H40M -K2M -p -h -RTS

total time  =       173.88 secs   (8694 ticks @ 20 ms)
total alloc = 289,944,372 bytes  (23011979 closures)
              (excludes profiling overheads)

COST CENTRE       MODULE         scc subcc  %time %alloc

timeOut_analyseS  WholeTextAn     7     7   41.3   58.3
parse2prag        WholeTextAn     7     7   28.0    8.9
GC                GC            119     0   11.5    0.0
nlg1              Generator      48    48    8.5   13.1
wholeTextAnalysi  WholeTextAn     5     5    6.9    8.4
nlgen             WholeTextAn    44    44    1.5    2.6
nlg               Generator      44    44    0.1    0.2
test_meaning      Generator      48    48    0.0    0.0
tag               WholeTextAn    44    44    0.0    0.0
  ....            .......        ..    ..    ..     ..
```

Figure 8.4: Example Time/Memory Profile

- Usability. The tasks for which applications have been developed and in which the generator is utilised are useful. This is reflected in the amount of research dedicated to these applications and by the fact that work in these areas continues to be funded by commercial organisations.

  Usability is more important when final products are being sold. However, potential customers from industry have been impressed with the LOLITA system as a whole and, in particular, the generation aspects have often been singled out as being impressive. Of course, a system which utilises NLG will be useful (as NL is how humans communicate) but this will only be the case if the NL produced is acceptable. This aspect has been covered elsewhere (see aim 2 for example). The NLG component has been especially useful in system development, without it, progress would undoubtedly been slower (see section 4.5).

- Wide range of techniques. The LOLITA system uses a full range of techniques ranging from generalised theories right through to rule exceptions (chapter 4).

  To meet the criteria for success, however, it is not necessary for each individual subproblem to adopt such a wide range if they are not required. This is the case for the generation module. The generator is largely based on a rule-based approach. If a general rule cannot cover all cases (and cost-benefit considerations mean that the cost of redesigning the rule outweighs its benefits, see below) then exceptions are used. Conversely, if aspects of the generator have functionality that cannot be controlled by an existing rule then this functionality is not suppressed. It is assumed that a controlling rule will exist in future and until then a more ad-hoc approach such as randomness is utilised. The project has also tried to suggest more generalised theories such as the particular roles of the planner and plan-realiser and the application of abstract transformations (see chapters 5 and 6).

- Cost-benefit. As discussed in chapter 1, cost-benefit analysis is a rather grey area: it is essential at a certain level but dangerous if applied too extensively (when the cost of doing the analysis outweighs its benefits).

Before this project was initiated, the LOLITA system did not have any generation capabilities. Generically, therefore, the benefits which have resulted from the development of the generator have greatly outweighed the costs. The need for a generator to be built quickly has influenced the cost-benefit analysis throughout the project; rather than trying to design general theories which cover a small subset of the problem in great detail, the project has concentrated on achieving practical and useful results.

Formal extensive cost-benefit analysis is not always practical or necessary in NLE. However a criterion for success was that informal investigations of alternatives should be undertaken. Section 6.7.3 presented an example of such informal cost-benefit analysis when considering the alternative of adopting a rule-based or explicit data-entry approach to the generation of de-lexical structures.

This section has shown that the NLE principles set out in chapter 1 and their criteria for success have been met in the LOLITA system and more particularly in the LOLITA generator described in this work.

8.4.2 Aim 2

To generate English expressions for concepts represented by LOLITA's SemNet representation.

One criterion for success for this aim was to judge the generator's capability compared with that of humans doing a similar task. This has been the subject of the extensive evaluation experiment in the first half of this chapter. Although not conclusive (due to the fact that the evaluation itself was an experiment), the results indicate that this criteria has been met. Another less stringent, but none-the-less important, indication, is the acceptability of the utterances produced from SemNet nodes resulting from demonstrations. The LOLITA system has been extensively demonstrated to a wide range of both academic and industrial people and the generation capabilities have received few (if any) comments as to its unacceptability.

In fact, the generation aspect has often been singled out as an impressive system feature. Finally, as discussed in sections 2.2 and 4.5, one use of the utterances generated from SemNet concepts is in the ongoing development and debugging of the system. The usefulness of these utterances has become so important that they are now used as the first measure as to whether the rest of the system is operating correctly. An alternative method of debugging would be to utilise a graphical representation of SemNet. Figure 8.7 however illustrates that the complexity of such a graphical input would mean that it would be difficult to use for this task.

8.4.3 Aim 3

To provide NLG capabilities for existing LOLITA prototypes.

All existing prototypes which require NLG capabilities have successfully utilised the generation system described in this work.

One problem encountered in evaluation (see section 8.3) is that it is hard to evaluate subsystems in isolation. This is specially the case in NLG as the utterances produced are the manifestation of the process of the whole application. An error in another subcomponent could manifest itself in NLG and it would be difficult to ascertain if it is the generator or the application which is at fault. As mentioned in section 8.3, evaluation is easier when an actual application and setup exists. However, only prototype applications have thus far been developed and these will not be expected to stand up to rigourous evaluation. Instead this section will give a few examples of how the generator has been used to generate utterances which highlight its flexibility in different applications (section 4.5 explains the requirements of NLG for each application).

Analysis of text

The basic operation of the LOLITA system is to analyse input text in order to build a SemNet representation of its meaning. The generator must then produce a NL utterance for each of the SemNet nodes which has been built. Figures 8.5

and 8.6 show two examples of SemNet nodes and the English utterance generated to describe them.

It is important to stress that the utterances shown are not generated solely from the two nodes shown. The generator takes as input the whole of SemNet together with a starting node from which to generate (see chapter 5). In order to produce the utterances shown the plan-realiser visits somewhere in the order of 30 nodes for each of the two examples. Figure 8.7 shows a graphical representation of some of the SemNet which is used to produce the utterance in figure 8.6. This diagram serves to illustrate the complexity of the SemNet from which the plan-realiser must generate.

Query

Figure 8.8 shows an example query session. All the utterances produced by LOLITA (marked 'L:') are produced by the NL generator. As well as producing responses to the user's input, the generator is also able to produce utterances for the original questions (e.g., *'how many vehicles do I own?'*). The utterances produced by the generator in this example are short and simple and do not show the full capabilities of the generator. This is due to the limitations of the query module rather than the generator.

Translation

Figure 8.9 shows an example of translation. As explained in sections 4.4.3 and 4.5 the current prototype does not produce a single polished translation but rather a series of utterances produced for each of the SemNet nodes that are built from the analysis of the input text. Work is currently underway to build a generator for different target languages (e.g., Spanish [Fernandez, forthcoming 1995]) although this is beyond the scope of this work.

```
    * event: 31941 *
 universal_:
   event - 7688 - rank: universal - definition_
 cause_:
   event - 31938 - rank: universal - suspended_
 subject_:
   roberto - 19845 - rank: named individual
 action_:
   give - 3936 -
 object_:
   tip - 31940 - rank: individual - suspended_
 destination_:
   driver - 31936 - rank: individual - suspended_
 time_:
   past_ - 20991 -
 date:
   26 September 1993
 source_:
   roberto - 19845 - rank: named individual
 status_:
   suspended_ - 29025 -

**********************************
 event:
  You gave a driver a big tip because he was cool and
  the taxi that you called was warm. You were tired so
  you went to your home.


(note: In this example 'Roberto' is the system user and so LOLITA
       uses 'you' to realise this node)
```

Figure 8.5: Example of a SemNet node with its generated NL description

```
    * event: 29180 *
 generalisation_:
   event - 7688 - rank: universal - definition_
 subject_:
   report - 29169 - rank: universal - suspended_
 action_:
   suggest - 3435 -
 time_:
   past_ - 20991 -
 date:
   31 October 1992
 source_:
   telegraph - 9994 - rank: named individual
 status_:
   suspended_ - 29025 -
 object_:
   explosion - 29156 - rank: individual - suspended_

**********************************
 event:
  First reports suggested that at 9pm at nights when
  a forceful person forced a driver to drive a black
  taxi to Whitehall, a bomb went off in it on a corner
  outside Cabinet Office and outside 10 Downing Street.
```

Figure 8.6: Example of a SemNet node with its generated NL description

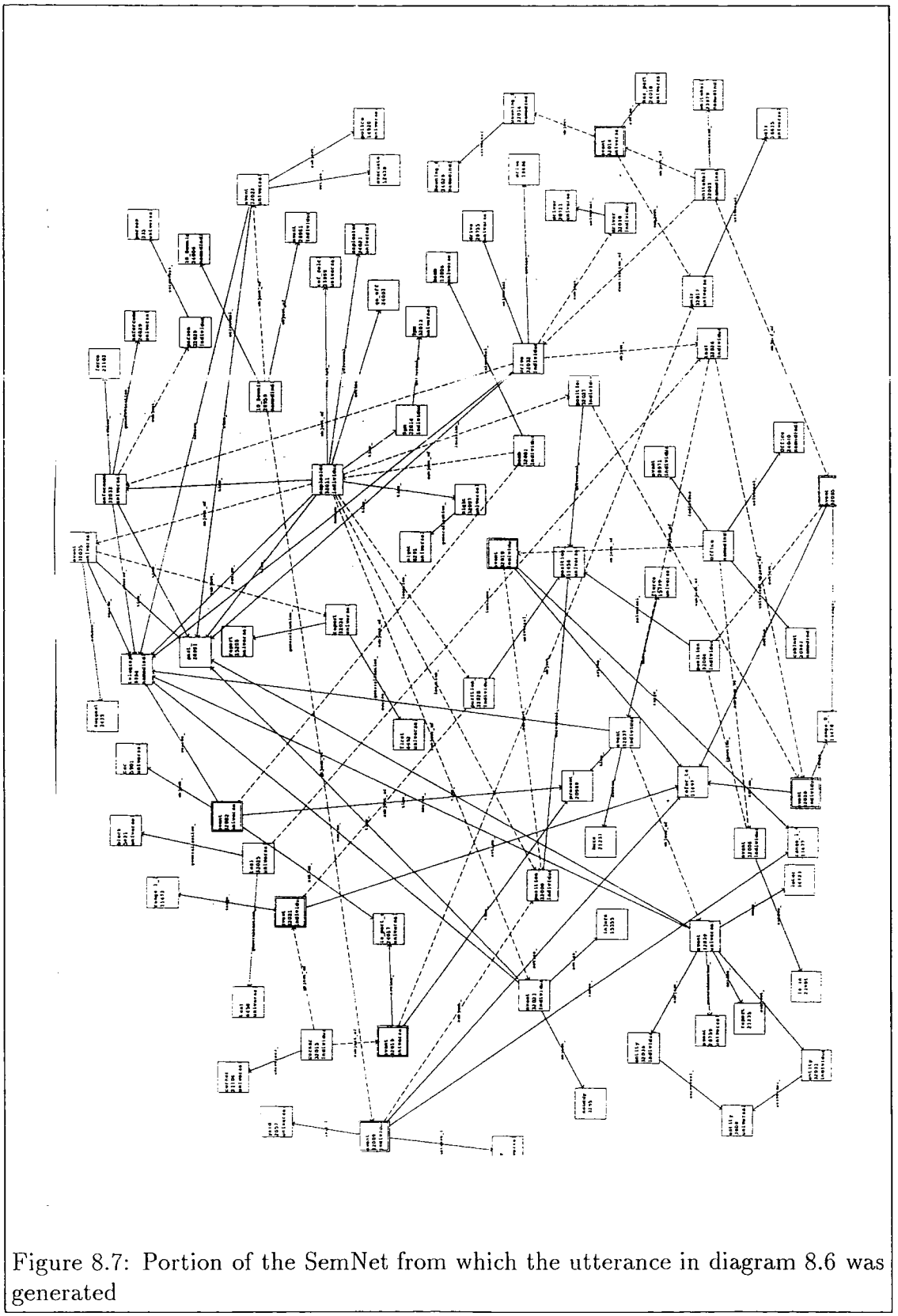


Figure 8.7: Portion of the SemNet from which the utterance in diagram 8.6 was generated

```
U: do I own a motorbike?
L: Yes, you own a big fast motorbike.
U: do I control a vehicle?
L: Yes, you own a big fast motorbike.
U: what do I use?
L: An office.
U: how many vehicles do I own?
L: At least 1.
U: What do I own?
L: A big fast motorbike and a home.
U: why did I go home?
L: You went to your home because you were tired.
U: I was owned by a motorbike
L: I do not believe that a motorbike owned you.
U: was I owned by a motorbike?
L: no
U: do I like you?
L: I do not know if you like me.
U: I like you
L: I understand.
U: and Keith too.
L: I understand.
U: do I like you?
L: Yes, you like me.
U: whom do I like?
L: Keith and me.
U: Keith
L: What about Keith?
U: you admire him
L: I understand.
U: whom do you admire?
L: Keith.
U: I like you.
L: I know.
U: I hate you
L: I do not believe that you hate me.
U: do you know that I like you?
L: Yes, you like me.
U: does Rick know that I like you?
L: I do not know if Rick knows that you like me.

Key:
 U: user's utterances,
 L: LOLITA's utterances (produced by the generator)
```

Figure 8.8: Example query session

```
ITALIAN INPUT:-
se avessi saputo che la grossa moto veloce che mi hai dato
era posseduta da lei, mi sarebbe piaciuta, perche' lei mi
piace davvero.

HUMAN TRANSLATION
(if I had known the big and fast motorbike you gave me was
 owned by her I would have liked it, because I love her.)

LOLITA GENERATOR OUTPUT:-

I gave a woman owner's big fast motorbike to you.

The woman owner whom you love owned the big fast motorbike that
I gave to you and that you would like if you knew that she owned it.

If you knew that the woman owner whom you love owned the big
 fast motorbike that I gave to you then you would like it.

You would like a woman owner's big fast motorbike that I gave
 to you if you knew that she owned  it.

You love the woman owner who owned the big fast motorbike that
 I gave to you. You would like it if you knew that she owned it.
```

Figure 8.9: Example of translation

Content Scanning

Figure 8.10 shows an example of the content scanner or template-filling prototype application. The LOLITA system analyses input articles and fills templates with relevant information. The content scanning module invokes the generator to produce utterances for each of the template slots. The numbers in brackets after each utterance refer to the SemNet node from which they are produced.

8.4.4 Aim 4

The suitability of LOLITA'S SemNet representation for generation.

Not only has the SemNet representation been shown to be suitable for gener-

TELEGRAPH
31/10/92
A car bomb exploded outside the Cabinet Office in Whitehall last night, 100 yards from 10 Downing Street.
Nobody was injured in the explosion which happened just after 9pm on the corner of Downing Street and Whitehall. Police evacuated the area.
First reports suggested that the bomb went off in a black taxi after the driver had been forced to drive to Whitehall. The taxi was later reported to be burning fiercely.

```
Template: Incident
    Incident: The bomb explosion.  (29156)
    Where: On a corner.  (29165)
           Outside Cabinet Office and outside 10
            Downing Street.  (29153)
           In a black taxi.  (29172)
    When: 9pm.  (29159)
          Past .  (20991)
          Nights.  (29152)
          When a forceful person forced a driver to drive
          a black taxi to Whitehall.  (29178)
    Responsible:
    Target: Cabinet Office.  (28969)
    Damage: Human: Nobody.  (3295)
            Thing: A black taxi.  (29171)
    Source: telegraph
    Source_date: 31 October 1992
    Certainty: Facts.  (18664)

Relevant Information

Police evacuated 10 Downing Street.  (29167)
```

Figure 8.10: Example of contents scanner

ation, but some aspects of this representation are critical for the approach taken. The architecture of the generation module and its sub-components has been discussed in chapter 5. The use of an input similar to SemNet is not just convenient but necessary to this proposed solution.

Other examples of aspects of SemNet which have been found to be useful in generation are:-

- SemNet contains rich information required for generation. What is more, this information is close at hand and where it is needed. For example, controls associated with each SemNet node (section 4.3.2) contain important information for generation (both semantic e.g., the rank, and linguistic e.g., the presence of an irregular verb).

- The special representations adopted for positions, time[3] and other internal representations have been useful. This is not least because they have been built in tandem with the generator's needs as well as the needs of other LOLITA subcomponents.

- The rich knowledge in SemNet allows knowledge intensive generation. Prototypical events (section 4.3.2, for example, allow paraphrasing via abstract transformations, see chapter 6).

- The size of SemNet and its lexicon mean that with respect to the lexicon the generator has large coverage: it can chose from over 100,000 root nodes.

- The SemNet assumptions on the relationship between concepts and words have positive effects on generation. The granularity means that there are many language isomorphic concepts which have a direct link to a lexical item that can express that concept. This is in contrast to systems that use primitive concepts or concepts that have a larger grain size than words: in these cases the lexicalisation process is more complicated as for each concept more than one possible word or phrase could be used. For non-language

[3]this development is ongoing.

isomorphic concepts, it is easy to follow round the SemNet representation in order to decompose these concepts into language isomorphic ones.

8.4.5 Aim 5

To adopt a broad coverage approach.

As mentioned in the relevant criteria for success (section 2.2.5), a broad coverage approach is necessary for aims 2 and 3 to be successful. The generator has had to cope with such subproblems as realisation, planning, the generation gap, anaphora and referring expressions, style, paraphrasing, user modelling, context, control methods etc.

8.4.6 Aim 6

To investigate the suitability of Haskell

LOLITA is the largest application program written in Haskell (i.e. discounting compilers) or indeed any pure functional language. Furthermore, the NL generator described in this thesis is, as far as is known, the only generator written in this language. Therefore the project is well qualified to evaluate the usefulness of this language for NLG.

Chapter 7 has discussed the features of functional languages and their effect (both advantageous and disadvantageous) on the development of LOLITA and its NL generator. It was concluded that Haskell is a suitable programming language for NLE tasks, especially for high-speed prototype development. It remains to be seen, however, if Haskell will be a success in the commercial environment.

8.4.7 Aim 7

To investigate NLG evaluation methods

The project has investigated existing methods of NL system evaluation with

particular interest to generation. One of these methods (that suggested by Galliers and Sparck Jones [1993]) has been applied to a real-life working NLG system rather than to hypothetical systems in hypothetical environments. This practical use of the evaluation method has resulted in suggestions as to its usefulness and possible extensions (i.e. the addition of an evaluation review, see section 8.1.2). The project has, therefore, been successful with respect to this aim.

Chapter 9

Conclusions

Chapters 1 and 2 discussed criteria for success for this project: chapter 1 discussed the methodological criteria for this work from the Artificial Intelligence and Natural Language Engineering viewpoints; chapter 2 presented seven Natural Language Generation specific project aims. After aspects of the solution and their implementation were presented, chapter 8 re-examined these criteria to ensure that they had been met.

This final chapter will conclude the thesis by summarising the project's successes (both practical and theoretical) and shortcomings. Possible avenues for further research arising from these shortcomings are also discussed.

9.1 Successes of the Project

9.1.1 Theoretical Impact

This section summarises the theoretical successes of the project :-

- The project has defined and followed methodological principles of Natural Language Engineering: *scale*, *robustness*, *maintainability*, *flexibility*, *integration*, *feasibility*, *usability*, *the use of a wide range of techniques* and *cost-benefit analysis*. Although other researchers may well have considered these

problems, they have rarely made their adopted methods explicit. Many researchers have ignored these problems altogether and their solutions to the problem of NLG generation and other NLP tasks have often been small-scale 'toy' prototypes. The following of such NLE principles is important if research is to lead to useful large-scale NL applications.

- The adopted solution to NLG is based on a novel theoretical architecture. Although, like many other NLG systems, the architecture is based on a two component arrangement, the roles of the planner and plan-realiser are different from traditional planners and realisers. By allowing both components access to the whole of the SemNet input, the effect is to shift some responsibility away from the planner to the plan-realiser. The planner need not formulate a complete plan of the utterance to be produced but merely pass down suggestions (maybe even conflicting) to the plan-realiser. Since the planner need not know any surface linguistic information, the problem of the 'generation gap' is avoided. The adopted solution was heavily influenced by the SemNet representation which is the input to the LOLITA NL generator. Assumptions about certain aspects (such as the relationship between concepts and words and the meaning of each particular concept being defined by the whole of the SemNet) have both influenced and allowed this adopted solution.

- The use of Abstract Transforms. Abstract transformations are a novel way of allowing the LOLITA generator to produce variation and paraphrasing. By performing transformations on the SemNet input before realisation (i.e. before the input is passed to the plan-realiser) the power of the generator is increased while avoiding over-complication at the realisation stage. The rule-based approach adopted to find possible transformations avoid the need for explicit lexical entries and the problems of lexical explosion which they may cause.

- The project has examined the problem of NLG evaluation. Because of the lack of evaluation and discussion about how to evaluate NLG systems it is

very difficult to judge the strengths and weakness of individual NLG systems let alone compare them. Chapter 8 discusses a possible generic evaluation method and suggests some extensions to allow for a review stage in the evaluation process. Although it would be both impossible and unwise to adopt a universal evaluation method or metric at this early stage it is recommended that other researchers consider and discuss how their systems are evaluated. At the very least researchers should include examples of their systems' output in their literature.

- The project has examined some theoretical issues of the adopted functional implementation language Haskell. The use of functional programming languages for NL processing tasks is novel. Although the use of Haskell was a starting point assumption for this work, its effect on the solution has been discussed. Chapter 7 shows how properties of functional programming such as referential transparency, lazy evaluation, higher order functions, currying, the type system and data structures and management are beneficial for the NLG task. Although Haskell (and other functional programming languages) have disadvantages its use is particularly suitable for the development of large-scale prototypes.

9.1.2 Practical Impact

This section summarises the practical successes of the project :-

- The project has resulted in a useful working generator for the LOLITA system. Before the project was initiated the LOLITA system had negligible generation capabilities. Despite the fact that a complete planner has not been implemented, the plan-realiser is already being successfully used for LOLITA application prototypes such as query, content scanning and translation. Perhaps more importantly, the generator has been crucial for the development of the LOLITA general purpose base. The first way of checking the consistency of SemNet and the success of text analysis is by reading the NL descrip-

tion generated for each SemNet node. Without this facility, tracing through the SemNet internal representation for errors would be extremely laborious (figure 8.7 illustrates the complexity of the SemNet output). The practical success of the generator is largely indebted to the initial aim of adopting a broad coverage solution: such practical results may not have been achieved if the project had tackled one specific generation subproblem in isolation.

- The theoretical discussion concerning NLG evaluation led to a practical evaluation experiment being carried out. This experiment aimed not only to evaluate the LOLITA NL generator but to show how such a practical evaluation could be achieved. The review stage of this evaluation allows us to learn from its shortcomings in order to design and execute better evaluations in the future.

- The implementation of the generation module and LOLITA as a whole has resulted in probably the largest functional application program in the world. As well as its successes concerning NLE, this program is a useful practical testbed for functional programming research. The LOLITA code has already been useful for work on error detection and profiling of functional programs. Ongoing work with the developers of the Glasgow Haskell compiler on parallel development of LOLITA will shape the development of Haskell as a parallel language.

9.2 Project Shortcomings and Suggestions for Further Work

This section describes some of the shortcomings of the project and suggests areas of further research:-

- Lack of a complete planner. Work on the design and development of the planning component, although underway, has not been completed. However, the plan-realiser has been designed and implemented with future integration

with the planner in mind, and can already be successfully used without the planning component. As well as further design and development of the planning component more work on the plan-realiser may be required so as to improve and 'fine-tune' the interface between the components. More realisation parameters, for example, may be required.

- Coverage of the generation grammar. Compared to some generation systems (especially those based on systemic principles, section 3.6.3) the grammatical coverage of the plan-realiser is poor. Coverage in generation however is not as critical as in interpretation: the grammar must only be sufficient to generate utterances to convey the meaning represented in the input. As the SemNet representation develops, the grammatical coverage of the plan-realiser may have to be extended to cope, for example, with an increase in the variety of possible SemNet structures. The procedural approach to generation means that the grammar is not made explicit and can therefore be difficult to modify (this is a common criticism of procedural systems, see section 3.4). Further work may try to extend the grammar and its representation. An ultimate goal would be to develop a unified bi-directional grammar for both interpretation and generation in LOLITA.

- Further development of solutions to the sub-problems in generation. One aim of the project was to adopt a broad coverage approach. Although this has resulted in a complete and practically useful generator, some of the sub-problems associated with generation have been solved using over-simplified algorithms. Further work could look into the 'deeper' development of these areas. Better algorithms described by other researchers may be incorporated: this will require work to modify the algorithms so they are not dependent on other formalisms and are compatible with the architecture and SemNet representation. Furthermore, any incorporated algorithms must conform to the principles of NLE.

- Further development of abstract transformations. The heuristics presented in chapter 6, which determine when and how certain types of abstract transfor-

mations can be performed, could be expanded to cover more cases. Other possibilities for abstract transformations could be investigated so as to increase both the efficiency of normalisation during interpretation and the ability to paraphrase during generation.

- The generation of other languages. The project has only concerned the generation of English. Work is already underway to enable Spanish generation[Fernandez, forthcoming 1995]. The Spanish generator has been built based upon the same theoretical principles as the English generator and has utilised a good deal of the same code. The experience obtained from modifying the English generator to a Spanish one can be used to abstract away some of generation principles so as to allow the generation of other target languages.

- Further work on NLG evaluation. Because of the state of the art in NLG evaluation, the evaluation experiment carried out on the plan-realiser is far from conclusive. As generation systems develop, further work on evaluation will be crucial. Experience gained from past evaluations (which can be conveyed using Evaluation Reviews) should allow evaluation methods to improve. Eventually NLG evaluation techniques should be able to compare systems explicitly rather than just evaluate a single system as is the current case.

Appendix A

Examples of Generator output

Sentences generated from an input article concerning hotels and booking arrangements. This text was generated using the 'story' command to simulate the planner (see section 5.1.3):-

> **Sogno is a company that manages its hotels. Sogno needs to modify its information system. New systems may improve reservation services in order to get a better integration level.**
>
> **Bolzano Hotel is near a cathedral and in a town centre. It is a four-star hotel. Bolzano has 5 single rooms and 15 double rooms. Single rooms and double rooms contain a small balcony and a bath. Bolzano has a garden, a private car park and a restaurant.**
>
> **Koenig is a hotel which Sogno owns. It is not far outside a railway station. Koenig is a big new three-star hotel.**
>
> **Sirena is a hotel which Sogno owns. It is by a sea. The rooms that Sirena has have a balcony. 10 rooms look out upon the sea by which Sirena is. Sirena has restaurants, a disco and a garage.**
>
> **Guests make a reservation by calling Sogno's central office or make a reservation by calling the hotels that Sogno owns. A person asks guests about a room type and arrival-departure times. The guests may leave a payment way, a telephone number and a name. A receptionist would propose a room in other hotels of the chain if there is not a vacant room. Guests would loose the deposit if they cancel the reservation.**

Variations for generator output from the input sentence:-

> 'I was very tired, so I called a taxi and went home. The cab was warm and the driver was cool, so I gave him a big tip"

You gave a driver a big tip because he was cool and the taxi that you called was warm. You were tired so you went to your home.
You gave a driver a big tip. A taxi was warm. You called it. You were tired. You went to your home.
A driver was cool and the taxi that you called was warm so you gave him or her a big tip.
The driver to whom you gave a big tip was cool.
The cool driver to whom you gave a big tip.
The taxi that you called was warm.
You called a warm taxi because you were tired.
This tremendous enormous tip was received by this driver from you.
This cab was warm. It was called!
A big tip was received by a driver from you.
You gave a driver a tremendous gigantic tip because this cab that you called was warm. You were tired.
A big tip was received from you!
This driver received this big tip from you because this taxi that you called was warm.
You gave a driver a tremendous gigantic tip. A cab was warm. You called it.

Variations for generator output from the input sentence:-

'if I had known the big and fast motorbike you gave me was owned by her I would have liked it, because I do love her.'

I gave you a woman owner's big fast motorbike that you would have liked if you knew that she owned it.
You received this woman owner's big fast motorbike from me!
A woman owner's big fast motorbike that you would have liked if you knew that it was owned by her was given by me to you.
You love the woman owner who owned the big fast motorbike that I gave to you. You would have liked it if you knew that she owned it.
You love a woman!
You love a woman. She owned a big fast motorbike. You would have liked it if you knew that she owned it. You received it from me.
This woman is loved.
You would have liked a woman owner's big fast motorbike if you knew that she owned it!
This woman owner's big fast motorbike would have been liked if you knew that it was owned by her. You received it from me!
You would have liked this woman owner's big fast motorbike that I gave to you if you knew that she owned it.
If you knew that the big fast motorbike that I gave to you was owned by the woman owner whom you love then it would have been liked by you.
A big fast motorbike was owned!

Examples of abstract transformations:-

I like comics.
I do not dislike comics.
Clowns are happy.
Clowns are not sad.
A greedy salesman sold John a car.
John bought a car from a greedy salesman.
Mary and John went to a supermarket.
John went with Mary to a supermarket.
Romeo kissed Juliet.
Romeo gave Juliet a kiss.
A man claimed that he climbed Everest.
A man made a claim that he climbed Everest.
Brutus wounded Caesar with a knife.
Brutus stabbed Caesar with a knife.

Appendix B

Summary of Systems

- **ANA** pg.55. [Kukich, 1988]
  A system that summarises stock market movements. It uses domain specific templates derived from studying real life reports and uses a special lexicon containing the most used phrases.

- **ATN** section 3.6.5, pg.50,78,91. Augmented Transition networks. [Woods, 1970]
  ATNs were originally used in NL analysis but have more recently been used in generation. Generation systems that use ATNS are **BABEL**, **TEXT**, Shapiro's **SNePS** generator and Simmons and Slocum's generator (section 3.6.5).

- **BABEL** section 3.12.1. pg 68,77. [Goldman, 1975]
  BABEL produces English sentences from **CDT**. Uses discrimination networks to choose appropriate verbs to express CDT primitives.

- **BLAH** pg.53. [Weiner, 1980]
  A system which explains why deductions were taken on income tax returns. One of the first systems to use naturally occurring texts as a basis for text organisation. However the system is very domain restricted and small-scale.

- **Conceptual Dependency Theory, CDT** section 3.12.1 pg.76. [Schank and Abelson, 1977]
  Conceptual Dependency Theory was one of the first attempts at a representation which aimed to capture the content of NL sentences. This representation is used by the generation systems **BABEL** and **PAULINE**. CDT is based on a small set of primitive acts which are too restrictive for a serious large scale system.

- **Conceptual Graphs** section 3.12.2, pg.83. [Sowa, 1984]
  Conceptual graphs are a modern and well used representation for NL semantics. They have been used as the input to various generators including Sowa's (page 84) and Nogier and Zock's (page 86).

- **COMET** (CoOrdinated Multimedia Explanation Testbed). pg 44,57,69. [McKeown *et al.*, 1990]
  The COMET system provides explanation for equipment maintenance and repair in which text and graphics are integrated and coordinated. It has a generation component which is based on the **Schema** planning approach (see **TEXT** and **TAILOR**) and the Functional Unification Formalism **FUF**.

- **DIAMOND** pg.105. [Horacek, 1993]
  DIAMOND is the generation component of OFFICE-PLANNER a expert system that tries to solve office allocation problems. The system is similar to Horacek's **WEIBER**.

- **COMMUNAL** pg.45,59. [Fawcett, 1994]
  The COMMUNAL project (COnvivial Man-Machine Understanding through NAtural Language) is concerned with applying and developing Systemic functional linguistics (section 3.6.3) in a very large, fully working computer system. The generation component for the project is **GENESYS**.

- **DIOGENES** pg.40,70. [Nirenburg *et al.*, 1988]
  DIOGENES is unusual as it is an integrated system built using a blackboard architecture.

- **EDGE** pg.53,55. [Cawsey, 1990]
  The EDGE (Explanatory Discourse GEnerator) is a dialogue system used to explain how electrical circuits work (using a combination of graphics and text). It uses a domain dependent **Schema** approach to plan utterances.

- **EES** pg. 55,58. [Paris, 1991]
  The EES (Explainable Expert System) is a system which aims to generate explanations for expert systems. It uses a planner system based on **RST**. See also **XPLAIN** (the predecessor of EES).

- **EPICURE** pg.53. [Dale, 1990]
  This system concentrates on how to build referring expressions which pick out complex entities in connected discourse. The domain is that of cooking recipes and so the generator ensures that ingredients, for example, are referred to in a correct way.

- **Functional Unification Grammars/ Formalism (FUG/FUF)** section 3.6.2 pg 42,67. [Kay, 1979] [Elhadad and Robin, 1992]
  The process of functional unification has been used in many areas of NLP. It has been used for realisation in the **TEXT** and **TELEGRAM** systems and extended (into FUF) in the **COMET** project.

- **GENESYS** pg.45,73. [Fawcett *et al.*, 1993]
  GENESYS is the generation component of the COMMUNAL project. It is based on a very large systemic grammar which gives extremely good grammatical coverage. It is unclear, however, how this realiser is controlled. Demonstrations of the system, for example, constitute either a human making

decisions or them being made randomly (according to probabilities assigned to each decision).

- **GLINDA** pg.39,40. [Kantrowitz and Bates, 1992]
  GLINDA is the generator used for narration and inter-character communication in the OZ interactive fiction and virtual reality project. It is based on an integrated architecture where there are no divisions into planning and realisation components.

- **GOSSIP** 3.12.4 pg.96. [Iordanskaja *et al.*, 1991]
  The GOSSIP system (Generation of Operating System Summaries in Prolog) is based on the **Meaning Text Model**.

- **IGEN** section 3.8.3, pg.66. [Rubinoff, 1992]
  The IGEN generator allows the linguistic realisation component to feedback information to the planner thus preventing the generation gap problem. No information is given about the domain or scale of the generator but, as many lexical items have to be checked for each input concept, the mechanism is clearly not feasible for large-scale systems.

- **IMAGENE** pg.60. [Vander-Linden *et al.*, 1992]
  IMAGENE (Instruction MAnual GENErator) operates in the domain of describing the operation of cordless telephones. It uses systemic networks to build **RST** structures from a list of processes that need to be expressed. Neither the exact nature of the input nor where it comes from is detailed. IMAGENE uses **NIGEL** for final realisation.

- **JO¨¨CE** [Rambow and Korelsky, 1992]
  JOYCE is the generator of the Ulysses project concerned with describing software design diagrams. It uses a domain dependent **Schema**-based planner and a realiser based on the **MTM** model.

- **KALOS** pg.92. [Cline, 1994]
  KALOS uses the SNePS formalism throughout the generation process. It contains an 'after realisation' revision process to improve the original output. The generator works in the restricted domain of generating descriptions of the M68000 processor systems.

- **KAMP** pg.40. [Appelt, 1985]
  The KAMP (Knowledge and Modalities Planner) is a system which formulates the best noun phrase to describe a particular object. It was one of the first systems to break from the traditional separated architecture. The complex planning mechanism works from first principles and thus takes nearly an hour to produce one sentence. See also **TELEGRAM**.

- **KDS** pg.53. [Mann and Moore, 1981]
  The KDS (Knowledge Delivery System) uses a hill climbing search mechanism to combine clause length propositions into complex sentences. It works in the limited domain of fire emergency procedures.

- **KING** pg.101. [Jacobs, 1987]
  KING (Knowledge INtensive Generator) is a small-scale generator (with no given task or domain) which illustrates a knowledge intensive approach to generation. It illustrates the advantages of an input representation which has rich information.

- **LILOG** [Dobeš and Novak, 1991] [Novak, 1987] [Herzog and Rollinger, 1991]
  LILOG is a dialogue system in the domain of street descriptions and route planning. Its generation component can answer questions in this domain. The system uses an adaptation of Meteer's Text Structure (see **SPOKESMAN**) in a pipelined architecture. It uses KL-ONE[Brachman and Schmolze, 1985] for semantic representation.

- **LOQUI** pg.69. [Horacek, 1987]
  Helmut Horacek has been involved in the development of various generation systems (for example **WEIBER VIE-GEN** and **DIAMOND**) as well as the generator for the LOQUI project. It utilises a largely one to one relationship between its concepts ('epistemological primitives') and lexical items. When this relationship is one-to-many a discrimination network is used.

- **Meaning Text Theory/ Model** section 3.12.4 pg.94. [Mel'čuk and Polguère, 1970]
  Meaning Text Theory or Model is a linguistic theory which associates linguistic meanings and the texts that carry out those meanings. It has been used for generation in, for example, the systems **GOSSIP** and **JO'CE**.

- **MUMBLE** section 3.6.4 pg.39,47,63. [McDonald and Meteer, 1988]
  MUMBLE is a final realisation component which realises a semantic notation into surface form. All decisions about the structure of the text are assumed to be contained in this input specification which is then 'executed' as if it were a program in a special programming language. Although MUMBLE leaves a lot of the work to other modules, the **SPOKESMAN** program has been built to interface with it.

- **PAULINE** pg.51,70,78. [Hovy, 1988a]
  PAULINE (Planning And Uttering Language In Natural Environments) generates single sentences from an input expressed in **CDT**. PAULINE can produce a great variation of sentences from the same input using large set of stylistic features. PAULINE is based on a formative lexicon. Its main weakness is that it is based on CDT which is now rather outdated.

- **PENMAN** section 3.6.3, section 3.8.2 pg.45,60,64. [Mann, 1983a] [Bateman *et al.*, 1990]
  PENMAN was originally the name of a surface realiser based on the **NIGEL** systemic grammar and a specification language called SQL. The PENMAN project then evolved based on this NIGEL realiser but also incorporating higher level planning components (see section 3.7.4) and the PENMAN Upper Model (section 3.8.2).

- **PHRED** pg.101. [Jacobs, 1987]
  PHRED is the generator for a project which provides information about the Unix operating system. It is based on the idea of pattern concept (PC). pairs which relate concepts to phrases in the lexicon. The generation process involves the incremental *fetching* of possible PC pairs for a given concept and checking that the pattern meets given constraints.

- **POPEL** [Reithinger, 1991]
  POPEL (Production Of (Perhaps, Possibly, P..) Eloquent Language) is the generator in the XTRA (eXpert TRAnslator) system which provided NL access to expert systems. It is an interleaved system in which there is bidirectional interaction between the realiser (POPEL-HOW) and the **RST**-based planner (POPEL-WHAT). The system has also been integrated with a 'gesture generator' (ZORA) which points to relevant parts of the screen as text is generated.

- **PROTEUS** pg.53. [Davey, 1979]
  PROTEUS was a very early generation system which provided commentary for a game of tic-tac-toe. It is based on systemic networks.

- **Rhetorical Structure Theory** section 3.7.3 pg.36,55,61. [Mann and Thompson, 1987]
  RST was originally a formalism for describing the structure of text but has more recently been used to prescribe text order in generation planners (for example see **PENMAN,EES**, **POPEL,TECHDOC**). RST is most commonly used for building texts from a set of clause-sized input predicates.

- **SLANG** pg.45. [Patten, 1988]
  SLANG (Systemic Linguistic Approach to Natural language Generation) is a generator based on a **S˝˝STEMIC** grammar. Unlike **NIGEL** and **GENES˝˝S** however, it appears that this system did not progress beyond the prototype stage.

- **SPOKESMAN** section 3.8.1, pg.36,47,63. [Meteer, 1993]
  The SPOKESMAN system was built 'on top of' the **MUMBLE** realiser and shares many of its psychologically motivated assumptions (such as an indelible pipelined architecture). It aims to cross the the 'generation gap' by using a representation called 'Text Structure'. This text structure is built using the output of underlying application programs and provides a mechanism to take advantage of the expressiveness of NL while preventing the building of utterances that are not expressible.

- **SUNDIAL** [Youd and McGlashan, 1992]
  SUNDIAL (Speech UNderstanding in DIALogue) is a large collaboration project concerned with building real-time integrated computer systems capable of maintaining co-operative dialogues over the phone (e.g flight reservations, train enquiries). It is based on a unification formalism (Unification Categorical Grammar [Calder *et al.*, 1989]) which relies on a formative lexicon.

- **SUS˝˝** [Luckhardt, 1988]
  A machine translation system which can deal with several languages. It translates text into a semantic, universal representation (SEMSYN, [Rösner, 1988]) and then transforms it to the target language.

- **SUTRA** [Busemann, 1988]
  The SUTRA (SUrface TRAnsformation) system is the surface realiser of the HAM-ANS project. German is a language with a rich inflectional system: SUTRA is responsible for correct word order.

- **S˝˝STEMIC GRAMMAR** section 3.6.3, pg.44. [Halliday, 1985]
  Grammars based on Haliday's systemic functional linguistics have been the basis of generation systems such as **NIGEL**, **GENES˝˝S** and **SLANG**. Systemic networks have also been employed in other areas of generation (e.g in the **PENMAN** project).

- **TAILOR** pg.55. [Paris, 1993]
  TAILOR is a generation system that produces NL descriptions of devices in the knowledge base of RESEARCHER (a system which reads, remembers and generalises from patent abstracts). It adopts a schema-based approach to planning and uses a user model to select appropriate schemas. For example, depending on the level of expertise, TAILOR uses a *constituency* schema which describes the structure of an object, or the *process* schema which describes its operation.

- **TECHDOC** pg.60. [Rösner and Stede, 1992]
  TECHDOC is a generation system for the automatic production of technical manuals (more specifically car maintenance instructions). It uses a combination of **RST** and schema for planning and uses the **PENMAN** realiser.

- **TEXT** section 3.7.2 pg.43,53,61. [McKeown, 1985]
  TEXT was one of the first systems to produce multi-sentence discourse and was the first system to use schemas. It provides paragraph-length responses to questions about the structure of a military vehicle and weapon database.

- **VIE-GEN** [Buchberger and Horacek, 1988]
  VIE-GEN is the generation component of the German Dialogue system (VIE-LANG). It is based on a semantic network representation that uses primitives and discrimination networks to choose relevant lexical entries for these primitives.

- **WEIBER** section 3.8.4, pg.53,55,67. [Horacek, 1990]
  WEIBER is a financial consultation dialogue system which plans utterances such as ASK, ASSERT and RECOMMEND using schemas. It uses a novel level in the generation process which maps conceptual predicates to linguistic ones.

- **XPLAIN** pg.58. [Swartout, 1983].
  A predecessor of the **EES** system which aimed to allow expert systems to explain their decisions as well as simply presenting them.

- **YH** [Gabriel, 1988]
  The YH system generated lisp program descriptions on basis of their text and included comments. Gabriel concentrates on trying to to create vivid and continuous images; a process he termed 'deliberate writing'.

SYSTEM	Architecture	Control	Input	Planner	Realiser
ANA	PIPE	DEC	?	Schema	?
BLAH	?	?	?	Schema	?
BABEL	-	-	CDT-	-	D.net [1]
COMET	IL	DEC	Pred	Schema	FUG
DIOGENES	IT	Black	Pred	-	-
EDGE	?	DEC	Pred	Schema	?
EES	SEP	DEC	Pred	RST	?
EPICURE c	SEP	?	?	Schema	FUG
GENESYS	SEP	GRAM		-	Systemic
GLINDA	IT	-	Pred	-	-
GOSSIP	SEP	?	SemR	?	MTM
IGEN	IL	PRO	Pred	?	?
IMAGENE	PIPE	DEC	Pred	RST [2]	Systemic
JOYCE	PIPE	DEC	Pred	Schema	MTM
KALOS	IL	DEC	SNePS	Schema	FUG
KAMP	INT	PRO	Pred	-	-
KDS	SEP	?	?	Hill	?
KING	SEP	PRO	SEM	?	FUG
LILOG	PIPE	PRO	PRED		
LOQUI	?	?	?	?	d-net
MUMBLE	PIPE	PRO		-	
PAULINE	PIPE	PRO	CDT	-	FORM
PENMAN	PIPE	GRAM		-	Systemic
PHRED	?	PRO	Pred	-	
POPEL	IL	DEC	?	Pred	RST
SLANG	-	GRAM		-	Systemic
SPOKESMAN	PIPE	PRO	?	-	-
SUNDIAL	PIPE	PRO			UG [3]
SUTRA	PIPE	DEC			
TAILOR	PIPE	DEC	Pred	RST	ATN
TECHDOC	PIPE	DEC	-	RST [4]	Systemic
TEXT	PIPE	DEC	Pred	Schema	ATN
VIE-GEN	PIPE	DEC	SemR	-	d-net
WEIBER	PIPE	DEC	Pred	Schema	-
YH	PIPE	PRO	-	-	-

NOTES FOR TABLE

1. Restricted by number of primitives
2. Uses a systemic network for applying RST relations
3. Also uses a formative lexicon
4. Also uses Schema

KEY TO TABLE

Architecture

- SEP - Separated but not clear whether interleaved or pipelined.
- IL - Interleaved
- PIPE - Pipelined
- IT - Integrated

Control

- PRO - procedural
- DEC - declarative
- GRAM - grammar directed (used for realisers)
- black - Blackboard

Input

- Pred - predicate based. Note: some systems assume the existence of the predicates to be expressed, others have to retrieve the required ones from a knowledge base.
- SemR - a semantic network type of input (section 3.12)
- CDT - Conceptual Dependency Theory (section 3.12.1)
- SNePs - Semantic Network Processing System (section 3.12.3)

Planner

- Schema - Schema based (section 3.7.2)
- RST - RST based (section 3.7.3)
- HILL - based on hill climbing techniques

Realiser

- MTM - Based on the MTM model (section 3.12.4)
- D.net - Uses Discrimination networks (section 3.9.1)
- FORM - Uses a formative lexicon (section 3.6.6)
- ATN - Uses Augmented Transition Networks (section 3.6.5)

- Systemic - Uses Systemic grammar (section 3.6.3)
- FUG - Uses Functional Unification (section 3.6.2)

Other

- - not applicable
- (blank) cannot be placed in the above categories
- ? no information

Appendix C

The Evaluation Instructions

Instructions to the writers

Background

My PhD is concerned with Natural language processing - the manipulation of typed English text by a computer. This research will hopefully lead to products such as automatic text summarisers and translators. More specifically, my work is concerned with generating English text from the internal representation stored in the computer.

One of the things the system we are developing does is :-

1. Takes an input piece of text (ranging from one sentence to a couple of paragraphs)
2. Analyses this piece of text
3. For every thing and every event mentioned in the input text, the system builds a internal representation in its memory.
4. The generation module then takes each of these internal representations and rebuilds a piece of English to describe it.

So the overall effect is to repeat the input text from different angles.

Simple Example

For the input sentence 'the cat sat on the mat', the computer would build utterances describing 'the cat', 'the mat' and the 'sitting event'. For example, 'The cat that sat on the mat', 'The mat on which the cat sat' and 'The cat sat on the mat'.

Of course this is a very simple example and there are very few differing ways in which each utterance can be expressed.

When the input sentence is more involved however, the job becomes harder and there is more than one way of doing it. An utterance might be short and only contain some of the information, or in order to express the concept in more detail it might be necessary to produce longer utterances, perhaps more than one sentence.

For example, in the utterance :- 'If I had known the big and fast motorbike you gave me was owned by her I would have liked it, because I do love her',

Example utterances for the woman could be :-

- The woman who owned the big and fast motorbike that you gave me and whom I love.
- The woman I love who owned the motorbike
- The woman I love
- The woman that owned the motorbike that I gave you.
- The woman that owned the big and fast motorbike. You would like the motorbike if you knew she owned it because you love her.
- The woman that owned the big fast motorbike that I gave you. If you knew she owned it you would like it because you love her.

The Experiment

The idea for the experiment is to get people (i.e you !) to do the same task as the computer and then to ask another group of people if they can tell which utterances were computer or human generated. (This is a scaled down version of the Turing test).

So what I want you to do is read the following paragraph and produce a few utterances describing the different entities and events mentioned. i.e for each entity/ event (list given below) I want a few utterances which vary in depth of description (and therfore length) and perhaps in grammatical style.

> A car bomb exploded outside the Cabinet Office in Whitehall last night, 100 yards from 10 Downing Street. Nobody was injured in the explosion which happened just after 9pm on the corner of Downing Street and Whitehall. Police evacuated the area. First reports suggested that the bomb went off in a black taxi after the driver had been forced to drive to Whitehall. The taxi was later reported to be burning fiercely.

So I need utterances for the following:- the bomb, the cabinet office, 10 Downing street, the explosion, the corner, the injure event, the evacuation event, the report,

the taxi, the driver, the driving event, the forcing event, the suggesting event, the burning event and the reporting event.

I hope these instructions are clear ! It is difficult to describe the task without giving fuller examples. If I did this however, the experiment would not be valid as you responses would be affected by my examples. If you need clarification please e-mail me.

Thanks again for you help !

Instructions to the judges

This is the second part of an experiment to evaluate the LOLITA natural language generator.

The Task

I previously asked people to read a paragraph length piece of text and write English expressions to describe the different entities and events that are mentioned. I asked people to write utterances in differing levels of detail and styles.

The LOLITA system on which I am working also does this task. It analyses a piece of text and builds an internal representation of the meaning of each of the entities and events described. The generation module - the subject of this evaluation - then regenerates different English utterances from these representations.

The evaluation

Following is an example input text together with utterances produced according to the task above. Some of these utterances were generated by humans, some by the computer.

I want you to mark each of these utterances on two counts.

Firstly, I want to you to mark the acceptability of each sentence. This 'acceptability' is difficult to define and will differ from person to person. It is basically a measure of clarity and accuracy and subdivided as follows. :-

- **G** or Good: The utterance is a clear an accurate description of the entity or event.
- **O** or Okay : The description is acceptable but may be slightly inaccurate or grammatically clumsy.
- **U** or Unacceptable : The utterance is not a good description of the entity or event. It may contain inaccurate information, not enough information for the description or just not make sense.

- **B** or Best: use this code for the utterance which is the best within each group.

If you have time, it would also be useful if, when you use code **U**, you could include a quick comment to say why the utterance is unacceptable.

Secondly, I would like you to mark some of the utterances as follows :

- **C** or Computer : if you think the utterance was written by a computer
- **H** or Human : if you think the utterance was written by a human
- (no code): if you cannot tell.

I will not give any hints on the criteria for making this decision - I just want you to use your instincts.

Finally I would welcome comments about the difficulties you had in doing this evaluation.

The Input text

> A car bomb exploded outside the Cabinet Office in Whitehall last night, 100 yards from 10 Downing Street. Nobody was injured in the explosion which happened just after 9pm on the corner of Downing Street and Whitehall. Police evacuated the area. First reports suggested that the bomb went off in a black taxi after the driver had been forced to drive to Whitehall. The taxi was later reported to be burning fiercely.

The utterances:-

1. about the Bomb.

   a) The bomb that exploded b) The car bomb that exploded 100 yards from 10 Downing Street c) The bomb that went off in a black taxi d) The car bomb that went off on a corner outside the Cabinet Office and 100 yards from Downing Street in the black taxi that a driver drove to Whitehall e) The bomb that exploded just after 9:00 PM outside the cabinet office at Whitehall.

2. about the Cabinet Office.

   a) The Cabinet Office that is in Whitehall 100 yards from 10 Downing Street b) The Cabinet Office that is in Whitehall b) The Cabinet Office c) The Cabinet Office outside which the car bomb exploded d) The Cabinet Office was the scene of an explosion.

3. 10 Downing Street

   a) 10 Downing Street that is 100 yards from the Cabinet Office b) 10 Downing Street c) 10 Downing Street which nearly got done in by a bomb last night d) 10 Downing Street which is 100 yards from where the explosion happened (the corner of Whitehall and Downing Street 10 Downing Street) e) 10 Downing Street that was evacuated by police

4. The explosion

   a) The explosion that happened in a taxi which was passing through Whitehall, just outside the Cabinet Office. b) The explosion which happened just after 9pm c) The bomb explosion on a corner outside the Cabinet Office and 100 yards from 10 Downing Street in the black taxi that a driver drove to Whitehall d) The explosion that happened on the corner of Downing Street and Whitehall e) Explosion in Whitehall, but no one got hurt at all.

5. The corner

   a) The Whitehall and Downing Street corner. b) The corner of Downing Street and Whitehall where, last night at just after 9, a car bomb exploded in a taxi whose driver was being forced to drive there. c) The corner of Downing Street and Whitehall d) The corner on which the explosion that injured nobody happened e) The corner which is 100 yards from 10 Downing Street

6. The injure event

   a) Nobody was injured b) Injuries didn't happen even though the bomb went off in Whitehall c) The explosion just after 9pm on the corner of Whitehall and Downing Street injured nobody d) No injuries after Whitehall explosion! e) Nobody was injured by the bomb explosion outside Whitehall in a black taxi

7. The evacuation event

   a) An evacuation of the area near the cabinet office was carried out by the police after a bomb went off in a passing cab. b) Whitehall was evacuated last night after an explosion outside the Cabinet Office where there were no injuries. c) An area is being evacuated after an explosion. d) The bomb explosion did not cause an evacuation at the corner of Whitehall + Downing e) Police evacuated Whitehall

8. The reports

   a) The initial reports said the taxi driver was made to drive a bomb to the scene. b) The report which suggested that the driver of the black taxi in which the bomb went off had been forced to drive to Whitehall c) The report indicated no injuries because of the bomb explosion d) First reports suggested that at 9pm last night, after a forceful person forced a driver to drive a black taxi to Whitehall, a bomb went off in it e) The report that suggested that the bomb went off after the driver had been forced to drive to Whitehall

9. The taxi

   a) The black taxi whose driver had been forced to drive to Whitehall b) The taxi which was the subject of the first reports about where the bomb went off c) The taxi that was later reported to be burning fiercely d) The black taxi that a driver drove to Whitehall and that burnt fiercely e) The taxi that had been forced to drive to Whitehall and that contained a bomb which exploded.

10. The driver

    a) The driver of a black taxi that was forced to drive to Whitehall, last night, with a bomb b) The driver who drove a black taxi c) The driver who had driven the taxi that the bomb went off in d) The driver who had to go in his cab with the bomb to Whitehall. e) A driver. He or she drove a black taxi to Whitehall

11. The driving event

    a) A driver drove the black taxi that later burnt fiercely b) A black taxi was driven close to 10 Downing Street last night. In it was a bomb which exploded but there were no injuries. c) The drive that the driver had been forced to do to Whitehall d) Car bomb driven to Whitehall. e) Driving the cab the bomb went off in was a driver who had been made to go to Whitehall. His taxi was later seen blazing fiercely.

12. The forcing event

    a) A driver was forced to take his car to Whitehall last night. b) A forceful person forced a driver to drive a black taxi to Whitehall. c) A person or persons unknown forced a taxi driver to drive with a bomb in his car to the Cabinet Office in Whitehall last night. There it exploded just after 9pm causing no injuries but the taxi burned fiercely. d) The driver had been forced to drive to Whitehall e) The driver had been forced

13. The suggesting event

    a) Suggestions were made that the bomb went off in a taxi which caught fire, and the driver made to drive the bomb to Whitehall b) First reports suggested that the driver had been forced to drive to Whitehall c) First reports suggested that after a forceful person forced a driver to drive a black taxi to Whitehall, a car bomb went off in it d) It was suggested at first that the bomb went off in a black taxi e) The suggestion as to the cause of the explosion point to a black taxi, which was reported to be burning fiercely, was involved. The area has been evacuated by the police

14. The burning event

    a) Fierce flames were seen coming from the taxi after the explosion. b) On fire was the taxi used to bring the bomb which exploded. c) The black taxi that a driver drove to Whitehall and in which the bomb explosion happened burnt fiercely d) The taxi was burning due to the exploding bomb e) Fierce burning was later noticed in the cab.

15. The reporting event

a) Reporters reported later that the black taxi in which a bomb exploded burnt fiercely. b) A report said the driver had been forced to drive to Whitehall. c) The taxi was reported to be burning after an explosion which took place just after 9pm. d) It was reported that the taxi was burning fiercely e) It was reported that a taxi which was burning had been driven to the corner of Downing Street and Whitehall before it exploded.

Appendix D

Glossary

abstract transformations: A process in the solution of the LOLITA generator that involves transformations on the SemNet input before realisation by the plan-realiser. (See chapter 6.)

abstract types: A section of code which appears to the application programmer as independent of any particular representation. (See section 7.2.4.)

black box testing: A testing technique where only input and output conditions are used.

closed events: An event which is expressed as a noun phrase. For example:- 'The bomb explosion', 'The assasination of Kennedy'. (See section 5.7.2.)

concept: A concept in the LOLITA system is any node in the SemNet representation. Its *meaning* is given by that particular node together with the whole of the semantic network. (See section 1.5.2.)

context: The linguistic or non-linguistic environment in which language is used.

deep structure: An underlying level of representation which captures the meaning of language (cf. surface structure).

currying: A device used in functional programming languages where a sequence of structured arguments are replaced by a sequence of simpler ones. (see section 7.2.3.)

de-lexical verbs: A verb which adds little meaning to a sentence but provides syntactic structure. Examples are 'to have', 'to give', 'to do'. (See section 6.7.)

determiner: An item that co-occurs with a noun phrase to express such meaning as number or quantity (e.g., the, some, each).

events: An event node in SemNet represents some relation between concepts.

flexibility: A principle of Natural Language Engineering concerned with the ability to modify systems between tasks or domains. (See section 1.2.5.)

formative lexicon: A lexicon which includes information about how words can

be grammatically combined together.

Haskell: A functional programming language used for the implementation of LOLITA and its generator (see chapter 7).

higher order function: A function which takes either a function as an argument or delivers one as a result. (See section 7.2.2.)

language isomorphic. A concept is language isomorphic (LI) if its meaning can be exactly expressed by a lexical item.

lexicon: Information about the vocabulary of a language.

lexicalisation: The process of choosing surface level words or phrases to express deep level concepts.

general purpose base NL system: A NL system which has not been designed for a particular application or for a particular domain. The system must perform certain core tasks such as syntactic and semantic analysis of text. Specific applications can be built 'on top' of the general purpose base. LOLITA is an example of a general purpose base NL system. (See section 1.5.5).

generation: The term generation in the sense 'natural language(NL) generation' is used by different researchers to mean different things (see section 1.5.4). In this thesis, generation is the process of producing English utterances given the whole of LOLITA's SemNet representation as input.

the generation gap: A term used to describe the problems which can occur at the interface between traditional planning and realisation modules. The solution adopted in this work avoids this problem by shifting responsibility from the planner to the plan-realiser.

internal events: Events in the SemNet representation that cannot be directly expressed in any language. They usually arise when input text has not been fully disambiguated and are distinguished by having an internal action role which does not correspond to a verb in the surface language (such as **is_a, relate_, possrelate_, has_part, controls_** etc.)

lazy evaluation: A property of functional languages which allows unevaluated expressions to be passed to a function leaving the function to be responsible to evaluate them as and when their values are needed. (See section 7.2.5.)

LOLITA: A Natural Language general purpose base system based on the principles of Natural Language Engineering. LOLITA is the acronym for Large scale, Object-based, Linguistic Interactor, Translator and Analyser.

maintainability:A principle of Natural Language Engineering concerned with the usefulness of a system over a long period of time. (See section 1.2.4.)

meaning: The meaning of a node in the SemNet representation is defined by that node and the whole of the SemNet network. (See section 1.5.1.)

Natural Language Engineering,NLE: A practical approach to NLP which in-

corporates engineering ideas and practices from other disciplines. (See section 1.2.)

open events: Events in the SemNet representation that are expressed with an utterance containing a verb. (See section 5.7.2.)

planning: Traditionally, a component in the generation process which is responsible for deep level tasks such as choosing and ordering content. In the solution adopted in this work however, planning involves merely suggesting a series of instructions which are to be followed by the plan-realiser. (See section 3.7 for details of planning approaches in other systems and chapter 5 for information about planning in the LOLITA generator.)

plan-realiser: The realisation component in the LOLITA generator which produces utterances from the SemNet representation and a list of instructions provided by the planner. (See section 1.5.7 and chapter 5.)

realisation: The process of actually producing surface level language from some deeper level representation. (See chapter 5 for approaches to realisation.)

rhetorical structure theory: A formalism originally used to describe the structure of text which has been used to prescribe text order in generation planners. (See section 3.7.3.)

referential transparency: A property of functional languages which ensures that a function with a particular set of arguments will always return the same value whatever the context in which the evaluation takes place. (See section 7.2.1.)

robustness: A principle of Natural Language Engineering concerned with the ability of a system to recover from error conditions. (See section 1.2.3.)

SemNet: The semantic network representation used in the LOLITA system which forms the input to the LOLITA generator. (See section 4.3.2.)

schema: An approach to generation planning; schema identify patterns of predicates which can be combined to produce coherent text. (See section 3.7.2.)

scale: A principle of Natural Language Engineering concerned with the desire to build systems of a realistic size rather than the development of 'toy' prototypes. (See section 1.2.2.)

surface structure: A representation level which is close to how an utterance actually appears (e.g., comprising syntactic information, cf. deep structure).

syntax: Rules for sentence structure and word combinations for surface language.

systemic grammar: A grammar based on functional analysis as well as syntax and semantics. (See section 3.6.3.)

universal: A universal concept in SemNet represents quantification over all members of a set. E.g., the universal concept for 'cats' represents the set of all cats.

unification grammar: A formalism for grammar which involves unifying compatible collections of features to form a more specific description. (See section 3.6.2.)

usability: A principle of Natural Language Engineering concerned with building systems that perform tasks that real end-users require and that are user friendly. See section 1.2.8.

user model: A model of the user which contains information useful for specific tasks. In the generation process the user model may contain such information as what the user already knows.

white box testing: A testing method where knowledge of the internal working of the algorithms and implementation is used to find test cases.

Wizard of Oz experiments: Experiments where subjects are lead to believe they are operating an automated computer system when, in fact, they are interacting with a human simulating system behaviour.

Bibliography

[Andersen, 1992] P. M. Andersen, "Automatic Extraction of Facts from Press Releases to Generate News Stories", in *Proceedings of the 3rd Conference on Applications of NLP*, Italy, April 1992.

[ANLP-92, 1992] *Proceedings of the Third ACL Conference on Applied Natural Language Processing*, Trento, Italy, 1992.

[Appelt, 1983] D. E. Appelt, "TELEGRAM: A Grammar Formalism for Language Planning", in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, pages 595–599, Karlsruhe, West Germany, August 8-12, 1983.

[Appelt, 1985] D. E. Appelt, *Planning English Sentences*, Cambridge University Press, Cambridge, UK, 1985.

[Ballard and Jones, 1990] B. Ballard and M. Jones, "Computational Lingusitics", in Shapiro [1990].

[Barnett and Mani, 1990] J. Barnett and I. Mani, "Using Bidirectional Semantic Rules for Generation", in *Proceedings of the Fifth International Natural Language Generation Workshop*, pages 47–53, Dawson, PA, 1990.

[Bateman *et al.*, 1990] J. A. Bateman, R. T. Kasper, J. D. Moore, and R. A. Whitney, "A General Organization of Knowledge for Natural Language Processing: The Penman Upper Model", Unpublished technical report, USC Information Sciences Institute, 1990.

[Becker, 1975] J. D. Becker, "The Phrasal Lexicon", in *Proceedings of Theoretical Issues in Natural Language Processing (TINLAP-1)*, volume 1, pages 60–64, University of Illinois at Urbana-Champaign, July 1975, Also appears in BBN Tech Report 3081.

[Bennett *et al.*, 1990] K. H. Bennett, B. J. Cornelius, and D. J. Robson, "Software Maintenance", in J. McDermid, editor, *The Software Engineer's Reference Book*, Butterworth Scientific Ltd., 1990.

[Bird and Wadler, 1988] P. Bird and P. Wadler, *Introduction to Functional Programming*, Prentice Hall International (UK) Ltd., 1988.

[Bokma and Garigliano, 1992] A. F. Bokma and R. Garigliano, "Uncertainty Management through Source Control: A Heuristic Approach", in *Proceedings, International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Mallorca, Spain, July 1992.

[Boyer and Lapalme, 1985] M. Boyer and G. Lapalme, "Generating Sentences from Semantic Networks", in *Natural Language Understanding and Logic Programming: Proceedings of the First International Workshop on Natural Language Understanding and Logic Programming*, Amsterdam, Netherlands, 1985, North Holland.

[Brachman and Schmolze, 1985] R. Brachman and J. Schmolze, "An Overview of the KL-ONE knowledge representation system.", *Cognitive Science*, 9:171–216, 1985.

[Buchberger and Horacek, 1988] E. Buchberger and H. Horacek, "VIE-GEN: A Generator for German Texts", in McDonald and Bolc [1988], pages 166–204.

[Busemann, 1988] S. Busemann, "Surface Transformations During the Generation of Written German Sentences", in McDonald and Bolc [1988], pages 98–165.

[Calder *et al.*, 1989] J. Calder, M. Reape, and H. Zeevat, "An Algorithm for Generation in Unification Categorial Grammar", in *Proceedings of the Fourth European Meeting of the ACL*, pages 233–240, Manchester, UK, April 10-12, 1989.

[Carlsson and Hallgren, 1993] M. Carlsson and T. Hallgren, "FUDGETS: A Graphical User Interface in a Lazy Functional Language", in *Proceedings of Conference on Functional Programming Languages and Computer Architecture (FPCA 93)*, pages 321–330, June 1993.

[Cawsey, 1990] A. Cawsey, "Generating Explanatory Discourse", in Dale et al. [1990], pages 75–101.

[Cercone and Pattabhiraman, 1992] N. Cercone and T. Pattabhiraman, "Special Issue on Natural Language Generation: Introduction", *Computational Intelligence*, 8(1):72–76, February 1992.

[Cline, 1994] B. E. Cline, *Knowledge Intensive Natural Language Generation with Revision*, PhD thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, May 1994.

[COLING-88, 1988] *Proceedings of the 12th International Conference on Computational Linguistics (COLING-88)*, Budapest, August 22-27, 1988.

[Dale *et al.*, 1990] R. Dale, C. Mellish, and M. Zock, editors, *Current Research in Natural Language Generation*, Academic Press, New York, 1990.

[Dale *et al.*, 1992] R. Dale, E. H. Hovy, D. Rösner, and O. Stock, *Aspects of Automated Natural Language Generation*, Lecture Notes in Artificial Intelligence, 587, Springer-Verlag, Berlin, April 1992.

[Dale, 1990] R. Dale, "Generating Recipes: An Overview of Epicure", in Dale et al. [1990], pages 229–255, Also appears as EUCCS Tech Report RP-37, Edinburgh.

[DAR, 1993] DARPA, *Proceedings of the Fifth Message Understanding Conference*, Baltimore, Maryland, August 1993, Morgan Kaufmann Publishers.

[Davey, 1979] A. C. Davey, *Discourse Production*, Edinburgh University Press, Edinburgh, 1979.

[Delin *et al.*, 1994] J. Delin, A. Hartley, C. Paris, D. Scott, and K. V. Linden, "Expressing Procedural Relationships in Multilingual Instructions", in NLG94 [1994], pages 61–70.

[DiMarco and Hirst, 1993] C. DiMarco and G. Hirst, "A Computational Theory of Goal Directed Style in Syntax", *Computational lingusitics*, 19(3), September 1993.

[Dobeš and Novak, 1991] Z. Dobeš and H.-J. Novak, "From Constituent Planning to Text Planning", in *Proceedings of the Third European Workshop on Natural Language Generation*, pages 46–54, Judenstein, Austria, 1991.

[Dogru and Slagle, 1992] S. Dogru and J. R. Slagle, "A System That Translates Conceptual Structures Into English", in Nagle et al. [1992].

[Elhadad and Robin, 1992] M. Elhadad and J. Robin, "Controlling Content Realization with Functional Unification Grammars", in *Aspects of Automated Natural Language Generation* [1992], pages 89–104.

[Emery, 1994] S. K. Emery, "An Investigation into the Characterisation of Style in Everyday Text", Master's thesis, University of Sunderland, 1994.

[Engelmore and Morgan, 1988] R. Engelmore and A. Morgan, *Blackboard Systems*, Addison Wesley, 1988.

[Fawcett and Davies, 1992] R. P. Fawcett and B. L. Davies, "Monologue as a Turn in Dialogue: Towards an Integration of Exchange Structure and Rhetorical Structure Theory", in *Aspects of Automated Natural Language Generation* [1992], pages 151–166.

[Fawcett and Tucker, 1990] R. P. Fawcett and G. H. Tucker, "Demonstration of GENESYS: A very large, semantically based systemic functional generator", in *Proceedings of the 13th International Conference on Computational Linguistics (COLING-90)*, volume 1, pages 47–49, Helsinki, 1990.

[Fawcett *et al.*, 1993] R. P. Fawcett, G. H. Tucker, and Y. Q. Lin, "How a Systemic Functional Grammar works: The role of realization in realization", in Horacek and Zock [1993], pages 114–186.

[Fawcett, 1994] R. P. Fawcett, "On Moving On On Ontologies", in NLG94 [1994], pages 71–80.

[Fernandez, forthcoming 1995] M. Fernandez, "Spanish Generation in LOLITA", Master's thesis, Durham University, forthcoming, 1995.

[Fox and Long, 1995] M. Fox and D. Long, "Hierarchical Planning using Abstraction", *to appear in IEE Procs. Control Theory and Applications*, May 1995.

[Gabriel, 1988] R. P. Gabriel, "Deliberate Writing", in McDonald and Bolc [1988], pages 1–46.

[Galliers and Sparck Jones, 1993] J. Galliers and K. Sparck Jones, "Evaluating Natural Langauge Processing Systems.", Technical Report 291, Computer Laboratory, University of Cambridge, 1993.

[Garigliano and Long, 1988] R. Garigliano and D. Long, "Inheritance Hierarchies", in COLING-88 [1988].

[Garigliano *et al.*, 1992] R. Garigliano, R. G. Morgan, and M. H. Smith, "LOLITA : Progress Report 1.", Unpublished Research Report 12/92, Department of Computer Science, University of Durham, 1992.

[Garigliano *et al.*, 1993] R. Garigliano, R. G. Morgan, and M. H. Smith, "The LOLITA System as a Contents Scanning Tool", in *Proceedings of the 13th International Conference on Artificial Intelligence, Expert Systems and Natural Language Processing*, Avignon, France, May 1993.

[Garigliano *et al.*, 1995] R. Garigliano, R. Morgan, M. H. Smith, and S. Peyton Jones, "DEAR: Project summary", in *Proceedings of the 1st AIKMS Conference*, Oxford, March 1995.

[Garigliano, 1992] R. Garigliano, "A Computational Semantics for 'The'", Technical report, Department of Computer Science, Durham University, 1992.

[Giegerich and Hughes, 1994] R. Giegerich and J. Hughes, editors, *Dagstuhl-Seminar-Report 89*, May 1994.

[Goldman, 1975] N. M. Goldman, "Conceptual Generation", in R. C. Schank and C. K. Riesbeck, editors, *Conceptual Information Processing*, American Elsevier, New York, NY, 1975.

[Granville, 1994] R. Granville, "Building Underlying Structures for Multiparagraph Texts", in NLG94 [1994], pages 21–28.

[Grice, 1975] H. P. Grice, "Logic and conversation", in P. Cole and J. L. Morgan, editors, *Syntax and Semantics*, volume 3: Speech Acts, pages 41–58, Academic Press, New York, 1975.

[Grishman and Sterling, 1993] R. Grishman and J. Sterling, "Description of the PROTEUS system as used for MUC-5", in *Proceedings of the Fifth Message Understanding Conference* [1993].

[Grishman, 1994] R. Grishman, "Report on a MUC-6 Planning Meeting", December 1994.

[Grosz, 1977] B. J. Grosz, "The Representation and Use of Focus in a System for Understanding Dialog", in IJCAI-77 [1977], pages 67–76.

[Halliday, 1985] M. A. K. Halliday, *An Introduction to Functional Grammar*, Edward Arnold, London, 1985.

[Harrius, 1992] J. Harrius, "Text Generation in Expert Critiquing Systems using Rhetorical Structure Theory", in Nagle et al. [1992].

[Hasida *et al.*, 1987] K. Hasida, S. Ishizaki, and H. Isahara, "A Connectionist Approach to the Generation of Abstracts", in Kempen [1987], pages 149–156.

[Hazan and Morgan, 1992] J. Hazan and R. Morgan, "The Location Of Errors in Functional Programs", Unpublished Research Report 2/92, Department of Computer Science, University of Durham, 1992.

[Hazan *et al.*, 1993] J. Hazan, S. Jarvis, and R. Morgan, "Understanding LOLITA: Program Comprehension in Functional Languages", in *Proceedings of IEEE Conference on Program Comprehension*, Capri, Italy, June 1993.

[Herzog and Rollinger, 1991] O. Herzog and C. Rollinger, *Text Understanding In Lilog*, volume 546, Springer Verlang Lecture notes in AI, 1991.

[Hirst, 1981] G. Hirst, *Anaphora in Natural Language Understanding: A Survey*, Springer-Verlag, 1981.

[Hobbs, 1991] J. Hobbs, "Description of the TACITUS system as used for MUC-3", in *Proceedings of the Third Message Understanding Conference*, SRI International, Morgan Kaufmann Publishers, May 1991.

[Holyer, 1991] I. Holyer, *Functional Programming with Miranda*, Pitman Publishing, 1991.

[Horacek and Zock, 1993] H. Horacek and M. Zock, editors, *New Concepts in Natural Language Generation: Planning, Realization, and Systems*, Pinter Publishers, New York, 1993.

[Horacek, 1987] H. Horacek, "Choice of Words in the Generation Process of a Natural Language Interface", *Applied Artificial Intelligence*, 1(2):117–132, 1987.

[Horacek, 1990] H. Horacek, "The Architecture of a Generation Component in a Complete Natural Language Dialog System", in Dale et al. [1990], pages 193–227.

[Horacek, 1992] H. Horacek, "An Integrated View of Text Planning", in *Aspects of Automated Natural Language Generation* [1992], pages 29–44.

[Horacek, 1993] H. Horacek, "Decision Making Throughout the Generation Process in the Systems WISBER and DIAMOD", in Horacek and Zock [1993], pages 215–237.

[Horacek, 1994] H. Horacek, "Building Another Bridge over the Generation Gap", in NLG94 [1994], pages 221–224.

[Hovy *et al.*, 1992] E. H. Hovy, J. Lavid, E. Maier, V. Mittal, and C. L. Paris, "Employing Knowledge Resources in a New Text Planner Architecture", in *Aspects of Automated Natural Language Generation* [1992], pages 57–72.

[Hovy, 1988a] E. H. Hovy, "Generating Language with a Phrasal Lexicon", in McDonald and Bolc [1988], pages 353–384.

[Hovy, 1988b] E. H. Hovy, *Generating Natural Language under Pragmatic Constraints*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1988, Based on PhD thesis, Yale University.

[Hovy, 1991] E. H. Hovy, "Approaches to the Planning of Coherent Text", in Paris et al. [1991], pages 83–102.

[Hovy, 1993] E. H. Hovy, "Automated Discourse Generation Using Discourse Structure Relations.", *Artificial Intelligence*, 63:341–385, 1993.

[Hudak *et al.*, 1994] P. Hudak, S. P. Jones, and P. Wadler, "Report on the Functional Programming Language Haskell, Version 1.2", May 1994, ACM SIGPLAN Notices 27.

[Hughes, 1989] J. Hughes, "Why Functional Programming Matters", *The Computer Journal*, 32(2), 1989.

[IJCAI-77, 1977] *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, MIT, Cambridge, MA, August 1977.

[Iordanskaja *et al.*, 1991] L. Iordanskaja, R. Kittredge, and A. Polguère, "Lexical Selection and Paraphrase in a Meaning-Text Generation Model", in Paris et al. [1991], pages 293–312.

[Jacobs and Rau, 1985] P. S. Jacobs and L. F. Rau, "Ace: Associating Language with Meaning", in T. O'Shea, editor, *Advances in Artificial Intelligence*, pages 295–304, North-Holland, Amsterdam, 1985.

[Jacobs and Rau, 1990] P. S. Jacobs and L. F. Rau, "SCISOR: Extracting information from On-Line News", *Communications of the ACM*, 33(11), November 1990.

[Jacobs, 1985] P. S. Jacobs, "PHRED: A Generator for Natural Language Interfaces", *Computational Linguistics*, 11(4):219–242, October-December 1985, Revised version of Berkeley Tech Report CSD-85-198.

[Jacobs, 1987] P. S. Jacobs, "Knowledge-Intensive Natural Language Generation", *Artificial Intelligence*, 33(3):325–378, November 1987.

[Jones, 1994] C. Jones, *Dialogue Structure Models : An Engineering Approach to the Analysis and Generation of Natural English Dialogues*, PhD thesis, Department of Computer Science, Durham University, 1994.

[Joshi, 1987] A. K. Joshi, "The Relevance of Tree Adjoining Grammar to Generation", in Kempen [1987], pages 233–252.

[Kantrowitz and Bates, 1992] M. Kantrowitz and J. Bates, "Integrated Natural Language Generation Systems", in *Aspects of Automated Natural Language Generation* [1992], pages 13–28.

[Kay, 1979] M. Kay, "Functional Grammar", in *Proceedings of the 5th Annual Meeting of the Berkeley Linguistic Society*, pages 142–158, Berkeley, CA, February 17-19, 1979.

[Kempen, 1987] G. Kempen, editor, *Natural Language Generation: New Results in Artificial Intelligence, Psychology and Linguistics*, NATO ASI Series – 135, Martinus Nijhoff Publishers, Boston, Dordrecht, 1987.

[Kramsky, 1972] J. Kramsky, *The Article and the Concept of Definiteness in Language*, 1972.

[Kukich, 1988] K. Kukich, "Fluency in Natural Language Reports", in McDonald and Bolc [1988], pages 280–311.

[Lientz and Swanson, 1980] B. Lientz and E. Swanson, *Software Maintenance Management*, Addison-Wesley, 1980.

[Linde and Labov, 1975] C. Linde and W. Labov, "Spatial Networks as a Site for the Study of Language and Thought.", *Language*, 57:924–939, 1975.

[Long and Fox, 1995] D. Long and M. Fox, "A Hybrid Architecture for Rational Agents", in C.Thornton and S.Torrance, editors, *Hybrid Models of Cognition*, AISB, 1995.

[Long and Garigliano, 1994] D. Long and R. Garigliano, *Reasoning by Analogy And Causality: A Model and Application*, Ellis Horwood, 1994.

[LRE, 1992] "Linguistic Research and Engineering European Programme", 1992.

[Luckhardt, 1988] H.-D. Luckhardt, "Generation of Sentences from a Syntactic Deep Structure with a Semantic Component", in McDonald and Bolc [1988], pages 205–255.

[Maier and Hovy, 1993] E. Maier and E. H. Hovy, "Organising Discourse Structure Relations using Metafunctions", in Horacek and Zock [1993], pages 69–86.

[Mann and Moore, 1981] W. C. Mann and J. A. Moore, "Computer Generation of Multiparagraph English Text", *American Journal of Computational Linguistics*, 7(1):17–29, 1981.

[Mann and Thompson, 1987] W. C. Mann and S. A. Thompson, "Rhetorical Structure Theory: Description and Construction of Text Structures", in Kempen [1987], pages 85–96, Also appears as USC/Information Sciences Institute Tech Report RS-86-174, October 1986.

[Mann, 1983a] W. C. Mann, "An Overview of the NIGEL Text Generation Grammar", in *Proceedings of the 21st Annual Meeting of the ACL*, pages 79–84, Massachusetts Institute of Technology, Cambridge, MA, June 15-17, 1983.

[Mann, 1983b] W. C. Mann, "An Overview of the Penman Text Generation System", in *Proceedings of the Third National Conference on Artificial Intelligence (AAAI-83)*, pages 261–265, Washington, DC, August 22-26, 1983.

[Marcus, 1980] M. Marcus, *A Theory of Syntactic Recognition for Natural Language.*, MIT Press, 1980.

[Matthiessen, 1991] C. Matthiessen, "Lexico(Grammatical) Choice in Text Generation", in Paris et al. [1991], pages 249–292.

[McDonald and Bolc, 1988] D. D. McDonald and L. Bolc, *Natural Language Generation Systems*, Springer-Verlag, New York, NY, 1988.

[McDonald and Busa, 1994] D. D. McDonald and F. Busa, "On The Creative use of Language: The Form of Lexical Resources", in NLG94 [1994], pages 81–90.

[McDonald and Meteer, 1988] D. D. McDonald and M. Meteer, "From Water to Wine: Generating Natural Language Text from Today's Applications Programs", in *Proceedings of the Second ACL Conference on Applied Natural Language Processing*, pages 41–48, Austin, TX, February 9-12, 1988.

[McDonald *et al.*, 1987] D. D. McDonald, M. M. Meteer, and J. D. Pustejovsky, "Factors Contributing to Efficiency in Natural Language Generation", in Kempen [1987], pages 159–182.

[McDonald, 1990] D. D. McDonald, "Natural Language Generation", in Shapiro [1990], pages 642–655.

[McDonald, 1991] D. D. McDonald, "On the Place of Words In the Generation Process", in Paris et al. [1991], pages 227–248.

[McKeown and Elhadad, 1991] K. R. McKeown and M. Elhadad, "A Contrastive Evaluation of Functional Unification Grammar for Surface Language Generation: A Case Study in Choice of Connectives", in Paris et al. [1991], pages 351–396.

[McKeown and Swartout, 1988] K. R. McKeown and W. R. Swartout, "Language Generation and Explanation", in Zock and Sabah [1988], chapter 1, pages 1–52.

[McKeown *et al.*, 1990] K. R. McKeown, M. Elhadad, Y. Fukumoto, J. Lim, C. Lombardi, J. Robin, and F. A. Smadja, "Natural Language Generation in COMET", in Dale et al. [1990], pages 103–139.

[McKeown, 1985] K. R. McKeown, *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text*, Cambridge University Press, Cambridge, 1985.

[Meehan, 1977] J. R. Meehan, "TALE-SPIN: An interactive program that writes stories", in IJCAI-77 [1977], pages 91–98.

[Mellish, 1988] C. Mellish, "Natural Language Generation from Plans", in Zock and Sabah [1988], chapter 7, pages 131–145, Also appears as CSRP Tech Report 031, University of Sussex.

[Mel'čuk and Polguère, 1970] I. Mel'čuk and A. Polguère, "Towards a Functioning Meaning-Text Model of Language", *Linguistics*, 57:10–47, 1970.

[Meteer, 1992] M. Meteer, "Portable Natural Language Generation using SPOKESMAN", in ANLP-92 [1992], pages 237–238.

[Meteer, 1993] M. Meteer, *Expressibility and the Problem of Efficient Text Planning*, Francis Pinter Publishers, London, 1993.

[Meteer, 1994] M. Meteer, "Generating Event Discriptions with SAGE : A Simulation and Generation Environment", in NLG94 [1994], pages 99–108.

[Miller, 1990] G. Miller, "WordNet: An On-Line Lexical Database", *International Journal of Lexicography*, 3(4), 1990.

[Moore and Swartout, 1991] J. D. Moore and W. R. Swartout, "A Reactive Approach to Explanation: Taking the User's Feedback into Account", in Paris et al. [1991], pages 3–48.

[Morgan and Jarvis, 1995] R. G. Morgan and S. A. Jarvis, "Profiling Large-Scale Lazy Functional Programs", in *Proceedings of the Conference on High Performance Functional Computing*, Denver, USA., April 1995.

[Morgan *et al.*, 1994] R. G. Morgan, M. H. Smith, and S. Short, "Translation by Meaning and Style in LOLITA", in *Proceedings of Machine Translation: Ten years on*, Cranfield University and the British Computer Society, November 1994.

[Mykowiecka, 1991a] A. Mykowiecka, "Natural-Language Generation — an Overview", *International Journal of Man-Machine Studies*, 34(4):497–511, April 1991.

[Mykowiecka, 1991b] A. Mykowiecka, "Text Planning — How to Make Computers Talk in Natural Language", *International Journal of Man-Machine Studies*, 34(4):575–591, April 1991.

[Nagle *et al.*, 1992] T. Nagle, J. Nagle, L. Gerholz, and P. Elklund, editors, *Conceptual Structures: Current Research and Practice*, Ellis Horwood, New York, NY, 1992.

[Nicolov *et al.*, 1995] N. Nicolov, C. Mellish, and G. Ritchie, "Sentence Generation from Conceptual Graphs", in *Proceedings of the International Conference on Conceptual Structures, ICCS'95*, Santa Cruz, August 1995.

[Nirenburg and Nirenburg, 1988] S. Nirenburg and I. Nirenburg, "A Framework for Lexical Selection in Natural Language Generation", in COLING-88 [1988], pages 471–475.

[Nirenburg *et al.*, 1988] S. Nirenburg, R. McCardell, E. Nyberg, P. Werner, E. Kenschaft, S. Huffman, and I. Nirenburg, "DIOGENES-88", Technical Report CMU-CMT-88-107, Center for Machine Translation, Carnegie Mellon University, 1988.

[Nirenburg *et al.*, 1989] S. Nirenburg, V. Lesser, and E. Nyberg, "Controlling a Language Generation Planner", in *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, volume 2, pages 1524–1530, Detroit, MI, August 20-25, 1989.

[NLG94, 1994] *Proceedings of the Seventh International Workshop on Natural Language Generation*, Nonantum Inn, Kennebunkport, Maine, June 21-24 1994.

[Nogier and Zock, 1992] J. Nogier and M. Zock, "Lexical Choice as Pattern Matching", in Nagle et al. [1992].

[Novak, 1987] H.-J. Novak, "Strategies for Generating Coherent Descriptions of Object Movements in Street Scenes", in Kempen [1987], pages 117–132.

[Novak, 1993] H.-J. Novak, "Ontology and Lexical choice", in Horacek and Zock [1993], pages 70–93.

[Panaget, 1994] F. Panaget, "Using a Textual Representation Level Component in the Context of Discourse and Dialogue Generation", in NLG94 [1994], pages 127–136.

[Paris and McKeown, 1987] C. L. Paris and K. R. McKeown, "Discourse Strategies for Describing Complex Physical Objects", in Kempen [1987], pages 97–116.

[Paris *et al.*, 1991] C. L. Paris, W. R. Swartout, and W. C. Mann, editors, *Natural Language Generation in Artificial Intelligence and Computational Linguistics*, Kluwer Academic Publishers, Boston, 1991.

[Paris, 1991] C. L. Paris, "Generation and Explanation: Building an Explanation Facility for the Explainable Expert Systems Framework", in Paris et al. [1991], pages 49–82.

[Paris, 1993] C. L. Paris, *User Modelling in Text Generation*, Francis Pinter Publishers, London, 1993.

[Parker, 1994] B. Parker, "Spell Checking in LOLITA", Master's thesis, Durham University, 1994.

[Patten, 1986] T. Patten, *Interpreting Systemic Grammar as a Computational Representation: A Problem Solving Approach to Text Generation*, PhD thesis, Edinburgh University, Department of Artificial Intelligence, 1986.

[Patten, 1988] T. Patten, *Systemic text generation as problem solving*, Cambridge University Press, New York, 1988, Based on PhD Thesis [Patten, 1986].

[Peyton Jones, 1989] S. L. Peyton Jones, "Parallel Implementation of Functional Languages", *The Computer Journal: Special Issue on Lazy Functional Programming*, 32(2), April 1989.

[Pustejovsky and Nirenburg, 1987] J. D. Pustejovsky and S. Nirenburg, "Lexical Selection in the Process of Language Generation", in *Proceedings of the 25th Annual Meeting of the ACL*, pages 201–206, Stanford University, Stanford, CA, July 6-9, 1987.

[Rambow and Korelsky, 1992] O. Rambow and T. Korelsky, "Applied Text Generation", in ANLP-92 [1992], pages 40–47.

[Reiter and Mellish, 1993] E. Reiter and C. Mellish, "Optimizing the Costs and Benefits of Natural Language Generation", in *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, Chambery. France, 1993.

[Reiter, 1990] E. Reiter, "Generating Descriptions that Exploit a User's Domain Knowledge", in Dale et al. [1990], pages 257–285

[Reithinger, 1991] N. Reithinger, "POPEL: A Parallel and Incremental Natural Language Generation System", in Paris et al. [1991], pages 179–200.

[Rösner and Stede, 1992] D. Rösner and M. Stede, "Customizing RST for the Automatic Production of Technical Manuals", in *Aspects of Automated Natural Language Generation* [1992], pages 199–214.

[Rösner, 1988] D. Rösner, "The Generation System of the SEMSYN Project: Towards a task-independent generator", in M. Zock and G. Sabah, editors, *Advances in Natural Language Generation: An Interdisciplinary Perspective*, volume 2, chapter 6, pages 76–85, Ablex Publishing Corporation, Norwood, NJ, 1988.

[Rubinoff, 1992] R. Rubinoff, "Integrating Text Planning and Linguistic Choice", in *Aspects of Automated Natural Language Generation* [1992], pages 45–56.

[Sacerdoti, 1977] E. Sacerdoti, *A Structure for Plans and Behaviour*, North Holland, 1977.

[Schank and Abelson, 1977] R. C. Schank and R. P. Abelson, *Scripts, Plans, Goals and Understanding*, Erlbaum, Hillsdale, N.J, 1977.

[Schank, 1975] R. C. Schank, *Conceptual Information Processing*, North Holland, Amsterdam, 1975.

[Scott and de Souza, 1990] D. R. Scott and C. S. de Souza, "Getting the Message Across in RST-based Text Generation", in Dale et al. [1990], pages 47–73.

[Shapiro and the SNePS Implementation Group, 1993] S. C. Shapiro and the SNePS Implementation Group, "SNePS 2.1 User's Manual", Technical report, State University of New York at Buffalo, Buffalo, NY, 1993.

[Shapiro, 1979] S. C. Shapiro, "The SNePS Semantic Network Processing System", in N. Findler, editor, *Associative Networks: Representation and use of Knowledge by Computers*, Academic Press, 1979.

[Shapiro, 1982] S. C. Shapiro, "Generalized ATN Grammars for Generation from Semantic Networks", *Computational Linguistics*, 8:12–26, 1982.

[Shapiro, 1990] S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, John Wiley and Sons, 1990.

[Short and Garigliano, 1993] S. Short and R. Garigliano, "The Representation of Location in LOLITA", Unpublished research report, Department of Computer Science, University of Durham, March 1993.

[Short, forthcoming 1995] S. Short, *Semantic Representation and Analysis in the LOLITA System*, PhD thesis, Department of Computer Science, University of Durham, forthcoming, 1995.

[Simmons and Slocum, 1972] R. F. Simmons and J. Slocum, "Generating English Discourse from Semantic Networks", *Communications of the ACM*, 15(10):891–903, October 1972.

[Smith *et al.*, 1995] M. H. Smith, R. Garigliano, and R. G. Morgan, "The DEAR project: A Natural Language Interface to Databases", in *Submitted to the 'Office Systems, translation, multilingual interfaces and software cluster' of the Language Engineering Convention*, London, October 1995.

[Sommerville, 1992] I. Sommerville, *Software Engineering*, Addsion-Wesley Publishing Company, 1992.

[Sondheimer *et al.*, 1989] N. K. Sondheimer, S. Cumming, and R. Albano, "How to Realize a Concept: Lexical Selection and the Conceptual Network in Text Generation", Technical Report RS-89-248, USC Information Sciences Institute, 1989, Also appears in the proceedings of the Workshop on Theoretical and Computational Issues in Lexical Semantics, Brandeis University, April 1989 and in Machine Translation 5(1):57-78, March 1990.

[Sowa, 1983] J. F. Sowa, "Generating Language from Conceptual Graphs", *Computers and Mathematics with Applications*, 9(1):29–43, 1983.

[Sowa, 1984] J. F. Sowa, *Conceptual Structures (Information Processing in Mind and Machine)*, Addison-Wesley, 1984.

[Stede, 1994] M. Stede, "Lexicalization in Natural Language Generation: A Survey", *Artificial Intelligence Review*, 1994.

[Swartout, 1983] W. R. Swartout, "XPLAIN: A System for Creating and Explaining Expert Consulting Programs", *Artificial Intelligence*, 21(3):285–325, September 1983, Also appears as USC Information Sciences Institute Tech Report RS-83-4.

[The AQUA Team, 94] The AQUA Team, "The Glorious Haskell Compilation System, User's Guide", 94.

[Tomita, 1986] M. Tomita, *Efficient Parsing of NL: A Fast Algorithm for Practical Systems*, Kluwer Academic Publishers, Boston, Ma, 1986.

[Turner, 1982] D. Turner, "Recusion Equations as a Programming Language", in Darlington, editor, *Functional Programming and Its Applications*, Cambridge University Press, 1982.

[Turner, 1987] D. Turner, "An Introduction to Miranda", in S. Peyton-Jones, editor, *The Implementation of Functional Programming Languages*, Prentice Hall INternational (UK) Ltd., 1987.

[van Rijn, 1992] A. van Rijn, "Generating Language from Conceptual Dependency Graphs", in Nagle et al. [1992].

[Vander-Linden *et al.*, 1992] K. Vander-Linden, S. Cumming, and J. Martin, "Using System Networks to Build Rhetorical Structure", in Vander-Linden [1992], pages 183–198.

[Velardi *et al.*, 1988] P. Velardi, M. T. Pazienza, and M. De'Giovanetti, "Conceptual Graphs for the Analysis and Generation of Sentences", *IBM Journal of Research and Development*, 32(2):251–268, March 1988.

[Viegas and Bouillon, 1994] E. Viegas and P. Bouillon, "Semantic Lexicons: The Cornerstone for Lexical Choice in Natural Language Generation", in NLG94 [1994], pages 91–98.

[Wadler, 1992] P. Wadler, "The Essence of Functional Programming", in *Invited talk : 19th Annual Symposium on Principles of Programming Languages*, Sante Fe, New Mexico, January 1992.

[Wang and Garigliano, 1992] Y. Wang and R. Garigliano, *An Intelligent Tutoring System for Handling Errors Caused by Transfer*, Lecture Notes in Artificial Intelligence, 608, Springer-Verlag, Montreal, Canada, 1992.

[Wang, 1994] Y. Wang, *An Intelligent Computer-based Tutoring Approach for the Management of Negative Transfer*, PhD thesis, Department of Computer Science, Durham University, 1994.

[Wanner, 1994] L. Wanner, "Building Another Bridge over the Generation Gap", in NLG94 [1994], pages 137–144.

[Weiner, 1980] J. L. Weiner, "BLAH, a System which Explains its Reasoning", *Artificial Intelligence*, 15(1):19–48, 1980.

[Woods, 1970] W. Woods, "Transistion Network Grammars for Natural Language Analysis", *Communications of the ACM*, 13(10):591–606, October 1970.

[Yazdani, 1987] M. Yazdani, "Reviewing as a Component of the Text Generation Process", in Kempen [1987], pages 183–190.

[Youd and McGlashan, 1992] N. J. Youd and S. McGlashan, "Generating Utterances in Dialogue Systems", in *Aspects of Automated Natural Language Generation* [1992], pages 135–150.

[Zock and Sabah, 1988] M. Zock and G. Sabah, editors, *Advances in Natural Language Generation: An Interdisciplinary Perspective*, volume 1, Ablex Publishing Corporation, Norwood, NJ, 1988.

[Zukerman and Pearl, 1986] I. Zukerman and J. Pearl, "Comprehension Driven Generation of Meta-technical Utterances in Math Tutoring", in *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 606–611, Philadelphia, PA, August 11-15, 1986.