GOTHENBURG MONOGRAPHS IN LINGUISTICS 14

A LOGICAL APPROACH TO COMPUTATIONAL CORPUS LINGUISTICS

Torbjörn Lager

Department of Linguistics, Göteborg University, Sweden, 1995

GOTHENBURG MONOGRAPHS IN LINGUISTICS 14

A LOGICAL APPROACH TO COMPUTATIONAL CORPUS LINGUISTICS

Torbjörn Lager

Doctoral Dissertation

publicly defended in Stora Hörsalen,
Humanisten, Göteborg University,
on December 16th, 1995, at 10.00
for the degree of Doctor of Philosophy

Department of Linguistics,
Göteborg University,
Sweden, 1995

ABSTRACT

The purpose of this thesis is to build a *corpus theory development environment* – to discuss its design, use, and implementation. The proposed system is based on a logical approach to computational corpus linguistics where sentences of logic are used to express statements about texts and logical inference is used to manipulate these sentences in order to analyse the texts.

The thesis demonstrates the remarkable ease with which the functionalities needed in a corpus system can be implemented when based upon adequate means of representing, querying, and reasoning. The proposed system implements hand coding, searching, concordancing, parsing, counting, tabling, collocating, automatic part-of-speech tagging, lemmatizing, excerpting, interpreting, treebanking, explanation, and various kinds of learning.

By linking all this functionality into a common representational framework characterised by high expressive power, declarativity, and explicit reasoning strategies, and by embedding the whole concept in a particular philosophical and methodological context, including an ontology of text, an analysis of the notion of theory, an explication of the notion of truth, and other foundational issues, we arrive at an interactive system which is multi-functional and general, yet simple, consistent, and highly usable.

Apart from being interesting from a practical point of view, the development of such a system raises intriguing philosophical and methodological questions: What is a corpus text? What is a corpus theory? What does it mean to develop a corpus theory? What does it mean for a corpus theory to be true about a corpus text? What is the link between the truth of such a theory and its usefulness for natural language processing purposes? These and related questions are discussed in the thesis.

The system exists in a prototype implementation and the thesis contains numerous examples from this implementation in action.

KEY WORDS: Corpus linguistics, Corpus tools, Grammar, Grammar development, Logic programming

Preface

This is my Ph.D. thesis. It is intended for an audience consisting of (computational) corpus linguists and theoretical (computational) linguists. I have put "computational" within brackets, to indicate that a background and interest in computer science is optional. The chapters covering the basics of computational logic are written for linguists with no such backgrounds, as well as for corpus linguists who have backgrounds in computer science but who prefer to have Σs and Πs rather than ∃s and ∀s staring at them from the pages when they open a book. Thus, I should let logic programming aficionados know that they can safely skip these parts, and I'd better tell the statistics people that there *are* a couple of Σs and Πs in this book, so hang on.

My sincere thanks and deeply felt gratitude go to the following persons: To my supervisor Jens Allwood, whose enthusiasm and generosity, coupled with a nagging criticism, has kept me on the track all the way to the end. To Joakim Nivre, assistant supervisor, whom I have often felt understood my ideas better than I did myself and whose sound advice has been invaluable to my work. To assistant supervisor Björn Haglund, philosopher, whom I have known since I first begun my studies at the university, and whose deep and clear thinking has always impressed me. To my thesis opponent Martin Kay, for a very rewarding discussion at the public defence of my thesis. To the members of my thesis committee – Lars Ahrenberg, Christian Bennet, and Robin Cooper – for letting me become

a doctor. To Lars and Robin also for the many stimulating discussions at various points during the development of my work. To Björn Beskow, inspiring friend, with whom I have discussed many of my ideas when they were still in their infancy. To Peter Campbell who did wonders to my English, proof reading my manuscript, correcting my style. To all colleagues at the department of linguistics in Göteborg, for providing such an enjoyable atmosphere to work in. To my parents, for always supporting me in all the things I have ever undertaken. To August, Fabian, Hillevi, and Malin, my dearest family – I hereby dedicate this thesis to you.

On the cover of this book is a picture of Saint Jerome (Eusebius Hieronymus), c.347-420, painted by Albrecht Dürer 500 years ago. Jerome was a Father of the Church, whose great work was the translation of the Bible into Latin. At that time there were no computers. In the picture, sitting in his lab, Jerome seems to be accessing his intuitions, while pointing at a corpus.

Göteborg in May, 1996

T.L.

Contents

CHAPTER 1 ***Introduction 1***

1.1 Introduction **1**
- 1.1.1 The dichotomy is false … **4**
- 1.1.2 … but still practised **5**

1.2 The State of the Art of Corpus Tools and Systems **6**
- 1.2.1 A functional view of corpus tools **6**
- 1.2.2 Existing tools for the corpus linguist **8**
- 1.2.3 Existing corpus tools – a critical discussion **12**

1.3 Related Tools and Systems **19**
- 1.3.1 Grammar development environments **19**
- 1.3.2 Knowledge acquisition systems **20**

1.4 The Aims of This Thesis **21**

1.5 What This Thesis Contains **24**
- 1.5.1 Chapter by chapter **24**
- 1.5.2 Some recurring themes **27**

1.6 A Short Note about Corpora **28**

1.7 Summary and Conclusion **28**

CHAPTER 2 ***Philosophy of Corpus Linguistics 30***

2.1 Introduction **30**

2.2 The Ontology of Text **32**
- 2.2.1 Texts, facts, and states of affairs **32**
- 2.2.2 Points, segments, properties, property values, and relations **33**

2.3 Statements and Theories **35**
- 2.3.1 Statements **35**
- 2.3.2 Theories **37**

2.4 Truth and Truthlikeness **37**

2.5 Corpus Linguistics as an Activity **39**
- 2.5.1 Three things linguists do **40**
- 2.5.2 The dynamics of theory development in linguistics **41**

2.6 Summary and Conclusion **42**

CHAPTER 3 ***The Logic of TagLog 44***

3.1 Introduction **44**
3.2 Requirements for the TagLog Language **46**
3.3 The Language of TagLog **47**
3.3.1 TagLog native syntax **47**
3.3.2 The grammar notation **49**
3.3.3 Condition-action rules **50**
3.4 What is a TagLog Theory About? **50**
3.4.1 The interpretation of terms **51**
3.4.2 The interpretation of formulas **52**
3.4.3 The interpretation of theories **53**
3.5 The TagLog Built-In Predicates **53**
3.5.1 Segment predicates **54**
3.5.2 String predicates **56**
3.6 Reasoning in TagLog **57**
3.6.1 Reasoning **57**
3.6.2 The resolution principle **58**
3.6.3 Proof trees **60**
3.7 Summary and Conclusion **60**

CHAPTER 4 ***TagLog Theories 62***

4.1 Introduction **62**
4.2 Text Descriptions **63**
4.3 Situation Descriptions **68**
4.4 Taxonomies **69**
4.5 Definitions **71**
4.6 Grammars **72**
4.6.1 Logic grammars **72**
4.6.2 A grammar as an axiomatic system **73**
4.6.3 What is a TagLog grammar about? **73**
4.6.4 Grammars and grammar theories **74**
4.6.5 Phrase structure grammar **75**
4.6.6 Categorial grammar **76**
4.6.7 Unification-based grammar **76**
4.7 Lexica **79**
4.8 Syntax, Semantics, and the Modelling of Context **81**
4.8.1 A situation theory as a content/context model **82**
4.8.2 Integrated grammar **82**
4.8.3 Now, what's the point of all this? **85**
4.9 Dialogue Grammars **86**
4.10 Summary and Conclusion **88**

CHAPTER 5 ***TagLog System Overview 89***

5.1 Introduction **89**
5.2 TagLog for Interactive Theory Revision **90**
5.2.1 TagLog is a theory revision system **90**
5.2.2 TagLog is an interactive system **91**
5.2.3 TagLog does this, user does that **92**
5.3 Against Tools! **94**
5.3.1 Cooperation between tools **97**
5.4 Macintosh TagLog User Interface **99**
5.4.1 Menus **99**
5.4.2 Windows **100**
5.5 The TagLog Schema **103**
5.6 The Choice of Interaction Styles **104**
5.6.1 The pragmatics of human-computer interaction **105**
5.6.2 Talking with TagLog tools about texts and theories **106**
5.7 Architecture **108**
5.7.1 The current system **108**
5.7.2 Alternative backbone systems **109**
5.7.3 Client-server architecture **110**
5.8 Efficiency **111**
5.9 Summary and Conclusion **111**

CHAPTER 6 ***Description 113***

6.1 Introduction **113**
6.2 The Importance of Hand Coding **114**
6.3 The Elements of Hand Coding **115**
6.3.1 Observing **115**
6.3.2 Deciding **118**
6.3.3 Framing **118**
6.3.4 Stating **119**
6.3.5 Checking **120**
6.3.6 Storing **121**
6.4 Coding as a Dialog **121**
6.5 Hand Coding in the TagLog System **122**
6.5.1 Ways to speed up coding **122**
6.5.2 Feedback when coding **123**
6.5.3 Undoing statements **123**
6.5.4 Macros **123**
6.6 Summary and Conclusion **124**

CHAPTER 7 ***Exploring Linguistic Reality 125***

7.1 Introduction **125**
7.2 Searching **126**
7.2.1 Searching as deduction **126**
7.2.2 Searching for syntax **128**
7.2.3 Pointing to what is found **129**
7.2.4 Searching as a dialog **130**
7.2.5 Searching and prediction **130**
7.2.6 Precision and recall **130**
7.3 The TagLog Search Tool **131**
7.3.1 Search Tool layout **131**
7.3.2 How to use the Search Tool **132**
7.3.3 A simplified notation **132**
7.3.4 Linking to other tools **133**
7.4 Exemplifying Searching **133**
7.4.1 Searching for instances of a basic string **133**
7.4.2 Searching for adjacency-pair segments **134**
7.4.3 Searching for errors and incompleteness **134**
7.4.4 Tracing the Text-Content/Context relation **135**
7.5 Text in Co-text – Concordancing **136**
7.5.1 The logic of concordancing **136**
7.6 The Concordance Tool **136**
7.6.1 Concordance Tool layout **137**
7.6.2 How to use the Concordance Tool **138**
7.6.3 Linking with other tools **138**
7.7 Concordancing Examples **138**
7.7.1 A concordance of word form tokens **138**
7.7.2 Concordancing based on non-string properties **139**
7.7.3 Using a theory to build a concordance **140**
7.7.4 The Concordance Tool – Statement Browser link **140**
7.7.5 Concordances and stop lists **141**
7.8 Concordancing for Hand Coding **141**
7.9 Concordancing for Grammar Development **142**
7.10 The Treebank Approach **144**
7.11 Summary and Conclusion **145**

CHAPTER 8 ***Browsing Theories 146***

8.1 Introduction **146**
8.2 Retrieval by Deduction **147**
8.3 The TagLog Statement Browser **148**
8.3.1 Inspecting the statements describing a segment **150**

8.4 Parsing as Deduction **150**
8.5 Alternative Theory Presentation Modes **153**
8.5.1 Alternative ways of presenting singular statements **154**
8.5.2 Alternative ways of presenting general statements **155**
8.6 Table-Based Presentation **156**
8.6.1 A simple table in the abstract **156**
8.7 The TagLog Table Tool **157**
8.8 Using the Table Tool **158**
8.8.1 Simple table **158**
8.8.2 Table with list-valued columns **158**
8.9 Summary and Conclusion **159**

CHAPTER 9 ***Quantitative Analysis 160***

9.1 Introduction **160**
9.2 Counting in TagLog **161**
9.2.1 Counting types or counting tokens **161**
9.3 The TagLog Count Tool **162**
9.4 Frequency Tabling **162**
9.4.1 Exporting tables from TagLog to other programs **165**
9.4.2 A word length graph **166**
9.5 Descriptive Statistics in TagLog **166**
9.5.1 Descriptive statistics, an example **168**
9.6 Other Uses of Tables **168**
9.6.1 Listing high frequency words **168**
9.6.2 Collocations **169**
9.7 Summary and Conclusion **170**

CHAPTER 10 ***Logical Extensions 171***

10.1 Introduction **171**
10.2 Hard to Say in Horn Clause Logic **172**
10.3 Explicit Negation **174**
10.4 Meta-Logic **176**
10.4.1 The *provable/1* predicate **177**
10.4.2 The *unprovable/1* predicate **178**
10.4.3 The predicates *observed/1* and *predicted/1* ***179***
10.4.4 The *consistent/1* predicate **180**
10.4.5 The known and the unknown **181**
10.4.6 Meta-logic summary **181**
10.5 Non-Monotonic Reasoning **182**
10.5.1 Describing communicative functions in TagLog **183**

10.6 Extensions Applied **185**
10.7 Summary and Conclusion **186**

CHAPTER 11 ***Automatic Tagging 187***

11.1 Introduction **187**
11.2 The Elements of Automatic Tagging **189**
11.2.1 Requirements **189**
11.3 Statistical Part-of-Speech Tagging **190**
11.4 Transformation-Based Part-of-Speech Tagging **194**
11.5 The Learning of Transformation-Rules **196**
11.6 Logic-Based Part-of-Speech Tagging **199**
11.7 Constraint-Based Part-of-Speech Tagging **202**
11.8 Summary and Conclusion **205**

CHAPTER 12 ***Hypothesis Testing 206***

12.1 Introduction **206**
12.2 The Logic of Hypothesis Testing **207**
12.3 Measuring the Status of a Hypothesis **207**
12.3.1 Coverage **208**
12.3.2 Truthlikeness **208**
12.3.3 Conclusiveness **208**
12.4 The TagLog Hypothesis Testing Tool **209**
12.5 Interpreting the Results of Hypothesis Testing **210**
12.5.1 The significance of predictive instances **210**
12.5.2 The significance of confirming instances **211**
12.5.3 The significance of falsifying instances **211**
12.5.4 The significance of non-covered instances **212**
12.5.5 Summing up significance **213**
12.6 Testing a Simple Hypothesis **213**
12.7 Testing a Grammar Rule **219**
12.8 Summary and Conclusion **221**

CHAPTER 13 ***Explanation 222***

13.1 Introduction **222**
13.2 Some Preliminaries of Explanation **223**
13.3 The Logic of Deductive Explanations **226**
13.4 Explanation and Presentation **228**
13.4.1 Graphical proof tree **229**
13.4.2 Explanation dialogue **230**
13.4.3 Analysis tree **231**

13.4.4 General statement **232**
13.4.5 General grammar rule **233**
13.5 Abductive Explanations **234**
13.6 The Logic of Abductive Explanations **235**
13.7 The TagLog Explanation Tool **237**
13.7.1 The Explanation Tool layout **237**
13.7.2 How to use the Explanation Tool **238**
13.8 Explaining Syntax **238**
13.9 Explanation and Logic-Based Tagging **239**
13.10 Lemmatization as Abduction **241**
13.11 Interpretation as Abduction Revisited **245**
13.12 Knowledge Acquisition as Abduction **246**
13.13 Summary and Conclusion **247**

CHAPTER 14 ***Grammar, Truth, and Context 248***

14.1 Introduction **248**
14.2 The Problem of False Grammars **249**
14.2.1 Grammars revisited **251**
14.3 Errors Tools Make **253**
14.3.1 Parsing and the problem of ambiguity **253**
14.3.2 Searching and the problem of imprecision **257**
14.3.3 Automatic tagging and the problem of tagging errors **259**
14.3.4 Truth and Usefulness **259**
14.4 How to Fix a False Grammar **260**
14.4.1 An integrated approach to context sensitivity **261**
14.4.2 A non-monotonic approach to co-text sensitivity **264**
14.5 Summary and Conclusion **266**

CHAPTER 15 ***Conclusion 267***

15.1 Meet This Linguist! **267**
15.2 What Does He Need? **269**
15.3 How Do We Give It to Him? **270**
15.4 What Does He Get? **271**
15.5 What More Could He Possibly Ask For? **273**
15.5.1 More testing **273**
15.5.2 Better formalism **274**
15.5.3 More speed **275**
15.5.4 More functionality **275**
15.5.5 More clarity **276**

References 278

APPENDIX A *TagLog Built-In Predicates 288*

A.1 Introduction **288**
A.2 Point and Segment Predicates **290**
A.3 String Predicates **294**
A.4 Tree Predicates **297**
A.5 Set and Bag Predicates **299**
A.6 Other Metalevel Predicates **299**
A.7 Arithmetic **301**
A.8 Schema Predicates **302**
A.9 Presentation Procedures **303**
A.10 Corpus Management Procedures **305**

APPENDIX B *Selected Code 306*

B.1 Viterbi Tagging in Prolog **306**
B.2 An Improved Error-Driven Learner **309**

APPENDIX C *An Overview of Corpus Texts Used 312*

C.1 Introduction **312**
C.2 The Brown Corpus Text **312**
C.3 The Doctor-Patient Dialogue Corpus **318**
C.4 The PLUS Corpus Text **323**

List of Figures

FIG.1. A Brown corpus text 32
FIG.2. Some possible relations between two segments 55
FIG.3. SLD resolution proof tree 59
FIG.4. Proof tree 60
FIG.5. A text and its description 65
FIG.6. Text and theory in the TagLog system 67
FIG.7. A situation theory and a situation 69
FIG.8. A taxonomy of word classes 70
FIG.9. A text, a situation theory, and a situation 82
FIG.10. Integrated grammar 84
FIG.11. From text to text with traditional corpus tools 95
FIG.12. TagLog: tool, theory, text 96
FIG.13. Piping between tools 97
FIG.14. Tools making use of uniform representation 97
FIG.15. From text to annotated text to concordance 99
FIG.16. The TagLog system's menu bar and some of its menus 99
FIG.17. The TagLog Text window 100
FIG.18. A TagLog Theory window 101
FIG.19. A Grammar (theory) window 101
FIG.20. A TagLog Situation Theory window 102
FIG.21. The TagLog Dialog window 103
FIG.22. Four ways of building a TagLog system 109
FIG.23. TagLog as a client-server system 110
FIG.24. Hand coding in TagLog 120
FIG.25. The TagLog Search Tool 131
FIG.26. Finding an adjacency pair 134
FIG.27. Tracing the text-content/context relation 135
FIG.28. The Concordance Tool dialog box 137
FIG.29. A concordance of "the" 139
FIG.30. A concordance of nouns 139
FIG.31. The Concordance Tool – Statement Browser link 140
FIG.32. Concordancing for hand coding 142
FIG.33. Traditional retrieval of tags 147
FIG.34. The TagLog Statement Browser 148
FIG.35. The Statement Browser in action 150
FIG.36. Parsing as deduction with the Statement Browser 151
FIG.37. Ambiguity in the eyes of the Statement Browser 152
FIG.38. Mapping theories to other modes of presentation 154
FIG.39. Mapping from clauses to grammar 155
FIG.40. Mapping from clauses to a graphically represented tree 155
FIG.41. The TagLog Table Tool 157
FIG.42. The TagLog Count Tool 162
FIG.43. Tabling in TagLog 165

FIG.44. Diagram produced by TagLog and a graphing program 166
FIG.45. The TagLog Statistics Tool 167
FIG.46. Automatic tagging 189
FIG.47. A Hidden Markov Model 191
FIG.48. Error-driven learning (Adapted from Brill, 1992) 196
FIG.49. The TagLog Hypothesis Testing Tool 209
FIG.50. Testing a simple hypothesis 214
FIG.51. Testing a simple hypothesis (cont.) 215
FIG.52. Testing a simple hypothesis (cont.) 216
FIG.53. Testing a simple hypothesis (cont.) 217
FIG.54. Testing a simple hypothesis (cont.) 217
FIG.55. Testing a simple hypothesis (cont.) 218
FIG.56. Testing a simple hypothesis (cont.) 218
FIG.57. Testing a simple hypothesis (cont.) 219
FIG.58. Grammar-rule testing in TagLog 220
FIG.59. Graphical proof trees 229
FIG.60. Explanation dialogue 230
FIG.61. Explanation dialogue (cont.) 230
FIG.62. Explanation dialogue (cont.) 231
FIG.63. Analysis tree 231
FIG.64. The TagLog Explanation Tool 237
FIG.65. Explanation in TagLog 239
FIG.66. Explanation and automatic tagging 240
FIG.67. Explanation and automatic tagging (cont.) 241
FIG.68. Lemmatization as abduction 243
FIG.69. Lemmatization as abduction (cont.) 244
FIG.70. Lemmatization as abduction (cont.) 244
FIG.71. Interpretation as abduction 246
FIG.72. Knowledge acquisition as abduction 247
FIG.73. Three ways of handling ambiguity 256
FIG.74. A treebank factory 258
FIG.75. An integrated system with a truthlike grammar 261

CHAPTER 1

Introduction

1.1 Introduction

The purpose of this thesis is to build a *corpus theory development environment* – to discuss its design, its use, and its implementation. To do this properly, I shall have to ask a number of important questions: What is a corpus text?[1] What is a corpus theory? What does it mean to develop a corpus theory? But first of all, I shall have to motivate my plans for the existence of yet another 'environment' or 'workbench'. Is this something that the linguist really needs, in addition to (or in replacement of) traditional corpus tools and grammar development environments?

I think so. Because if you are a theoretical (computational) linguist your favourite grammar development environment will probably not be 'sensi-

1. Already at this point, let me state clearly that the term "corpus" in this thesis will be used to denote *written* texts or collections of written texts. Although some of these texts are transcriptions of spoken language dialogues, they are still texts, according to my notion of text. I do not mean to imply that corpus linguistics cannot deal with spoken language directly. On the contrary, the work contained here might even be possible to extend in that direction, but the present system just does not deal with it.

tive' to corpora in any real sense. Or if your are a corpus linguist your tools will probably not let you use current state of the art grammar formalisms to describe and navigate your corpus, which is a pity since notions like unification and inference are really quite handy to have for a corpus linguist too. There is a gulf here between two traditions, a gulf that I would like to see closed.

Most traditional approaches to corpus tools are based on mark-up languages. I am suspicious of mark-up languages. I don't think they are languages. They are notations without formalism, food for brainless text crunching programs. Not as a matter of necessity, but as a matter of fact.

My own approach to corpus linguistics is based on logic, formal reasoning, and an ambition to (re)think scientific methodology in the area of corpus linguistics. On the conceptual/logical level everything will be much clearer, I will argue, if formal logic, reasoning, and scientific method alone are allowed to reign.

Although the purpose of this thesis may not be that 'important' in itself, it is set against a background of really important questions, and it may, I believe, have the power to make us see that background in a different light.

So let me begin there, in this background, with a quote.

> Armchair linguistics does not have a good name in some linguistics circles. A caricature of the armchair linguist is something like this. He sits in a deep soft comfortable armchair, with his eyes closed and his hands clasped behind his head. Once in a while he opens his eyes, sits up abruptly shouting, "Wow, what a neat fact!", grabs his pencil, and writes something down. Then he paces around for a few hours in the excitement of having come still closer to knowing what language is really like. (There isn't anybody exactly like this, but there are some approximations.)
>
> Corpus linguistics does not have a good name in some linguistics circles. A caricature of the corpus linguist is something like this. He has all of the primary facts that he needs, in the form of approximately one zillion running words, and he sees his job as that of deriving secondary facts from his primary facts. At the moment he is busy determining the relative frequencies of the eleven parts of speech as the first word of a sentence versus as the second word of a sentence. (There isn't anybody exactly like this, but there are some approximations.) These two don't speak to each other very often, but when they

> do, the corpus linguist says to the armchair linguist. "Why should I think that what you tell me is true?", and the armchair linguist says to the corpus linguist, "Why should I think that what you tell me is interesting?"
>
> *Charles Fillmore (1992)*

The table below elaborates on Fillmore's caricatures, for the purpose of making a more systematic comparison between the two kinds of linguists. They too are just cartoon figures, obviously, and should not be taken too seriously.

The Corpus Linguist	**The Theoretical Linguist**
Focuses on *Parole/Performance/Process.*	Focuses on *Langue/Competence/System.*
Aims for a *description* of the facts of language and language use as they appear in a corpus.	Aims for an *explanation* of the facts of language and language use.
Is *data-driven* in his approach to linguistic research. A lover of long lists.	Is *theory-driven* in his approach to linguistics.
Is satisifed with simple measures, since they are fast, even for large texts. Thinks that he can live with the noise created by the simplicity of his methods.	Prefers sophistication over simplicity, and accuracy over speed, even if that means that he has to be satisfied with smaller samples.
Prefers *quantitative* methods.	Prefers *qualitative* methods.
Sees himself as part of an *empiricist* tradition.	Sees himself as part of a *rationalist* tradition.
Focuses on the text as a *physical product*.	Focuses on the text as an *abstract entity.*
Mainly interested in grammars of particular languages.	Aims for a Universal Grammar.
Focuses on *form* only.	Focuses on form *and* meaning.
Takes a *global* perspective of texts. Wants an overview of what is there.	Takes a *local* perspective of texts, i.e. zooms in on them, and looks at details.
Focuses on broad (though possibly superficial) coverage of unrestricted text.	Performs deep analyses of (artificially) restricted domains.
Prefers methods based on probability theory and statistics.	Prefers *rule-based* (knowledge-based, inference-based, logic-based) methods.
Relies on observation of (the products of) overt language behaviour.	Relies on intuition/introspection as the main way of accessing data.
Works with authentic data, and with utterances in the context of other utterances.	Works with 'toy examples', in the form of isolated, decontextualized sentences.
Has a Baconian frame of mind. Thinks that the inductive method is the essence of science.	Thinks that the hypothetico-deductive method is the essence of science.
Is a firm believer in Discovery procedures.	Believes in Justification/Evaluation procedures.
If using a computer for other things than word processing, this would be for building concordances, lemmatizing, or counting statistics.	Works with parsers, generators, and theorem provers, if he's computationally minded at all.
If he does any programming at all, it's in Pascal or C.	Prolog or Lisp is what counts, if he's at all interested in programming.

1.1.1 The dichotomy is false …

> My conclusion is that the two kinds of linguists need each other. Or better, that the two kinds of linguists, wherever possible, should exist in the same body.
>
> *Charles Fillmore (ibid.)*

Judging from the literature, most linguists tend to agree with Fillmore on this point. Most linguists seem to think that the dichotomy is false, or at least, that the oppositions are not as strong as they have been made to look here.

First of all, everybody is likely to agree that both theory and data are necessary ingredients of science, that science without theory is most likely to be uninteresting, and that science without respect for data is likely to come up with theories that are false. Interesting and true scientific theories can only spring from the interplay of both theory and empirical data.

As regards the question of what linguistics is really *about*, many linguists simply do not believe that the gulf between Langue and Parole (between Competence and Performance, between System and Process, or whatever pair of terms one prefers) is as large as one is led to think.

> It can [...] be argued that the putative gulf between competence and performance has been overemphasised, and that the affinity between (say) the grammar of a language as a mental construct and the grammar of a language as manifested in performance by the native speaker must be close, since the latter is a product of the former.
>
> *Geoffrey Leech (1992)*

Concerning our ways of accessing data, observation and intuition/introspection are usually thought to complement each other (except, perhaps, by certain extremists in either camp).

> Grammars do not grow magically from corpora. They require the intelligent analytical mind of a grammarian who draws on knowledge of previous studies, on his or her own intuition as well as on observations of text.
>
> *Stig Johansson (1992)*

Concerning method, there is, on the one hand, the belief that statistical analysis ought not to be an end in itself, but only a starting point for further, qualitative analysis of data. On the other hand, there is the idea that statistical treatments presuppose some initial classification.

> There can be no opposition between quantitative and qualitative method, since quantity and quality are mutually complementary rather than exclusive. Indeed, every quantity is either the numerosity of a collection of items sharing a certain quality, or the intensity of a quality. Hence, in the process of concept formation, quality precedes quantity
>
> *Mario Bunge (1995)*

Also, theoreticians argue that quantities can be added to any qualitative model of language anyway, without sacrificing any of the existing features of that model (one could think here of probabilistic logic and probabilistic phrase structure grammar). All in all,

> we may note that matters are not so polarized (if they ever were) as they may appear: many researchers recognize the value of statistically based techniques in the rule-based world, and vice versa.
>
> *John McNaught (1993)*

Concerning descriptive versus theoretical linguists:

> Both types of linguistics are valid in their own terms, and should be regarded as mutually contributory. Descriptive linguistics can be just as answerable to theory as the "theoretical linguistics" of language universals. In fact, descriptive linguistics is more amenable to theory construction and testing in accordance with the tenets of scientific method, because the nature of its data (e.g. utterances in a particular language) is less abstract and more directly observable.
>
> *Geoffrey Leech (1992)*

So to summarize, there are reasons to believe that most corpus linguists and most theoretical linguists realize, or would realize if they thought more deeply about it, that to take an extremist position in relation to this dichotomy would not be very wise.

1.1.2 … but still practised

Having given proper attention to some of those who try to bridge the gap of the dichotomy, it remains to be said that not much work has been done in order to *actually* reconcile the two views. The dichotomy may be seen as false, but it is still practised.

Theoretical linguists usually *do* work with artificial examples, rather than authentic utterances. Fair enough, they are often able to say quite interesting things about their examples, explaining some features of them in

terms of other features, or connecting them to meaning and context in various ways. Unfortunately, the artificial flavour of the examples often makes it less interesting than it would otherwise have been.

Corpus linguists *are* usually preoccupied with the classification, description, and summarizing of data, and the result of this work *can* be rather uninteresting from a theoretical point of view.

Yet the corpus linguist still sits there with his concordance programs and his harddisk filled with multi-megabytes of texts. The theoretical linguist may have traded his armchair for a computer and a couple of parsers, generators, and theorem provers, but he is not using them for analysing text, only for modelling the understanding and production of artificial examples.

To bridge the gap of the dichotomy also in practice, I believe that appropriate tools will have to be developed. This is what I will attempt in this thesis. I will approach this task from the corpus linguist's point of view. Let's begin with a peek in his toolbox.

1.2 The State of the Art of Corpus Tools and Systems

In this section, I will take a brief, critical look at the current state of the art of corpus tools and systems. As my point of departure, I will use a set of informal functional specifications of tools of the kind that corpus linguists often find useful in their daily work on language data. That is, I will try to characterize different types of tools in terms of what they can *do* for the linguist, rather than in terms of what data structures they manipulate, or how they are manipulated. I will then look at some existing tools and systems, and finally, on a generic level, I will give some criticism levelled against these tools and systems.

1.2.1 A functional view of corpus tools

Here is a list of informal specifications of operations relevant to corpus work. Each specification lists the required input as well as the resulting output. Nothing, however, is implied about the representations, algorithms, or heuristics used.

- Hand coding - given a description of a text, a ‘pointer’ to a segment of text, and a ‘predicate’, produces a ‘predication’ which is added to the description.
- Searching - takes a text, perhaps together with a description of it, and a target specification (e.g. a partial description), and produces a set of ‘pointers’ to segments in the text that satisfies the specification.
- Concordancing - takes a text, perhaps a description of the text, and a target specification, and produces a concordance (i.e. a list of words and phrases in context) that satisfies the specification.
- Parsing - takes as input a (segment of) text, a grammar, and a parsing goal, and produces something that satisfies the goal, e.g. a set of assignments of categories to segments, or a set of parse trees.
- Counting - takes a text, perhaps a description of the text, and a target specification, and returns the number of text segments that satisfies the specification.
- Tabling - takes a text, perhaps together with a description of it, and a target specification, and produces a table (e.g. a frequency table, a table of collocations, etc.).
- Collocating - given a description, and a target specification, produces a list of *collocations*, that is, a list of words that co-occur more often than expected by chance.
- Automatic part-of-speech tagging - takes a text, a lexicon (and perhaps some rules, or transition probabilities), and produces a description of the text, at the level of parts-of-speech.
- Lemmatizing - takes a text, a lexicon (and perhaps some kind of rules), and produces a description of the text which specifies the *lemma* for each word token, so as to abstract away from the inflected form of it.
- Interpreting - takes a segment of text, a grammar or some rules of interpretation, and finds the content corresponding to the segment.
- Excerpting - given a text, and a target specification (e.g. a partial description), produces strings corresponding to matching segments.
- Manual disambiguation - given a number of alternative descriptions, and the user’s act of choosing between them, returns one description.
- Automatic disambiguation - given a number of alternative descriptions, and rules for selecting between them, returns one description.

This list does not aim to be exhaustive. Anyone could probably easily think of other functionalities as well. Nor are the operations necessarily primitive: automatic part-of-speech tagging involves automatic disambiguation, concordancing may involve searching, automatic part-of-speech tagging is sometimes regarded as one step in the process of parsing, etc.

Functional specifications are potentially very useful. First, they allow us to *categorize* different existing tools and systems in terms of which functionalities they do, or do not, support. Second, they give us a way to specify the functionality of a new design; the service provided by a corpus 'workbench' could be completely specified by operations such as these. Third, they allow us to *compare* different tools of the same category with respect to how and how well they realize the functionalities. Finally, they give us a handle on an important notion of correctness: given a particular implementation, does it correctly realize the functionalities that it aims for; does it really do what it is supposed to do?

1.2.2 Existing tools for the corpus linguist

A wide range of implementations of tools and systems realizing the behaviour of the operations listed above could be imagined. In this section some more or less well-known examples of such tools will be given.

Brill's rule-based tagger

This is a tool for automatic tagging of parts-of-speech, developed by Eric Brill (1992). It is a rule based tagger where tagging is done in two stages. First, every word is assigned its most likely tag in isolation. For that purpose, a lexicon is consulted. Words not in the lexicon are first assumed to be nouns (proper nouns if capitalised), and then various cues are used to change the guess of the most likely tag. Next, contextually triggered transformation-rules of the kind "if a word is tagged V, and the preceding word is tagged DET, then switch the tag from V to N" are used to improve accuracy. Transformation-rules can be acquired automatically from a training corpus.

The CLAN system

CLAN is a system of tools for analysing spoken language transcripts that are part of CHILDES – a large archive of child language data

(MacWhinney, 1990). These are some of the tools available in the CLAN suite: CHECK - Verifies data accuracy and correct use of coding syntax; COMBO - Searches a file for segments matching a regular expression; DIST - Lists average distance between words and codes; FREQ - Gives a word frequency count; KEYMAP - Creates a concordance; MLU - Calculates the mean length of utterance in a file; WDLEN - Tabulates word and utterance lengths and prints a histogram. Each tool in CLAN is a separate program. Output from one CLAN program can be 'piped' to another CLAN program for further analysis.

The Constraint Grammar system

The *Constraint Grammar system* (Karlsson, 1990; Karlsson et al. 1995) is a rule-based (or constraint-based) system for automatic tagging (and parsing) of running text. Tagging is performed in two stages: first, a morphological module looks up all the possible tags for each word token in a text; secondly, a morphological disambiguator, using a set of carefully hand crafted constraints, discards illegitimate alternatives, based on the context. Tags that 'survive' the constraints are assumed to be the correct ones.

The DagTag system

DagTag is a system for hand coding of dialogues, developed at the University of Linköping (Ahrenberg & Jönsson, 1989). It supports searching and counting. Tagging is performed by assigning DAGs (Directed Acyclic Graphs) to segments of text of arbitrary length, by first marking the text (with the help of a mouse) and then selecting the appropriate tag structure from a menu. The tagged dialogues can be searched for occurrences of specific words or string sequences, or occurrences of special constructions that match a certain DAG.

The Penn treebank

The *Penn treebank* is a 4-5 million word corpus that has been annotated for part-of-speech information and 'skeletal' syntactic structure (Marcus, Santorini, & Marcinkiewicz, 1993) and predicate-argument structure (Marcus et al.,1994). Various tools for manipulating and navigating the corpus have also been developed. The *PARTS* part-of-speech tagger (Church, 1988) was used to automatically tag the text. The *Fidditch* partial parser (Hindle, 1983) was used to parse the corpus automatically.

Tools for ‘tag-disambiguation’ and correction of part-of-speech tagged text, as well as tools for correcting analysis trees (bracketed structures) were developed in GNU-Emacs (Santorini, undated). *Tgrep* is an analysis tool for pattern matching against the skeletally parsed corpus.

The Stuttgart Corpus Workbench

The Corpus Query Processor (CQP) is the heart of a corpus management system developed at the university of Stuttgart (Christ, 1994). (Other tools of the system include tools for parsing, collocation extraction, clustering, and statistics-based tools for automatic parts-of-speech tagging.) CQP is a command language based query interpreter. Queries are entered by the user, the result is computed and presented to the user in a simple concordance format.

Queries can be seen as a set of constraints which must be fulfilled by a sequence of corpus ‘entities’, usually words. These constraints can be expressed by several means: the central construct is regular expressions over ‘attribute expressions’, which are in turn boolean expressions.

The Tagger system

Tagger (Meyer & Tenney, 1993) is a system that supports hand coding, counting (statistics), and excerpting. The Tagger system is designed to be “useful to the corpus linguist who wishes to study a particular linguistic construction in detail and to customise the tags that he or she develops to study the constructions.” In brief, Tagger is a program that allows a linguist to mark constructions in a corpus, create a series of tags that could be associated with the marked constructions, conduct statistical analyses of the tags that were created, and print out examples of constructions.

The Xerox Part-of-Speech Tagger (XPOST)

XPOST is a hidden Markov model based part-of-speech tagger. Given a sentence, each word token is assigned a part-of-speech ambiguity class from a lexicon (e.g. “package” is in the ambiguity class {noun,verb}). Words not in the lexicon are subjected to suffix analysis. A probabilistic model that assesses the likelihood of particular part-of-speech assignments based on word order is then applied to disambiguate the available choices. The final output is a sentence with each word tagged with the most likely part-of-speech tag. (This paragraph is quoted in its entirety from DFKI NL Software Registry.)

The Xtract tool kit

Xtract is a set of (mainly) statistics-based tools for the automatic extraction of collocations from text corpora (Smadja, 1993). There is a tool for finding sentences containing given collocations, tools for the extraction of all n-grams from a text satisfying certain conditions specified by the user, tools for presentation of output, a frequency tabling tool, etc. Xtract can make good use of part-of-speech tags in the input texts, if they are given in the right format.

Sinclair's Strategy

This is not really an existing system (at least as far as I know). Rather, it is a list of tools believed by a distinguished corpus linguist to be needed by the corpus researcher: a parser, a boundary marker, a tagger, a collocator, a lemmatizer, a compounder, a disambiguator, a lexical parser, a phrase finder, an exemplifier, a setter, a classifier, a typologizer, etc. (Sinclair, 1992).

Although I cannot say I understand the purpose of every one of these tools, the idea seems to be to isolate the smallest possible chunks of functionality and to implement these as separate tools. Then complex routines "can be built up for particular applications by organizing several tools together" (Sinclair, ibid., p. 396).

In the same paper, Sinclair also presents what he claims are "guidelines for software design that seem to be appropriate to the nineties in language text processing" (Sinclair, ibid., p. 396):

- Analysis should be restricted to what the machine can do without human checking, or intervention.
- Analysis should be done in real time.
- Operations should be designed to cope with unlimited quantities of text material.
- Software will be designed to operate at more than one level of discrimination, so as to bypass doubtful decisions.
- Speed of processing should take precedence over ultimate precision of analysis.
- Software should be robust.

1.2.3 Existing corpus tools – a critical discussion

Typically, existing corpus systems implement only some of the functionality required by the corpus linguist. A corpus linguist hoping to find *one* single system fully implementing *all* of the functionality listed in Section 1.2.1 will be disappointed. (In fact, some functions are not supported by *any* traditional corpus system that I know of. For example, I have not been able to find corpus systems that relate any (interesting) representation of the content of a text to its form, despite the fact that many linguists see this as important.)

A problem with current *tools* is that often they do not fully realize their inherent functional potential. Their functionality does not generalize the way one expects it to. There may be restrictions on the length of segments that can be handled – perhaps only word-long segments can be marked, searched for, or counted. There may be restrictions on the level of descriptions that can be handled – perhaps only one level at a time. There may be restrictions on the relation between tags – perhaps only one tag per segment is allowed, perhaps overlapping tags are forbidden. Or there may be a gross incompleteness of retrieval tools with respect to the information actually encoded in marked up texts – perhaps the query "give me all segments that consist of a determiner which is not followed by a noun", is a query the system cannot handle.

Many existing systems suffer from what I will call "a lack of connectedness": a bad, inflexible integration of tools, that does not let the user exploit already existing knowledge sources, or combine functionality in novel, as of yet unthought of ways. Typically, knowledge used by one tool cannot be used by another tool, not even when the knowledge is relevant for the other tool as well. Furthermore, output from one tool can often not be taken as input to another, not even when this would *in principle* be possible.

Some systems lack full relevance for researchers coming from a tradition different from the one in which the system was developed. A linguist writing unification-based grammars, for example, who wants to take advantage of existing corpora and corpus tools in his own research, might find the discrepancy hard to handle, between, on the one hand, his own systems of concepts and terminology, his own goals and methods, and, on the other hand, the concepts, goals, and methods of the tradition from

which he borrows his corpus tools. The tools might not let him structure and organize the corpus in the way that he would like, and he will have to learn to continuously 'translate' between two modes of thinking: he will have to learn to translate questions concerning correctness and coverage of his grammar rules into queries that the system can handle, and he will have to learn to turn the answers that he gets into actions modifying or extending the grammar.

Systems that are interactive in any real sense ought to support incrementality, and reversibility, and they ought to supply immediate feedback (cf. Shneiderman, 1992). But many traditional corpus tools are 'all at once' systems, that do not let us add or delete information incrementally, or if they do allow that, they are not 'reversible', so that if an algorithm has worked on a text, and we remove some of the information that it used, the effects are still not undone. Systems are often built to work in a batch oriented, non-interactive, non-cooperative, style. So sometimes, even when the user feels that he knows something that would be of help to the system, he is not able to share this knowledge with the system, to cooperate with the system. Moreover, in those cases where a system is interactive, it is so in a rigid way, with the system in control, and the user at its mercy as an 'oracle', supposed to answer the questions the system asks. Systems are seldom cooperative, in any real sense.

Many systems have inadequate usability. For one thing, they often lack a graphical user interface (GUI), even when they *do* allow some interaction. But I am not so much talking about a lack of GUIs. Given the current state of the art of user interface technology, any program can be provided with a GUI. What I am after is rather the fact that many existing tools and systems lack a clear and consistent conceptual model – the logic of their functionality is unclear. The often cited goal of a successful interactive system – having the computer vanish as users become completely absorbed in their task domain – is therefore seldom achieved.

To sum up this section so far, I want to argue that current corpus tools and systems are lacking in:

- multi-functionality
- generality
- connectedness

- full relevance
- interactiveness
- usability

But these are just symptoms; I would like to offer the following diagnosis. I do not think that the problem is bad engineering, or bad choice of data structures or algorithms. The problems go deeper than that. I believe that the most important source of problems with the state of the art of corpus tools and systems is a lack of appreciation of the importance of adequate representation formalisms, query languages, and explicit reasoning strategies.

Let us investigate, albeit at a tentative level, the representational strategy underlying the tools and systems mentioned above. Most of the systems are based on the general idea of *mark-up*, i.e. labels are attached to segments of text, describing their properties, or relations between them. I will refer to this idea as 'Tagging by Labelling'.[2]

In a particularly simple version of the idea, each word is followed immediately by a label. An occurrence of "John loves Mary" marked up in this way would look roughly as follows:

(1) John/pn loves/v Mary/pn

Different versions of the general idea exist, depending on the spatial orientation of the text and how the labels are attached. Each word can be placed on a separate line, followed by its tag:

(2)
```
John   pn
loves  v
Mary   pn
```

Or labels may be located below the words that they attach to:

(3)
```
John   loves   Mary
pn     v       pn
```

Regardless of version, these simple 'mark-up languages' can be enhanced with a simple mechanism for extending the referential 'scope' of a label.

---

2. It may be interesting to note that according to the Frequently Asked Questions document (FAQ) of the newsgroup 'comp.text.sgml', "mark-up" is a term coming from the publishing and printing business, where it means the instructions for the typesetter that were written on a typescript or manuscript copy by an editor.

In the case of the linear format, a pair of brackets will do. One may thus attach one label to a sequence consisting of more than one word, in the following manner:

(4) [s John loves Mary]

By allowing *embedding* of one such scoped tag within another, arbitrarily deep tree structures can be created.

(5) [s[np John/pn][vp[loves/v [np Mary/pn]]]

Up until now, I have referred to a system of mark-up labels as a 'language'. It should be noted, however, that the status of mark-up languages as languages is not completely clear. As far as I know, no comprehensive effort has been made to understand exactly what their status is.

If they really *are* languages, it should be possible to say something about their syntax, their semantics, their expressive power, and how we may reason by means of them. But many things are unclear with respect to these properties, even on a very basic level. Consider the example in (3) again, given here as (6):

(6)

John	loves	Mary
pn	v	pn

Does this mean that these particular *tokens* of "John", "loves" and "Mary" are classified as a proper noun, verb, and proper noun, respectively? Or does it mean that the word *types* represented by these strings are classified in this way? This is not clear.

Should the expressions in (6) also be read as claims about the particular *order* of items in the text? Do they claim that each of the word-long segments is a proper noun, a verb, and a proper noun, respectively, *and* that the verb follows the first proper noun and that the second proper noun follows the verb? One can certainly imagine an application that is capable of finding instances of verbs followed by instances of proper nouns, on the basis of descriptions like the one in (6). But do the expressions *explicitly* claim that the followed-by relation holds between these two segments, so that the application can derive this *in* the language, or is this something that the application would have to 'read off' the expressions? This is not clear.

One could still argue that the 'language' is so simple that there is no *need* for an explicit semantics or theory of inference. If we can keep the lan-

guage as simple as it appears in (6), this would perhaps be true. But as soon as we start to ‘enhance’ it in order to get some more expressive power, we get entangled in semantic problems.

Suppose we want to extend the expressive power of our language by allowing the attachment of more than one label to each word, as in (7), for example.

(7)

John	saw	Mary
pn	v,n	pn

What does this mean? Does it mean that this particular token of “saw” is classified as a noun *and* a verb? Or should we understand it disjunctively, as noun *or* verb? And what ‘conclusions’ is a system using this information entitled to draw? If we tried to search for all instances of nouns, would this token of “saw” be found then?

Or suppose we leave out a label, as in the following:

(8)

John	loves	Mary
pn		pn

Is this supposed to mean that the word in question does not belong to *any* part-of-speech? Or is it supposed to mean that we don’t know what part-of-speech it belongs to? Or is it a sign of something forgotten? Or is it supposed to express a set of negative statements: that the word is *not* a verb, *not* a noun, *not* an adjective, etc.

I am not saying that such semantic issues cannot be fixed for a particular mark-up language. I am just saying that usually they *are* not fixed, or if they are fixed, they are fixed in a different ways for different mark-up languages, or worse, in different ways for different applications using the same language, so that the ‘semantics’ of the language is identified in practice with what a particular application can extract from a tagged corpus described by means of it.

Furthermore, as regards expressive power, I am not saying that the expressive power of such a language is too weak. I am saying that I do not know what the expressive power *is*. In fact, expressive power of a language is a meaningless notion if we do not know what the expressions in the language mean. However, before we can say anything about the meaning of expressions, we ought to be clear about their form. Though, not even *that* is clear. Are we supposed to see the labels in a marked up text such as (6) as expressions describing the words above them? Or are

we supposed to see the *whole* of (6), *including* the strings “John”, “loves”, and Mary”, as one expression describing the original text, “John loves Mary”? This is not clear.

Speaking of weak expressive power, one must not forget that this is not necessarily a bad thing, as long as there is *enough* expressive power for the purpose at hand. But *if* expressive power is too weak, ‘tagging problems’ are bound to arise, that can only be solved by ‘hacking’ solutions. Everybody will then start to hack their solutions in different ways. (Or a standard is created, to make people hack their solutions in the same way.)

One area where hacking might be called for is where formal grammar theory meets traditional mark-up languages. Traditional mark-up languages are hard to reconcile with state of the art grammar theory. The labels of traditional mark-up languages are usually atomic, i.e. they lack internal structure. By contrast, the ‘labels’ in formalisms used by theoretical linguistics – formalisms such as PATR and DCG – are structured objects. In a feature-structure based formalism, the structure of a ‘label’ is defined by its *attributes* and their *values*. The value of an attribute may be any element, either atomic or complex. In a term-based formalism, the structure of a ‘label’ is determined by the predicate symbol, the arguments, and the order of the arguments. In either case, the structure can be used to build an hierarchy of ‘labels’ based on the subsumption relation, it can be used to express agreement restrictions, etc. This cannot be done in a traditional mark-up language.

Traditional mark-up languages have no means of expressing generalizations. Thus there are no ways to write grammars in these languages; there are no ways to explicitly account for the relation between a corpus and a lexicon; there are no ways to express generalizations to submit to a hypothesis testing procedure; and there are no ways to express the result of a learning procedure.

Traditional approaches to tools for corpus linguistics lack well-defined query languages. This is a source of incompleteness: information may be there, implicitly encoded in the pattern of labels, or encoded in some more active data structure, but if the query languages lack the necessary expressive power, we may not be able to retrieve it.

In the above, I have dealt with the lack of appreciation of the importance of representation formalisms, query formalisms, and explicit reasoning

strategies. Another (albeit related) problem with the state of the art of corpus tools and systems is a lack of concern for matters of foundation – a tendency for philosophical naivety or lack of interest. Important distinctions are neglected. Important notions are not defined. The logic of important knowledge structures is not analysed. Ontological or epistemological questions are seldom discussed. In the long run, I believe that a lack of discussion of 'first principles' is bound to affect corpus linguistics – and its tools – in a negative way.

Yet, another (and again related) problem is that the *purpose* of some of these corpus systems is somewhat unclear. Technology aims at practical ends, pure science aims at understanding, and in between there are all sorts of grey areas: technology more or less principled, science more or less applied. So what about these tools? Are they supposed to form modules in NLP systems – robust, error-free and efficient black-box components from which industrial applications can be built? Are they supposed to support acquisition of knowledge for such NLP system modules – to overcome the 'knowledge acquisition bottleneck' of that particular system? Or are they supposed to be tools for the scientific investigation of text – research instruments and workbenches to support development of theories about texts?

These are three different purposes that such systems can have. It seems to me that many designers of traditional tools try to remain neutral with respect to these aims, with the idea that *their* system may serve all three purposes at the same time. However, it is easy to see that the measure of success is different depending on the purpose. An NLP software module might be regarded as successful if it is modular, efficient, robust, well engineered, etc., and if it works correctly for input within its intended range of application. It does not necessarily have to work with explicit representations that are meaningful to the human eye, it doesn't have to be able to explain its behaviour to the user, and it doesn't have to work for unexpected input. A knowledge acquisition tool or method for a particular target application, on the other hand, is regarded as successful if it can help to provide the NLP application with the rules, representations, databases, or data structures, that will make it work as intended. However, if corpus linguistics is a science, it ought to set its goals higher than this. It is here, I think, that traditional measures of success of scientific theories comes in: truth, interestingness, coverage, simplicity, and so on.

1.3 Related Tools and Systems

In this section, I present two kinds of systems that are more or less related to corpus systems. At least they are the kind of systems that have inspired the design of the corpus system proposed in this thesis. I am thinking here of *grammar development environments* and *knowledge acquisition systems*.

1.3.1 Grammar development environments

A grammar development system (or 'environment') is a computational tool designed to assist the linguist in the development of grammars for natural languages. Usually, at least the following functionality is provided:

- Parsing - takes a sentence, a grammar, and a parsing goal, and produces something that satisfies the goal, i.e. usually a set of parse trees, logical forms, etc., that reflects syntactic and semantic properties of the sentence.
- Generation - takes a parse tree, and/or a logical form, a grammar, and a generation goal, and produces a string of words that satisfies the goal.

Typical systems also provide facilities for maintaining grammars (modifying, adding, and deleting grammar rules, checking well-formedness and consistency, etc.) and debugging grammars (monitoring or interacting with a parser, analysing failure to parse, etc.). Here, I will give just one representative example of such a grammar development environment: the *Pleuk* system.[3]

The Pleuk system

Pleuk (Calder & Humphreys, 1994) represents "an attempt to provide a 'shell' within which to situate implementations of a variety of grammar

[3] But let me also, since I am one of the authors, briefly mention *Linguistic Instruments' Grammar Laboratories*. First and foremost, this is a set of easy-to-use grammar development tools for *educational* purposes, but can also be used as lightweight research tools. Each tool contains a parser and a generator, facilities for printing analysis trees, etc. Currently there exist versions for phrase structure grammar, definite clause grammar, PATR grammar, and categorial grammar.

formalisms. The system offers a way of describing the various modules of a grammar formalism and defining operations over files and the objects described in them, such as compilation, editing and display." (ibid.). The latest version of Pleuk has some facilities for testing against a 'corpus' of sentences represented as lists of words annotated with descriptions of what results should be found. During testing of a grammar against such a corpus, Pleuk is able to generate a report of which analyses match or fail to match their expected results.

1.3.2 Knowledge acquisition systems

A knowledge acquisition system is a system designed to assist a 'knowledge engineer' in the development of theories (or 'models') of some application domain.

Here is a list of functionalities that a knowledge acquisition system might be able to provide:

- Generalization - given a description of a certain subset of a domain, generates a description that characterizes (or refers to) a larger part of the domain.
- Specialization - given a description of a certain subset of a domain, generates a description that characterizes (or refers to) a smaller part of the domain.
- Explanation - given a statement, a description of data, and a background theory, produces an explanation of the statement.
- Prediction - derives consequences of given knowledge.
- Hypothesis testing - given a hypothesis, a description of data, and a background theory, produces some measure of the empirical support for the hypothesis.
- Consistency checking - given a theory, checks its logical consistency.

I am particularly interested in the kinds of systems that are categorized as *theory revision systems*. The problem that a theory revision system is designed to solve is to revise an imperfect domain theory – obtained from a textbook, from interviews with an expert, from the users own intuitions about the domain, etc. – to make it consistent with empirical data.

Theory revision systems might be further classified as *interactive* systems or *balanced cooperative* systems (Morik, 1994). In an interactive theory

revision system, the user is involved, but only as an ‘oracle’; he adds knowledge to the process when the systems ask him for it, but the system is always in control. *Clint* (DeRaedt, 1992) is an example of such a system.[4] In a balanced cooperative theory revision system, either the user or the system can perform a task on the evolving theory. The control is flexible, and the user is free to manually add or delete a fact or a rule at any time, and the system uses it. Systems in this category include MOBAL (Morik et al., 1992).

Here I will only briefly describe one theory revision system – the balanced cooperative system MOBAL.

MOBAL

MOBAL (Morik et al., ibid.) is an integrated knowledge acquisition and machine learning system. By using MOBAL’s knowledge acquisition environment, a user can incrementally develop a model of an application domain in terms of logical facts and rules. The knowledge entered can be viewed and inspected in text or graphics windows, augmented, or changed at any time. The built in inference-engine can immediately execute the rules that have been entered to show the consequences of existing inputs, or answer queries about the current knowledge. Machine learning methods can be used to automatically discover rules based on the facts that have been entered, or to form new concepts. If there are contradictions in the knowledge base due to incorrect rules or facts, there is a knowledge revision tool to help locate the problem and fix it by recording exceptions or modifying rules (cf. Morik et al., ibid.).

1.4 The Aims of This Thesis

Even if the corpus linguist and the theoretical linguist are reconciled on a philosophical and ‘political’ level – and I think they are – it still remains to set up goals, and to develop tools and methods, that overcome the dichotomy on the practical level. This is what I will attempt in this thesis.

4. Note that the term “interactive” is used in a more narrow sense here. In this thesis, I will use the wider sense of this term, unless otherwise stated.

At the centre of our attention stands the design, use, and implementation of a system for the description and analysis of corpora, more powerful and versatile and flexible than tools already existing. I have called this system *TagLog*.[5]

The title of the thesis reveals what I believe is the key to success in this enterprise: a logical approach to corpus linguistics. This approach can be summarized as follows: the use of sentences of logic to express statements about texts; the use of inference to manipulate these sentences in order to analyse the texts. I intend to investigate the possibilities inherent in the approach, as well as some of its consequences.

I will try to show that logic may indeed play a significant role in the design of corpus tools and systems. A logical language may be used to 'logically reconstruct' traditional corpus tools and methods. A logical reconstruction of a particular tool or method is simply a logical theory that together with a certain reasoning strategy is able to explicitly perform the same reasoning as the tool is doing implicitly, and thus, computationally interpreted, realize the same functionality. To take one simple example, parsing as deduction is a logical reconstruction of parsing, in exactly this sense.

What will be demonstrated is the remarkable ease with which the functionalities needed in a corpus system are implemented when based upon adequate means of representing, querying, and reasoning. In fact, the proposed system implements *all* the functionality listed in Section 1.2.1 (hand coding, searching, concordancing, parsing, counting, tabling, collocating, automatic part-of-speech tagging, lemmatizing, excerpting, disambiguating, interpreting) *plus* some of the functionality provided by grammar development systems (generation and presentation of analysis trees, parse failure analysis, etc.), *plus* a a great deal of the functionality provided by theory revision tools (hypothesis testing, explanation, and various kinds of learning). Thus, the need for *multi-functionality* is well catered for.

---

5. *TagLog* stands for *Tagging in Logic* just like *Prolog* stands for *Programming in Logic*. This name was given to it long ago, and it has stuck. But since I no longer regard the TagLog system as 'yet another tagging system', but prefer to see it as a corpus theory development environment, this name is no longer very fitting.

Furthermore, I will argue that an increase in *generality* is something that follows naturally, if we concentrate on reconstructing the functionalities needed using a sufficiently expressive logic. Also, I will try to show that *connectedness* can be a matter of using a uniform underlying representation format and that, to a large extent, composing functionalities can be a matter of *logically* interfacing different 'sub-theories', or a matter of composing expressions in the query language. Finally, I will argue that in the TagLog system too, *interactiveness* is best supported by a well-designed conceptual model and by the incrementality allowed by a closed-loop system,[6] the reversibility allowed by certain truth maintenance methods, and the feedback allowed by the conversational view of human-computer interaction.

Moreover, the proposal made in this thesis should not only be regarded as a proposal for yet another corpus system – a contender of the systems described in Section 1.2.2. I believe it *is* a better way of making corpus systems, but at the same time, it can be described as a way of *avoiding* making corpus systems, and as a way of liberating corpus linguistics from what I believe is a rather unhealthy reliance on tools, and make it focus instead on *theories*. So when Sinclair (1992) claims that it is "necessary to mount a regular critique of any analytic software, to explore its limitation, check its result and search for improvements.", I think that's fine, except that the term "analytic software" should be replaced with the term "theory". This is my war cry: against tools!

I will be very explicit about the goal of the approach to corpus linguistics advocated in this work. I would like to rekindle a very old idea of what science is all about, and relate this idea to corpus linguistics. According to this view, science is a quest for true theories of the world. I think this view is still valid (despite the frequent instrumentalist attacks against it), and that it is valid for corpus linguistics too, and consequently, I would like to regard corpus linguistics as the business of *developing true theories about texts*. In fact, I would like to elaborate on this a bit, and regard the purpose of corpus linguistics as the *development of true, wide-coverage, general, explicit theories about syntactic, semantic and pragmatic aspects of authentic texts.*

---

6. A closed-loop learning system is a learning system that is able to take the knowledge learned as an input to another learning process (cf. Michalski, 1994).

Thus, as far as a methodology is proposed in this thesis, it will be a methodology for the development of such theories. As far as tools are designed and implemented, they will be tools to support this methodology, to store and maintain general and explicit corpus theories, to allow testing of their truth and coverage, etc. In fact, the focus on development of explicitly represented bodies of knowledge, suggests that we regard the proposed system as a *corpus theory development environment* (analogous to a grammar development environment), or as a theory revision system tailored to the particular domain of text.

1.5 What This Thesis Contains

In this section, I provide an overview of the content and structure of this thesis. First, a chapter-by-chapter overview is given, then some of the themes that cross the chapter boundaries are presented.

1.5.1 Chapter by chapter

This chapter has set the scene for the whole of this work. The rest of the chapters are devoted to the articulation of the logic-based approach to corpus linguistics, and to the design and use of the TagLog system.

Chapter 2 presents and discusses, under the heading of ‘Philosophy of Corpus Linguistics’, some more or less philosophical assumptions concerning the nature of scientific theories, about reality – linguistic reality in particular – and about the relation between theories and reality. Also, a picture of the process of developing corpus linguistic theories will be sketched, inspired by philosophy of science, and in particular by the work of Karl Popper.

In Chapter 3, a notion of *formal theory* is developed, that will allow us to some extent capture the informal view of a theory explained in Chapter 2. Some important requirements for a language for description and analysis of text are proposed. Syntax and semantics for the formalism of choice – Horn clause logic – is given, and the only rule of derivation that is really needed – the resolution principle – is explained.

In Chapter 4 we explore the expressive power of the TagLog formalism, and use this power to develop various example theories about different aspects of text: data descriptions, grammars, and lexica, for example.

In Chapter 5, a presentation of a Macintosh implementation of the TagLog system is given. This is just an overview, however, and more detailed descriptions of the TagLog analysis tools are given throughout the thesis, in connection with the sections that describe the theory behind the tools, and give examples of their use.

Chapter 6 contains a discussion of the process of *description*, something that in the TagLog framework is regarded as the process of *updating* a theory. This chapter is concerned mainly with manual description, or *hand coding*. (Automatic tagging will be the topic of Chapter 11.) The connections between the notion of hand coding and the notions of *observation* and *pointing* are accounted for.

Chapter 7 covers two important aspects of *data-exploration*, namely *navigation* and *presentation*. It is shown that the TagLog system, through an implementation of the notion of *searching as deduction*, and by means of a built-in predicate that implements a kind of *deixis*, is able to find segments satisfying partial descriptions provided by the user and subsequently point them out to him. It is also demonstrated how the TagLog concordance tool works. Finally, it is shown how navigation and presentation in this sense is linked to observation and description in the sense of Chapter 6.

In Chapter 8, the browsing of theories is discussed. Among other things, the notion of *parsing as deduction* is explained. The TagLog Statement Browser is presented – a tool designed to support inspection and manipulation of the current description of a particular segment of text. Whereas the Statement Browser always presents descriptions in the form of sets of logical formulas, it is also argued that a logic-based system for corpus analysis is not confined to looking at the result of analyses in the form of sets of logical formulas. Often, systematic mappings between TagLog formulas and ‘visual’ forms that convey the same information exist.

Chapter 9 is devoted to the quantitative summary of data: counting, descriptive statistics and frequency tabling. The TagLog Table Tool is used to generate listings of high frequency words and collocations.

In Chapter 10, by the addition of a restricted form of classical negation, and some meta-logical capabilities, the expressive power of the logic presented in Chapter 3 is significantly raised with very little loss in terms of efficiency. The main motivation for the proposed extension is the perceived need for dealing with *negative*, *incomplete* and *inconsistent* information.

In Chapter 11, the focus is set on methods for automatic tagging. Some traditional methods are reviewed, and logical 'reconstructions' of some of them are proposed – reconstructions that shed light on how these methods work and how they can be generalized.

In Chapter 12, the TagLog Hypothesis Testing Tool is presented – a tool designed to provide the researcher with a measure of the degree of success with which a general hypothesis describe the relevant facts of a text. The significance of *confirming*, *falsifying*, *predictive* and *non-covered* instances of a hypothesis expressed as a universal statement is explained, and related to the notions of *checking*, *confirmation*, *disconfirmation*, and *modification* of hypotheses.

In Chapter 13, an explication of a very important concept from the philosophy of science, namely *explanation*, will be made. Sentences expressing universal statements give rise to the need to be able to explain conclusions, in terms of the application of such sentences. I deal with straightforward (deductive) explanations as well as with abductive (or 'hypothetico-deductive') explanations. I make an attempt to separate the issue of the semantic content of explanations, from their 'syntax', i.e. how this content is actually presented (as a proof tree in one of its many variations, as an ordered set of rule instantiations, or as a general rule).

In Chapter 14, I will show that given the particular view of what grammar rules mean that is favoured in this thesis, many rules in 'traditional' grammars are strictly speaking false, so that they therefore make tools that depend on them less useful than they would have been if the rules were true. In particular, notions like 'ambiguity' and 'imprecision' are related to the truth values of grammar rules. Various ways of 'correcting' grammars are discussed.

In Chapter 15 the logical approach to corpus linguistics and its realization in the TagLog system is assessed with respect to the overall picture given in the first chapter. In what sense can they be said to have bridged the gap

of the dichotomy presented in the introduction? More generally, in what sense can the advancement of our understanding of language and language use benefit from a logic-based approach, and a system like TagLog?

Finally, the thesis contains three appendices: a listing of (a selected number of) TagLog built-in predicates, a selection of various Prolog programs discussed in the thesis, and an overview of the different corpora used to exemplify the logical approach to computational corpus linguistics.

1.5.2 Some recurring themes

Some recurring themes in this thesis, partly reflected in the layout of the individual chapters, are:

- *Philosophy.* In sections of this kind, I present assumptions concerning the subject matter and methods of corpus linguistics. The assumptions are more or less philosophical in nature – they concern ontology, epistemology, and philosophy of science.
- *Technical matters.* In these sections, I will specify, often in logical and set-theoretical terms, the basic ideas and techniques underlying the design of each of the TagLog tools, and the functionalities that they provide. The sections have names like ‘Tagging with a Theory’, ‘Searching as Deduction’, ‘Parsing as Deduction’, ‘Lemmatizing as Abduction’, etc. Often, the logical way of doing things is compared with more traditional techniques.
- *Manual extracts.* These sections can be thought of as extracts from the TagLog user manual. For each tool, they explain the roles of the buttons, text entry areas, pop-up menus, etc. that populate it. Typically, a picture is given that shows the layout of the tool. These sections also explain, step by step, how to use the tools. The idea is to exemplify, and to make more concrete, the material presented in the technical sections.
- *Examples.* These sections explain how to solve *particular* problems in TagLog, either representation problems (how can I describe the linguistic phenomenon X in TagLog?; how can I capture the relation of X to Y?) or analysis problems (how can I write a formula that will allow me to find out X?). They do so by presenting ‘abstract’ solutions in the form of expressions in the TagLog formalism, together with natural-language paraphrases that explain what the expressions mean, and

sometimes they even give concrete instructions for how to use the expressions in the environment of the appropriate tool (described in a manual section) in order to solve a problem. In the latter case, a picture of the full screen is often given that shows the tool in action.

- *Discussions.* In these sections, points are brought up to discussion that summarize, conclude, try to make a point, link different ideas, assess, compare with traditional tools, relate to the background, etc.

1.6 A Short Note about Corpora

Before closing this chapter, let me briefly introduce some of the corpus texts that I will use in this work. I will use one of the texts from the now classic *Brown corpus* of written American English (Francis & Kučera, 1982), I will use a text-based dialogue between a 'user' and a 'system' that was recorded in a so called 'Wizard of Oz' setup in the ESPRIT project PLUS (Nivre, 1992), and I will use a doctor-patient dialogue transcribed by Kós-Dienes (forthcoming) in the course of her thesis work. A more detailed presentation of these texts can be found in Appendix C.

It must also be said that I myself will not use these texts in the manner that I think a corpus linguist should use such texts: to develop better theories about texts. Rather, I will use them to discuss my ideas about what corpus linguistics is or ought to be, and to demonstrate the capabilities of my system. I am not aiming at saying anything particularly interesting about the texts as such, and should that happen anyway, it will be by accident.

1.7 Summary and Conclusion

Let me end this introduction by summarizing the foci of this thesis. What I am aiming for is a corpus theory development environment, a system that scores high in terms of:

- multi-functionality
- generality

- connectedness
- full relevance
- interactiveness
- usability

Such an environment can be built, I think, by designing a representation strategy which is

- expressive enough for the purpose
- declarative
- computationally tractable

and by embedding the whole concept in a convincingly told philosophical and methodological story, including an ontology of text, an analysis of the notion of theory, an explication of the notion of truth, and other foundational issues.

CHAPTER 2

Philosophy of Corpus Linguistics

2.1 Introduction

The aim of this chapter is to present and discuss some (more or less) philosophical assumptions about the nature of scientific theories, about reality – in particular linguistic reality – and about the relation between theories and reality. Also, a picture of the process of developing corpus linguistic theories will be sketched, inspired by philosophy of science,[1] and in particular by the work of Karl Popper.

One does not commonly find a chapter on philosophy and philosophy of science in a text about corpus linguistics. But I have several good reasons for including such a chapter in this thesis.

1. I take philosophy of science to be the branch of philosophy which tries to analyse, define and explicate concepts like theory, explanation, prediction, falsification, verification, law, descriptive generalization, and other important concepts relevant to science. Some philosophers of science deal specifically with particular sciences such as physics and psychology, and some (but fewer) with the application of philosophy of science to linguistics (Botha, 1989; Carr, 1990; Dyvik, 1992; Itkonen, 1978; Yngve, 1986).

First, whether the view of corpus linguistics as the business of developing true theories about text is a correct and fruitful view, does, of course, depend on what notions of theory, text, truth, and theory development, we employ. If *truth* is a matter of the statements of a theory fitting the facts of linguistic reality, then the notions of 'theory', 'statement', 'linguistic reality', and 'fact', are indeed important.

Secondly, from a slightly more narrow point of view, the logical approach to corpus linguistics, and the tool designed to support this approach, involves a logical language in which to form descriptions of texts. For such a language to be *meaningful*, its symbols must be associated with elements of text, in a clear and systematic way. Hence we need to know: what are the elements of text?

Thirdly, the designer of a computer system – *any* computer system – ought to be guided by a clear view of the entities that are to be manipulated by the system. According to (Shneiderman, 1992) the secret behind a successful interactive system is to come up with an appropriate representation of reality. Only then will the designer be able to create a system that follows a consistent, coherent conceptualization. Also, the users will form some kind of conceptual model of the system and its workings – users always do – and it is the duty of the designer to help the users form a good such model. In fact, corpus tools have been criticized for lack of conceptual clarity in their user interfaces (cf. Davies, 1992). A likely source of that problem is an inadequate user's conceptual model.

Thus, this chapter has a distinct philosophical flavour. I can only hope that it is sufficiently clear that my ambitions in this respect are fairly limited. Let me sum up the limitations as follows. The task of providing a philosophy of corpus linguistics is completely subsumed under the tasks of developing a logical approach to corpus linguistics and the task of designing a tool for corpus linguistics, based on this approach. Also, I do not claim that I have provided *the* philosophy of corpus linguistics. I think that many views exist that are compatible with the methodology and the tool proposed in this thesis. It is for the promotion of clarity that *at least* one such view is presented, and it is to avoid confusion that *only* one such view will be discussed.

2.2 The Ontology of Text

2.2.1 Texts, facts, and states of affairs

What is a text? Consider the following concrete example:

(9) John loves Mary

This thesis assumes an ontology according to which a particular text is a collection of *facts*.[2] One fact holding in the example text above is that the first segment is an instance of the string “John”. Another fact is that the very same segment is a noun phrase denoting a certain individual. A third fact is that part of the content (or the context) of the text states that the individual is named “John”, and so on.

Now, (9) certainly is not much of a text. The text shown in Figure 1 – a text from the Brown corpus – is much more representative.

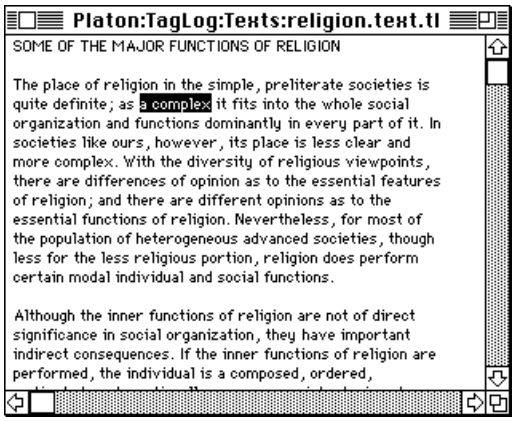


FIGURE 1. A Brown corpus text

Even though this text is not shown in its entirety – it consists of more than two thousand words – it is easy to see that it is a much more complicated object than the text in (9). It involves far more facts, and its syntactic, semantic, and pragmatic aspects are interrelated in much more complicated ways. Still, that ought to be a matter of degree, and some parts of it

---

2. Should I ever get around to writing my “Tractatus Taglogico-Philosophicus”, it would surely begin:

    1. A text is a totality of *facts*, not of strings.

are not harder to describe than the text in (9): the segment between positions twenty-two and twenty-four – the segment highlighted in the text – is a noun phrase, for example.

In this thesis we will concentrate on the facts of linguistic reality. However, texts are not the only things consisting of facts. People, flowers, computers, Christmas Eves, wars, and hotel rooms are also collections of facts. To a certain extent, such facts too are a matter of concern for us, not only by way of comparison with linguistic facts, but also on their own, since non-linguistic facts create a *context* for linguistic fact. Texts are parts of the world; they are *in* the world, and they stand in relations to lots of other things that are in the world.

Texts have *contents* (or, with another term, *interpretations*). Thus, not only do the facts of linguistic reality involve segments, strings, and grammatical categories, they also involve content. The content can be thought of as statements describing states of affairs (forming a situation) that, in turn, involve individuals, their properties, and relations between them. (The notion of 'statement' will soon be defined.)

Typically, knowledge of the content of a text is gained by *reading* the text. Perhaps less evident, knowledge of the *context* of a text can also be gained by reading the text, while trying to figure out what one has to know in order to be able to get at the content of the text, if one did not already know it. We can also learn about the described and presupposed situation (and thus the content and context) in other ways: by observing them directly, or by reading other texts about them, for example.

Let us also make a distinction between facts and *states of affairs*. Facts are states of affairs that hold. States of affairs that do not hold are not facts. States of affairs are possible facts. As an example, consider the state of affairs in which the first segment of our example text in (9) is an instance of the lexical category verb. This does not hold, and thus, even though it is a state of affairs, it is not a fact.

2.2.2 Points, segments, properties, property values, and relations

States of affairs are by no means atomic. Linguistic states of affairs of the kind we are discussing here typically involve text *segments* (which in turn involve *points*), *properties* and *property values* of text segments, *relations* between text segments, or between text segments and entities in the extra-

linguistic reality. Extra-linguistic states of affairs may involve other kinds of things, for example *objects* (things), *events*, *states*, and *statements*.

The notions of point and segment can be introduced simultaneously. A point in a text is simply that which delimits two adjacent segments. A segment of text is that which is delimited by two points in the text.

Segments have a very special status in the picture that we are developing in this thesis. Segments make up the *substance* of a text. Here, "substance" refers to the old Aristotelian idea of a 'bearer of property', rather than to Saussure's concept of substance. I will use the term "segment", in this Aristotelian fashion, to refer to the bearer of a property relevant to text. Another very important thing about segments is that they are located in time and space. From this it follows that they can be *observed*, and that they can be *pointed at*. The importance of observation and pointing will become apparent as we go along.

Moreover, as will be seen in Chapter 3, segments can be given unique *names*, and these names can be used in order to say things about the segments.

Note that segments and strings are not the same. Rather, segments are *instances* of strings, and two segments may very well be instances of the same string (note that in this sentence, the string "instances" has four instances!). Of course, this is just an example of the well-known distinction between *types* and *tokens*. Strings are abstract entities – types rather than tokens – whereas segments are tokens.

A final important fact about segments is that they are always immersed, as it were, in *context*. It is not common to conceive of strings, or sentences, in this way; they are usually regarded as *decontextualized* entities. Segments, by contrast, are always surrounded by context. Usually a distinction between two kinds of context is made: *co-text* and *situational context*. I would like to characterize the difference as follows: The distinction between co-text and situational context is a matter of 'underlying substance'. Co-text consists of text segments, and situational context consists of states of affairs pertaining to the non-linguistic world.

Let us also consider the distinction between a property and a value of a property. This distinction is not controversial. To take an example not relevant to text, *colour* would be a property, while *red*, *yellow*, *green* etc.,

would be possible values of this property. Here's another example, this time relevant to text: *syncat* (short for 'syntactic category') is a property while *noun*, *adverb*, and *noun phrase* etc. are possible values of this property.

2.3 Statements and Theories

In science, *theories* are of great importance. In TagLog too, theories are important. However, there are (at least) two notions of 'theory' around. A theory can be regarded as a set of sentences, defined by an axiomatic system. This is a *syntactic* notion of a theory. Alternatively, a theory can be regarded as a system of *universal statements*. This is a *semantic* notion of a theory. These notions are related in so far as utterances of sentences express statements. I will discuss and make use of both notions. In this chapter, however, the focus will be on the latter. As a starting point an analysis of the notion of statement is given.

2.3.1 Statements

Statements are, to put it simply, what 'declarative sentences assert to be the case'. Thus, statements fall into a semantic category. Statements can be *expressed* by different means. For example, statements can be expressed by (utterances of) sentences, if by 'sentences' we mean natural language sentences, or formulas in a formal language. Statements can also be expressed by non-verbal means – gestures, pictures, and so on – or by a combination of verbal and non-verbal means.[3] Note also that *one and the same* statement can be expressed by different means, e.g. by two different sentences, perhaps in two different languages.

To illustrate, consider the segment highlighted in the text in Figure 1 on page 32. Here are a couple of statements about that segment of text, expressed in ordinary English:

(10) The segment between position twenty-two and twenty-three is a determiner.
The segment between position twenty-three and twenty-four is a noun.
The segment between position twenty-two and twenty-four is a noun phrase.

3. The making of a statement by means of a combination of verbal and non-verbal means will turn out to be important. More on this in Chapter 6.

A distinction is often made between *singular statements* (particular statements) and *universal statements*. Singular statements are statements that describe singular facts. A singular statement has the form 'x is A' (or, 'It is not the case that x is A'). An example would be an utterance of 'Socrates is a human', or an utterance of 'This is not a swan' while pointing to something. Note that the statements in (10) are singular statements. Sometimes, I will use the term *observation statement* when I want to stress the connection to actual observation.

Universal statements have the form 'All A's are B's', or 'For any x, if x is A, then x is B', where A and B are descriptive terms. An example would be an utterance of 'Every human is mortal'. Another, more grammar-like example is given in (11).

(11) For any positions x,y and z, if the segment between x and y is a determiner, and if the segment between y and z is a noun, then the segment between x and z is a noun phrase.

Thus, instead of saying about a particular segment, consisting of a determiner followed by a noun, that it is a noun phrase, we say about *all* segments consisting of a determiner followed by a noun, that they are noun phrases. Not only does the general statement in (11) summarize the observations in (10). It also transcends these observations and points to other texts of which it predicts that determiners followed by nouns will be noun phrases. This way of transcending experience is a hallmark of a scientific descriptive generalization.

Statements relate to the rest of the world, in that they *describe* states of affairs. This is indeed an important property of statements, and it will be the topic of Section 2.4. But statements relate to *each other* in various ways as well. For our purposes, the most important relation between statements is *entailment*: the fact that certain statements follow logically from others. Another important notion is *consistency*: statements can be logically compatible or incompatible.

Relating to entailment, there are a couple of important roles played by statements in science that deserve to be introduced at this point: statements enable *explanations* to be made of specific matters of facts, and they enable *predictions*.[4] The first two statements in (10), together with

4. As has often been remarked, the logical structure of prediction and explanation is identical (cf. Hempel & Oppenheim, 1948).

the statement in (11), for example, both explain and predict the following statement:

(12) The segment between position twenty-two and twenty-four is a noun phrase.

Note, finally, that we need not discuss the deeply philosophical problem: What *exactly* is a statement? Usually, they are regarded as abstract, but nothing really hinges on the position that we take on this question here.

2.3.2 Theories

I will use the term *theory*, without qualification, when I mean to refer to a set of statements, and leave unspecified whether the statements are universal, or singular, or both. But in many circumstances, it turns out to be important to distinguish between two kinds of theories. On the one hand, we have *universal theories*, consisting of universal statements only. These are Theories with a capital T. When we speak of scientific theories, these are the kind of theories that we have in mind. On the other hand, sets of singular statements are also theories. In the particular context of corpus linguistics, they are often used to describe the data of corpus linguistics. When I can do so without causing confusion, I will allow myself to refer to this kind of theories as *data descriptions*, or simply as *descriptions*.

2.4 Truth and Truthlikeness

As mentioned above, statements relate to the rest of the world in that they describe states of affairs. In doing this, statements can be *true* or *false*, or their truth value can be *unknown*. If the states of affairs described by a statement *hold* (i.e. if they are facts) the statement is said to be true (otherwise false or unknown).

For example, the singular statements expressed in (10) are all true, since the segment highlighted in Figure 1 on page 32 *is* a noun phrase, and since the first and second part of this segment *is* a determiner and a noun, respectively. On the other hand, if the statement in (13) is taken to be about the second part of that segment, it is false, since this is *not* a verb.

(13) The segment between position twenty-three and twenty-four is a verb.

A universal statement of the form ‘For any x, if x is A, then x is B’ is false only if there is an x such that ‘x is A’ is true but ‘x is B’ is false. It is true otherwise. A true *theory,* then, would be a theory in which every statement is true.

Truth is an important property of theories; maybe, along with interestingness and coverage, the most important of their properties. But truth is in many ways a problematic notion. First, there are, some philosophers say, no *criteria* for truth. So there is no way to tell a true statement from a false one, they say. This claim, however, should be evaluated against the examples in (10) and (13). Clearly these sentences differ in exactly this respect: one is true, and one is false. It may be that when philosophers say there are no criteria for truth, they are thinking of some kind of *absolute* truth. But absolute truth (whatever that is) need not concern us here. We are perfectly happy with criteria (based on observation or whatever) that let us tell (10) and (13) apart.

When it comes to universal statements, matters are different. It is characteristic of (scientific) universal statements that they claim something about a very large (sometimes infinite) number of things, and that what they claim concerns the future, and the past, as well as the present. This means, as Karl Popper has repeatedly stressed, that such a statement cannot be conclusively verified. It can only be falsified, since to falsify a universal statement, it is enough, in principle, to find one falsifying instance of it.

This gives another meaning to the claim that there are no criteria for truth, and it means, indeed, that *falsity* has an important role to play in science. The quest for truth is important, but notions of falsity, inconsistency, conflict, and error are of no less importance, in that they suggest points where theory can be improved. Improvement is ‘error-driven’, so to speak.

Another problem, from our point of view, is that truth is an ‘all-or-nothing’ notion. One falsifying instance of a universal statement makes it just as false as a million falsifying instances would do. But in *practice,* finding one falsifying instance of a statement is not likely to make us abandon it, especially not if we lack a better alternative. Furthermore, given two false statements or theories, one of them might be better than the other. It seems that we need a way to speak of *degrees* of truth.

I will therefore – once again borrowing from Popper – introduce a notion of *truthlikeness*.[5]

> We have even good reason to think that most of our theories – even our best theories – are, strictly speaking, false; for they oversimplify or idealize the facts. Yet a false conjecture can be nearer or less near to the truth. Thus we arrive at the idea of nearness to the truth, or of a better or less good approximation to the truth; that is, at the idea of '*verisimilitude*'.
>
> *Popper (1972, p. 318)*

This criterion, this measure of truthlikeness, is going to look roughly as follows. The higher the proportion between the number of confirming instances and the number of falsifying instances plus the number of confirming instances, the more truthlike is the theory. Hence, the degree of truthlikeness of a statement, contrary to the truth of a statement, can be measured from what instances of it we have seen *so far*.

So what is to become of truth in itself? I wish to advocate truth as a *regulative principle* for TagLog theory development. Even though we may never be *completely sure* if and when we have found truth (for reasons given above), truth is nevertheless something we ought to strive for.

The principle is simply this: *think* about truth. Do not allow yourself to be satisfied with a theory just because it works for a certain purpose. If it works for a certain purpose, it is presumably not totally wrong, but it may still be possible to improve – it may still be made more truthlike – and may then become useful for other purposes as well.

I believe that in particular the results reported in Chapter 14 prove that this is a fruitful methodological principle to keep in mind during theory development work in TagLog.

2.5 Corpus Linguistics as an Activity

An analysis of the scientific activity that is to be aided by the corpus tool proposed in this thesis must start from some idea of what the goals of this activity are. I take the main long-term goal of the science of corpus lin-

5. It should be noted, however, that the notion of truthlikeness defined in this thesis, is not identical to Popper's notion.

guistics to be *true and wide-coverage and explanatory theories of texts*. The goal is to be realized through the pursuit of scientific activity in the domain.

2.5.1 Three things linguists do

We will conceive of the enterprise of scientific corpus linguistics in general as being made up of three main kinds of interrelated activities:

1. Development of descriptions (i.e. sets of observation statements) of particular texts (usually referred to as 'tagging' or 'coding').
2. Development of descriptive universal theories of (sets of) texts (e.g. grammars).
3. Development of theoretical principles (e.g. grammar theory).

The relation between a description of a particular text and a general descriptive theory must be roughly as follows: The description of data must 'fit' (be syntactically compatible with) the statements of the descriptive theory. Only then is it possible to test the latter statements with respect to data, and only then do configurations of data 'suggest' statements to be included in the descriptive theory.

The relation between a descriptive general theory and theoretical principles is roughly a follows. Theoretical principles determine the structure of categories and (e.g.) grammar rules. But descriptive theories do not 'follow from' theoretical principles. Some properties follow, but essentially, descriptive theories must be developed independently, in close contact with data.

It is in the second activity – the work of developing descriptive general theories – that corpus linguists and theoretical linguists meet, or should meet. Yet, as I have argued, traditional computational tools used by theoretical linguists (e.g. grammar development environments) do not usually support access to corpora, while traditional corpus tools do not usually support formalisms in which categories and grammar rules derived from theoretical principles can be formulated. TagLog can be seen as an attempt to fill this need.

2.5.2 The dynamics of theory development in linguistics

Popper (1972) proposes that the following schema captures the 'dynamics' of scientific theory development.

(14) $$P_1 \Rightarrow TT \Rightarrow EE \Rightarrow P_2$$

A problem P_1 is formulated; a tentative theory *TT* is proposed; error elimination *EE* is attempted; and if errors were found they give rise to further problems P_2, and so on.[6]

This is how we might conceive of the process of developing descriptive corpus theories as an instance of Popper's schema:

1. Problems are detected, either in the form of facts not accounted for by the current theory, or in the form or contradictions or other kinds of errors that appear in the current theory.
2. Updates (additions or deletions or modifications of statements) to the current theory, accounting for the new facts, or correcting the errors, are proposed (tentatively).
3. The (relevant part of) the current theory is tested (again), that is, an attempt to eliminate errors is made.
4. Often, new problems arise, concerning how to correct errors detected in 3.

However, Popper's schema applies to the development of theoretical principles as well.

1. A *big* problem is detected, perhaps a source of systematic (descriptive) errors. Something is *fundamentally* wrong.
2. A way to restructure the whole theory is proposed (tentatively). Novel categories, new ways of writing grammar rules, etc. are introduced.
3. Testing: New descriptive theories, formulated using the strategy proposed in 2 are developed (using the low-level schema). These statements are tested, and errors that arise as a consequence of them are detected.
4. New (big) problems may arise as a consequence of performing 3.

---

6. This schema, or method, is referred to by Popper alternatively as *the method of conjecture and refutation*, *the critical method*, or *the method of trial and error.*

Theoretical principles can be seen as means of developing descriptive theories. But it is also the case that development of descriptive theories can be seen as a way of testing theoretical principles. Theoretical principles are testable with respect to descriptive theories in the sense that a particular set of theoretical principles passes the test only if it allows the formulation of true descriptive theories.

We will see the radical revision of theoretical principles happen a couple of times in the course of this dissertation. We will place phenomena into novel theoretical frameworks, rather than modifying old ones by piecemeal changes. We will see different grammar theories, different ways of looking at a lexicon, and different ways of making grammars sensitive to context.

2.6 Summary and Conclusion

In this chapter, matters of foundation were discussed. Corpus linguistics was characterized in terms of its object of study – texts – and in terms of its long-term goal – true, wide-coverage theories about texts.

Text was subjected to an analysis in terms of facts and states of affairs. The facts and states of affairs of linguistic reality involve segments, properties, and relations. Segments (in contrast to strings) are immersed in context, they can be observed, and they can be pointed at. Segments can also be named.

A fairly standard view of theories as systems of universal statements was presented, and universal statements were contrasted with singular statements. Statements were said to describe states of affairs, and were said to be true if the states of affairs described hold, but false otherwise. Furthermore, the relation of entailment between statements, and the property of consistency of a system of statements, were introduced. The role of the notions of entailment in prediction and explanation was anticipated. Finally, I have argued for a simple, de-mystified view of truth. Truth, not as a matter of metaphysics, but as a methodological principle.

Also in this chapter, corpus linguistics was characterized as a particular kind of *activity.* The approach to corpus linguistics advocated here does not treat the logic of corpus linguistics in isolation from the activity of

corpus linguistics. The logical approach to corpus linguistics is logic in action.

The hope is that a concern for first principles of corpus linguistics will help to clarify the role of the system and the role of the researcher in the scientific process, and thus give us, the system designers, better ideas of what kind of tools to implement, and how to let the user interact with them. Also, the hope is that a system of notions from the philosophy of science will serve as a good *conceptual model* for the users, and give them the ideas and the perspective that will support a correct and fruitful use of the system.

CHAPTER 3

The Logic of TagLog

3.1 Introduction

In this chapter, a notion of *formal theory* is developed, that to some extent will allow us to capture the informal view of a theory explained in Chapter 2. This notion of formal theory is rather standard (see Partee, ter Meulen, &Wall, 1990), in that it consists of three parts:

- A formal language
- A finite set of *axioms* (or *axiom schemata*)[1]
- A finite set of *rules of inference*

From the axioms, a (potentially infinite) number of additional sentences – the *theorems* – are derivable by repeated application of the rules of inference.

Again, this is a *syntactic* notion of a theory, and the inference rules are possible to apply in purely *mechanical* way, without knowing anything

1. By 'axioms' I refer to non-logical axioms, rather than logical axioms.

about the meaning of the formulas. Obviously, this is exactly what makes them useful as a basis for computation. The formal system is of course *motivated* by our desire to reflect corresponding relations of logical entailment with respect to statements. As will be seen, this is where the notions of *intended interpretation*, *soundness*, and *completeness* comes in.

Language design is a big thing in computer science. Almost every large area of application demands its own language: programming languages, knowledge representation formalisms, database languages, grammar formalisms, and so on. In this chapter, I present my current choice of formalism for description and analysis of text, and I discuss some of the design decisions that I have made in order to come up with a language to facilitate the 'talking about text' with computers. I hesitate to use the word "design" though, because our language will only be a rather moderate extension of Horn clause logic, similar to Prolog, but with a different set of built-in predicates, and in other ways tailored to its task.

The TagLog formalism is a *declarative* language. One of the great advantages of using a declarative formalism is that expressions are possible to understand without knowledge of how they will be put to use. The meaning of expressions is a matter of the relation between the symbols of the formalism and elements of the world, plus the way the symbols are combined, and nothing else. In this chapter, but even more so in the next chapter, theories about text will be presented, expressed as sets of axioms interpreted by the user. Not until later chapters, however, will the use of these theories by the TagLog analysis tools be discussed.

The third component of a formal theory – the rules of inference – will also be given proper attention. In particular, reasoning in TagLog by means of the resolution principle is explained.

First, however, before moving on to an ample presentation of the language of TagLog, some important requirements for such languages will be discussed. Towards the very end of the present chapter, I perform a brief evaluation with respect to these criteria.

3.2 Requirements for the TagLog Language

Let's list some of the requirements for the TagLog language:

1. It should have the *necessary expressive power*, i.e. it should allow us to express the statements that are needed to form theories about texts, and it should support the functionality needed for analysis of the texts by means of the theories.
2. It should have an *explicit declarative semantics*, i.e. it should be understandable without reference to the procedures that use it.
3. It should have an *explicit theory of inference*, with known properties with respect to soundness and completeness (note that I'm not saying that it must be sound and complete).
4. It should be *computationally tractable*. In fact, with an eye to the vast amount of input data common in the intended application, it must allow very efficient implementation.
5. It should be *easy to read*, *easy to write*, and *easy to learn*.

In my view, the requirements 1 – 4 are extremely important. However, I consider 5 to be slightly less important, for reasons to be given later on.

There is a well-known trade-off between the expressive power of a language and its computational tractability. The more expressive power a formalism has, the less efficiently we can reason with it. To put it drastically, languages "will either be limited in what knowledge they can represent or unlimited in the reasoning effort they might require" (Levesque & Brachman, 1985, p. 43).

To simplify things, we shall take as our point of departure a language that is known to strike a very good compromise between expressiveness and efficiency, namely *Horn clause logic*.

There is a simple and efficient inference rule for Horn clause logic – *the resolution principle* – that is both sound and complete.[2] Moreover, Horn

2. Prolog's proof procedure is incomplete, however, since it uses a depth first strategy for applying the resolution principle, that will lead to non-termination in certain cases. Also, due to lack of occurs check, most implementations of Prolog are not *sound* either (see e.g Hogger, 1990).

clause logic is quite powerful and, perhaps surprisingly, most of what we want to be able to express turns out to be expressible in Horn clause logic. But it has its limitations: there is no classical negation available, for example, and as a consequence there are certain statements that we simply cannot express.

Now, for some of the ideas that we will want to develop in the TagLog framework, it turns out that we will need to be able to reason about negative information. In Chapter 10, we will therefore extend the Horn clause logic, very carefully, so that no efficiency is lost in the process, by adding (a limited form of) classical negation and various meta-logical operators. These extensions will bring us more expressive power, but one disadvantage, which may not matter, is that completeness is lost, so that a great many conclusions that actually follow logically from a theory expressed in the language, cannot be drawn.

What about the requirements in 5? Perhaps these are the requirements that seem to have stood out as most important for traditional approaches to tagging formalisms. But if they are given prominence over the others, we are bound to end up with something simple, and hence, with something not very expressive. The requirements in 5 should really be formulated as follows instead: the language should be as easy to read, write, and learn, as possible, *given* the expressive power required.

3.3 The Language of TagLog

Most of the syntax of TagLog has been taken over from the programming language Prolog (Clocksin & Mellish, 1981; Sterling & Shapiro, 1986). Prolog is a well-established programming language which was invented in the early seventies and for which an ISO standard has now been created (ISO, 1995).

3.3.1 TagLog native syntax

Since the reader of this thesis is likely to be a linguist, I prefer to give the syntax for the TagLog language in a simple context-free grammar notation, rather than in a style that perhaps mainly computer scientists are acquainted with. The only meta-symbol used here is '→'.

(15)
taglog_theory → clause
taglog_theory → clause taglog_theory

clause → predication :- predications .
clause → predication .

predication → predicate_name
predication → predicate_name (term_sequence)

predications → predication
predications → predication , predications
predications → predication ; predications

term_sequence → term
term_sequence → term , term_sequence

term → constant
term → number

term → segment_expression
term → set_expression
term → list
term → string
term → variable_name
term → functor_name (term_sequence)

set_expression → { matrix | predications }

matrix → variable_name
matrix → (variable_name , matrix)

query → !- predications .

We now go on to exemplify these constructions, and in passing introduce some more terminology:

(16) lexcat(3-4,adj).

(17) function(S,statement) :- mood(S,declarative).

(18) {S | syncat(S,n)}

The expression in (16) is a clause consisting of a single predication, which in turn consists of a predicate name, an opening parenthesis, a segment-expression, a comma, a constant, a closing parenthesis, and a full stop, occurring in that order. More specifically, it is a *unit clause*, or as it is often called, a *fact*. Note that the term “fact” denoted something rather different in Chapter 2: something ‘in the world’ rather than a syntactic construction. Here, I will allow myself to use the term in both senses, and rely on the context for deciding between them.

The expression in (17) is also a clause. The ‘:-’ symbol is an implication arrow, pointing ‘backwards’, so that the antecedent is on the right, and the consequent is on the left. Variable names start with an upper-case letter,

and variables are implicitly universally quantified. Thus, the statement in (17) would be written as (19) in standard predicate logic notation.

(19) ∀*s*[*mood(s,declarative) -> function(s,statement)*]

In Prolog terminology, a clause like the one in (17) is often called a *rule*. A user with an instrumentalist inclination, who sees the statements of science as rules of derivation, may be happy with this term. Other users, however, would want to reserve the term 'rule' for grammar rules and condition-action rules (see Section 3.3.3), and instead speak of constructions like the one in (17) as 'universally quantified implications', or 'non-unit clauses'.

The expression in (18) is a set-expression. This is one of the few constructions in TagLog that is not part of Prolog, at least not on the surface.

3.3.2 The grammar notation

In TagLog, a special purpose *grammar notation* can be used to express grammar rules. The syntax of this notation is given in (20).

(20)
taglog_theory → grammar_rule
taglog_theory → grammar_rule taglog_theory

grammar_rule → head –-> body .

head → category

body → categories

category → functor_name
category → functor_name (term_sequence)

categories → category
categories → category , categories
categories → category ; categories

Here is an example of a grammar rule written in this notation, consisting of one atomic category in the head, and two categories in the body, that both contain a variable *Num*:

(21) s --> np(Num), vp(Num).

It is important to realize that grammar rules may just as well be written according to native TagLog syntax. However, the expression in (21) is simpler and more familiar to a linguist than the corresponding expression in TagLog native syntax would be. The use of grammar rules is exemplified in Chapter 4, and more details on the translation of grammar rules into native syntax are given in Chapter 5.

3.3.3 Condition-action rules

TagLog has at its disposal also a type of rule that I have chosen to call *condition-action rule*. The syntax for condition-action rules is as follows:

(22) Condition_1,..., Condition_n ==> Action_1,..., Action_m

where $Condition_i$ and $Action_j$ are TagLog predications and TagLog actions (such as adding or deleting clauses from a theory), respectively.

The meaning of such a rule is that the actions $Action_1$,...,$Action_m$ are executed once for each way in which the conditions $Condition_1$,..., $Condition_n$ can be proved. An interpreter for condition-action rules can be implemented as in (23).

(23)
```
ca_run :-
        (       (Condition ==> Action),
                call(Condition),
                once(Action),
                fail
        ;       true
        ).
```

Here's an example:

(24)
```
bstring(S,_+"s") ==> add(lexcat(S,v)).
```

This rule will add a statement *lexcat(S,v)*, with *S* instantiated to a segment expression, for each *S* that instantiates a (basic) string that ends in "s".

Condition-action rules do not express statements. They do not have a truth value, and they are not logical. They do not form theories. They are simply *tools*, powerful and handy.

3.4 What is a TagLog Theory About?

Section 3.3 provides a few examples of what might be considered axioms: (16) and (17), as well as (21). But what do they *mean*; what are they about?

First, it is very important to note that the 'aboutness' of the sentences of a theory is, to a great extent, a question of *intended interpretation*. It is the user who gives content to TagLog theories. The intended interpretation of a formal language specifies two things:

- the domain, and
- the meaning of the symbols of the language in the domain

In our case, the relevant domains are simply the constituents of the facts of the linguistic and extra-linguistic reality, that were discussed in Chapter 2: segments (and other kinds of 'substance'), their properties, relations between segments and between segments and other things, and so on.

The meaning is the associative links, for most parts established by the user, between the constants, function symbols, and predicate symbols of the TagLog language, and objects in the domain.

3.4.1 The interpretation of terms

Among the terms in the TagLog language, we find constants, variables, and complex terms.

Constants
In general, individual constants denote (simple) property values, or individuals in the text or in the extra-linguistic reality. Most of these links are established by the user. For example, the user may decide to let the constant *np* denote the syntactical category noun phrase, and he may decide to let the constant *x378* denote the university library in Göteborg.

A *point expression* is a special kind of individual constant that denotes a rather special kind of individual: a point in the current text. The link between the point expression and the point in the text is established automatically by the system, and hence, is not a matter of the user's association. As will be seen, point expressions play an important role in segment expressions.

Complex terms
By means of the application of functor names to term sequences, complex terms, nested to an arbitrary depth, can be built.

Typically, complex terms in the language denote complex properties, such as grammatical categories. For example, a term like *v(fin(3,sg),[np,pp])* might be used to denote a syntactical category (finite verb, third person singular, subcategorizing for a noun phrase and a prepositional phrase).

A *segment expression* is a special kind of complex term – consisting of functor '-' infixed between two point expressions – by means of which each segment in a text is given a unique 'name'. For example, the segment between the third and the sixth point in a text is given the name *3-6*. We say that *3-6* is a segment expression that *denotes* the segment in question.

3.4.2 The interpretation of formulas

Atomic sentences

By applying predicate symbols to sequences of terms, atomic formulas are composed in such a way that when stated they describe states of affairs. Such a formula is *true* if and only if the relation denoted by the predicate symbol holds between the entities denoted by the terms in the sequence. More formally:

(25) $P(t_1,...,t_n)$ is **true** iff the interpretation of P is an n-ary relation that holds between the interpretations of $t_1,...,t_n$

As a special but very common case, let us look at *tags*. In order to classify segments in the text, we use atomic sentences of the form *P(S,V)* where *P* is referring to a property, where *S* is a segment expression referring to a segment in the current text, and where *V* is a term referring to *a value* of the property denoted by *P*. We say that *P(S,V)* is true iff the segment has that value for that property. For example, *syncat(234-236,np)* is true iff the segment *234-236* in the current text is in fact a noun phrase.

Complex sentences

So far we have considered only the simplest sort of sentence – atomic formulas. A conventional truth conditional semantics for *complex* Horn clause logic sentences in the TagLog language might be given as follows.

For variable-free (or 'ground') sentences:[3]

(26) (A_1 , A_2) is **true** iff A_1 is **true** and A_2 is **true**
$(A_1 ; A_2)$ is **true** iff A_1 is **true** or A_2 is **true**
$(A_1 :- A_2)$ is **false** iff A_1 is **false** and A_2 is **true**

If a clause contains variables, they are universally quantified, and thus:

---

3. The term 'ground' is logic programming jargon for 'variable-free'. I will use both terms in this thesis.

(27) A is **true** iff each of its variable-free instances is true

If a query contains variables, they are existentially quantified, and thus:

(28) A is **true** iff it has some variable-free instance which is true

Together, the rules in (25), (26), (27) and (28) enable us to determine the truth-value of any sentence under a given interpretation.

3.4.3 The interpretation of theories

Theories consist of sentences. A theory is true iff each of its sentences is true. To summarize and exemplify, let us use the TagLog language to express the statements about the segment selected in Figure 1 on page 32 (i.e. the instance of "a complex"). This segment can be described by means of the clauses in (29).

(29)
```
syncat(22-23,det).
syncat(23-24,n).
syncat(22-24,np).
```

The statements expressed by the sentences in (29) are true, under the (obvious) intended interpretation, since (again) the segment in question *is* a noun phrase, and since the first and second part of this segment *is* a determiner and a noun, respectively. On the other hand, the statement in (30) is false, since (again) the segment referred to is *not* a verb.

(30)
```
syncat(23-24,v).
```

To say that every segment consisting of a determiner followed by a noun is a noun phrase, the sentence in (31) is sufficient.

(31)
```
syncat(P0-P2,np) :- syncat(P0-P1,det), syncat(P1-P2,n).
```

3.5 The TagLog Built-In Predicates

This section describes some of the built-in predicates available in TagLog. These predicates are provided in advance by the system and cannot be redefined by the user.

Some of the built-in predicates reflect the basic ontology assumed in TagLog (see Chapter 2), in the sense that they define basic relations over

segments, strings, and other entities. The relations can be referred to in sentences formulating definitions or descriptive generalizations, and as will be shown, they can be referred to in set-expressions specifying segments to search for or count, and in set-expressions specifying concordances and tables.

The following categories of such built-in predicates are available:

1. Segment predicates
2. String predicates
3. Tree predicates
4. Meta-level predicates
5. Arithmetic
6. Schema predicates

Other TagLog built-ins define the functionality of the system, in terms of what analyses can be performed, and how their results can be presented:

7. Presentation procedures
8. Corpus text and corpus theory management procedures

3.5.1 Segment predicates

The notion of *segment* is extremely important in TagLog, and the TagLog system comes with a family of built-in predicates to deal directly with segments. Here, only an overview of the most important segment predicates is given, but a complete and systematic listing of all the available predicates can be found in Appendix A.

First, there are predicates for checking if something is a segment in the current text, and there are predicates for determining the length of a segment.

As explained in Chapter 2, a segment is a piece of ‘text substance’ stretching between two points in the text. Consider the ‘utterance’ in (32), where the relevant points are indicated by subscripts.

(32) $_1$ bla $_2$ blabla $_3$ bla $_4$ blablabla $_5$

Here, 1-2 is a segment, 2-3 is another segment. As a matter of fact, there are 10 different segments in (32).[4] The predicate *segment/1* is true of each of them.[5]

The *length* of a segment is given by the number of points between the points that delimit the segment, plus one. Thus, the length of the segment between two adjacent points is 1. The length of the segment 2-5 in (32) is 3. The TagLog built-in predicate *segment_length/2* holds between a segment and its length.

A *basic segment* is a segment of length 1. The predicate *bsegment/1* is true of each of the basic segments in the current text.

Figure 2 gives an overview of some of the possible relations that can hold *between* two segments.

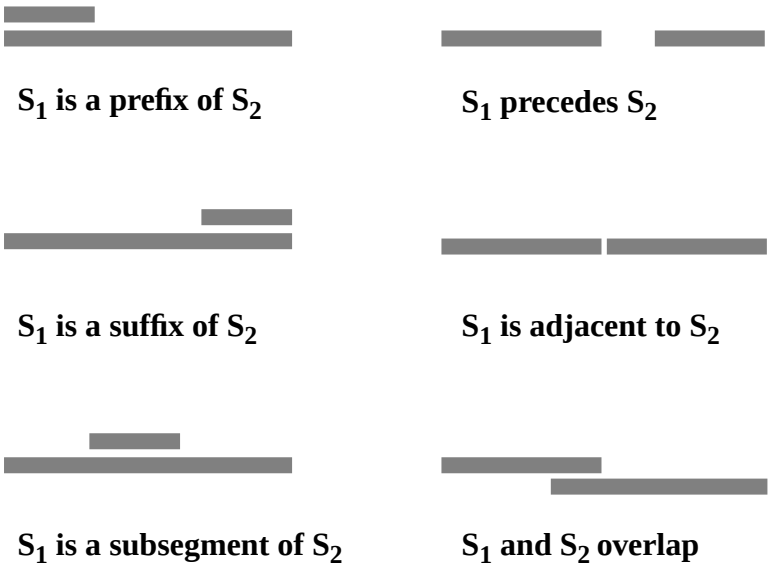


FIGURE 2. Some possible relations between two segments

---

4. There are *(n - k + 1)* subsegments of length *k* in a segment of length *n*. The total number of subsegments is therefore given by:

$$\sum_{k=1}^{n}(n-k+1) = \sum_{k=1}^{n} k = (n+1)\frac{n}{2} = \frac{(n^2+n)}{2}$$

5. I do not assume that there are *discontinuous* segments. I suspect that what most people are tempted to refer to by that name, is something that I would regard as *two* (or more) segments, rather than one segment ‘distributed’ over different locations.

There are predicates to relate a segment to its parts: prefixes, suffixes or subsegments; and there are predicates that relate a segment to surrounding and overlapping segments.

The predicate *segment_prefix/2* holds between a segment and its prefix segments, whereas *segment_suffix/2* holds between a segment and its suffix segments. The predicate *subsegment/2* holds between a segment and its subsegments.

In general, these predicates can be used ('called') with all sorts of instantiation patterns, i.e. they can be used to *check* if two segments stand in a particular relation, or they can be used to *find* a segment that stands in a particular relation to a given segment. They can even be used to *enumerate* all the pairs (or triples) of segments that stand in a particular relation.

3.5.2 String predicates

As we saw in Section 2.2.2, segments and *strings* must not be confused. Strings are very important objects in TagLog, almost as important as segments. Consequently, there are built-in predicates for reasoning about strings. In certain respects they are isomorphic to the segment predicates, although great differences exist as well.

Thus, TagLog provides the user with predicates for reasoning about the relation between segments and strings. There is a predicate for computing the length of a string, and there are predicates for reasoning about the relations between a string and its prefixes, suffixes and substrings.

The atomic formula *string(Segment,String)* is true iff *Segment* is a segment in the current text and *String* is the string of which *Segment* is an instance; *bstring(Segment,String)* is true iff *Segment* is a *basic* segment in the current text and *String* is the string of which *Segment* is an instance.

The predicate *string_length(String,Length)* is true iff *Length* is the length, i.e. the number of characters, of *String*.

The atomic formula *string_prefix(String1,String2)* is true iff *String2* is a prefix of *String1*. A similar predicate for reasoning about the suffix relation is available as well. Finally, *substring(String1,String2)* is true iff *String2* is a substring of *String1*.

3.6 Reasoning in TagLog

Logical reasoning is the basis for prediction and explanation in TagLog, and an integral part of the generalized procedures for searching, counting, concordancing, etc., that are supported by TagLog.

3.6.1 Reasoning

Reasoning (theorem proving, inferencing) is the business of trying to determine whether a formula *A* is a theorem in the current theory or not, i.e. whether |-*A* holds or not.

Suppose, for example, that we want to prove *∃(X)[human(X)]* in a theory consisting of the following clauses (for clarity, I have used explicit quantifiers here):

(33) *biped(socrates)*
featherless(socrates)
∀(X)[human(X) :- biped(X), featherless(X)]

No matter what sound and complete proof method for Horn clause logic we use, we will get the same result. For example, we could prove *∃(X)[human(X)]* by performing a bottom-up *natural deduction* proof (which is called 'natural' since it is supposed to be the way humans perform reasoning).[6]

From *∀(X)[human(X) :- biped(X), featherless(X)]*, infer *human(socrates) :- biped(socrates), featherless(socrates)* (Universal Instantiation). From *biped(socrates)* and *featherless(socrates)*, infer *biped(socrates), featherless(socrates)* (Conjunction Introduction). Then, by Modus Ponens, infer *human(socrates)* from *biped(socrates), featherless(socrates)* and *human(socrates) :- biped(socrates), featherless(socrates).* Then finally infer *∃(X)[human(X)]* from *human(socrates)* (Existential Generalization).

The TagLog theorem prover does not use natural deduction to derive theorems, and it does not reason from the bottom and up. Nevertheless, the proofs generated by the TagLog theorem prover could be *thought* about in this way.

---

6. See e.g. (Allwood et al., 1977) for a short introduction to reasoning by natural deduction.

Alternatively, we could think about the initial *A* as a *query* (is there an *X* such that *X* is human?). In an attempt to answer the main query, the prover uses the clauses from the current theory in order to break the query down into simpler queries and to answer the simplest ones. (Is there an *X* such that *X* is human? Is there an *X* such that *X* has two legs and *X* has no feathers? Is there an *X* such that *X* has two legs? Yes, Socrates! Is Socrates featherless? Yes! So there is an *X* such that *X* is human, namely Socrates!) In fact, this is more in accordance with how proofs are actually handled by the system, top-down and all.

So how is it *actually* done? The resolution principle is somewhat more complicated than many other inference rules, since it is 'machine oriented, rather than human oriented' as Robinson, the inventor of resolution, puts it in his 1965 paper. Even though a user of the TagLog system does not have to know the intricacies of resolution-based proofs, a brief explanation is nevertheless included here for completeness.

3.6.2 The resolution principle

In this section, a rather standard introduction to the resolution principle will be given, akin to (and inspired by) the one given by (Hogger, 1990). Before presenting the resolution principle as such, I will introduce the notions of *substitution* and *most general unifier.*

A *substitution* is a finite set of pairs of the form *U/t* where *U* is a variable and *t* is a term. Thus any non-empty substitution σ takes the form $\sigma = \{U_1/t_1,...,U_m/t_m\}$. The application of σ to any formula *A* is the act of simultaneously replacing, for all $i \in \{1,..,m\}$, every free occurrence of U_i in *A* by t_i. The result, denoted by $A\sigma$, is said to be a *substitution instance* of *A*.

A *unifier* of two atomic formulas *A1* and *A2* is a *substitution* σ such that $A1\sigma = A2\sigma$; σ is the *most general unifier* (mgu) of *A1* and *A2* iff for every other unifier τ of *A1* and *A2*, $A1\tau$ is a substitution instance of $A1\sigma$. For example, the substitution {*X/socrates*} is the *mgu* for the formulas *human(X)* and *human(socrates)*.

A proof by resolution is a *proof by contradiction*: we want to prove *A*, so we assume the negation of *A*, and if a contradiction can be inferred, *A* is in fact proved. For Horn clause logic, and assuming a formula $A_1,...,A_m$

(possibly containing some existentially quantified variables) to be proven, the resolution principle can be given as follows:

$$\frac{\text{:- } A_1,...,A_k,...,A_m \qquad\qquad B \text{ :- } C_1,...,C_n}{(\text{:- } A_1,...,A_{k-1},C_1,...,C_n,A_{k+1},...,A_m)\,\sigma} \quad \sigma = \text{mgu}(A_k,B)$$

The formula to the left above the line is the negated form of $A_1,...,A_m$ expressed as a conditional with an empty consequent. It is called the *input clause*. To perform a resolution proof we apply the resolution rule repeatedly to the input clause and a clause from the current theory, the consequent of which unifies with a conjunct in the input clause, resulting in a certain *mgu*. The result is a new input clause where the application of the *mgu* may have replaced some of the variables by other terms. This is repeated until either the empty clause (i.e. ':-' with nothing in the antecedent and nothing in the consequent, meaning contradiction) is derived, and in which case the proof is completed, or it is clear that no such derivation is possible, and thus the proof has failed.

For example, suppose we want to prove $\exists(X)[human(X)]$, i.e. that there exists an *X* such that *X* is human. To perform a proof by contradiction we first negate this formula and get $\neg\ \exists(X)[human(X)]$. This is equivalent to $\forall(X)[\ \neg\ human(X)]$, which in turn can be rewritten as a conditional with an empty consequent $\forall(X)[\ :-\ human(X)]$. Finally, hiding the quantifier we arrive at *:- human(X)*, which is what the resolution principle wants. To derive our theorem from the theory, we start from this clause and we need to apply the resolution principle three times before arriving at a contradiction. The proof can be displayed as in Figure 3.

$$\cfrac{\cfrac{\cfrac{\text{:- human(X)} \qquad \text{human(Y) :- featherless(Y), biped(Y)}}{\text{:- featherless(Y), biped(Y)}}\ \sigma = \{X/Y\} \qquad \text{featherless(socrates)}}{\text{:- biped(socrates)}}\ \sigma = \{Y/socrates\} \qquad \text{biped(socrates)}}{\text{:-}}\ \sigma = \{\}$$

FIGURE 3. SLD resolution proof tree

The above principle is in fact a particular refinement of the resolution principle, called *SLD resolution.* It is demonstrably sound and complete, and it can be implemented very efficiently.

It is characteristic of SLD resolution proofs that they are *constructive*. The procedure returns the bindings of the variables (the substitutions) and hence not only do we learn that there exists a human, we also get to know an example. As we will see, this property is quite important for the particular use of logic that we will make in this thesis.

Horn clause logic has proved to be a stable basis for computing with logic. A lot of research has addressed the problem of extending to full first order logic, while still keeping the good computational properties, but with limited success. SLD resolution in Horn clause logic is probably still the most efficient way of computing in logic.

3.6.3 Proof trees

No matter how we think of these proofs, it is convenient to have a way of displaying a proof that makes it easy to follow. Proof trees are often displayed as in Figure 4, with the conclusion at the top, and the formulas on which they depend thereunder. This captures most of what we will need to know about a particular proof.

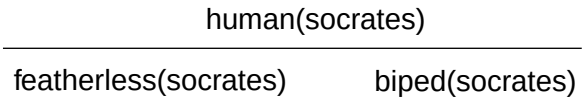


FIGURE 4. Proof tree

3.7 Summary and Conclusion

In this chapter, a notion of *formal theory* was developed, that will to some extent allow us to capture the informal view of theories explained in Chapter 2. Some important requirements for a language for description and analysis of text were proposed. Syntax and semantics for the formalism of choice – Horn clause logic – was given, and the only rule of derivation that is really needed – the resolution principle – was explained.

Many advantages come with the use of logic as underlying knowledge representation formalism: a great deal of expressive power, existing

results concerning soundness and completeness, formal semantics, and explicit theories of inference.

As will be seen, the availability of logical variables and universal quantifiers guarantee a level of expressiveness sufficient for a natural formulation of systems of definitions and descriptive generalizations, and the availability of logical connectives and 'higher-order' predicates involving sets and their properties, allows us to formulate and deduce answers to fairly complex queries.

A formalism like this, with an explicit declarative semantics, will allow us to enforce the view of corpus theories as declarative bodies of knowledge, 'detached' from the tools that use them. Everything is explicit, nothing is hidden. In the course of analysis, the user only has to state *what* (information) he wants to get as the result of a computation, not *how* he thinks the system should compute it. The user does not have to be a programmer.

If we pack the criteria concerning readability, writeability, and learnability together, and label them "ease of use", I think we would soon find that "ease of use" in this sense is, to a considerable degree, a question of previous acquaintance. Today a substantial number of people – linguists included – are familiar with the Prolog programming language, and this, I think, is a major argument for the choice of a Prolog-like formalism. Yet, TagLog is not Prolog. TagLog is not a programming language. The notorious *cut* in Prolog, for example, is not part of the TagLog language. Hence, TagLog is easier to learn than Prolog. But with no background in logic or Prolog programming, TagLog fluency does not come cheap. Part of my task here is to convince people without background knowledge in any relevant area that it is well worth the effort.

Despite these advantages it may not be entirely clear that Horn clause logic is a formalism useful for our purpose. Corpus research often involves searching and counting, but can we search and count with logic? Can we build concordances? Can we implement good methods for automatic tagging? Yes, from a logical point of view, as we will see in Chapter 7, searching is not different from other kinds of deduction. Furthermore, as we shall see as well, the expressive power of the TagLog (query) formalism is certainly enough to allow natural and declarative specifications of concordances (see Chapter 7), tables of collocations (see Chapter 9), etc. to be written. In fact, the major part of the rest of this thesis can be read as an argument for this.

CHAPTER 4

TagLog Theories

> Theories are nets cast to catch what we call 'the world': to rationalize, to explain, and to master it. We endeavour to make the mesh ever finer and finer.
>
> *Karl Popper*

4.1 Introduction

In this chapter, we explore the expressive power of the TagLog formalism, and use this power to develop various theories about different aspects of texts. Two kinds of theories will be considered: data descriptions and universal theories.

The actual theories as such are not important. What *is* important is to gain an understanding of the numerous possibilities inherent in the logical approach to corpus linguistics, an understanding of how lexical, syntactic, semantic, and pragmatic aspects of corpus text can be captured by means of TagLog universal theories and descriptions. It is impossible for us to explore all the possibilities here, however. When the expressiveness of TagLog meets the creativeness of corpus linguists, anything can happen.

By placing the present chapter *before* the chapters describing various procedures and tools producing and making use of descriptions and theories, I am trying to make the following point. TagLog theories and descriptions can be *understood* without reference to how they came about, or how they are to be used. All one needs to know is what the formulas *mean*, what they *claim* about their subject matter. This, again, is the heart of declarativity.

4.2 Text Descriptions

Again, consider the following text:

(34) John loves Mary

The particular segment *2-3* (i.e. the occurrence of "loves") can be described as a verb, as follows:[1]

(35) syncat(2-3,v).

Note the use of 'names' of points to explicitly refer to points in the text, and thereby to refer to segments delimited by these points. This then, should be compared with the practice of inserting labels at (or in relation to) these points, that was described in Chapter 1.

One might think that the notion of tagging, in the sense of 'labelling', is rather strange, if taken literally. One would certainly never dream of approaching the study of the (non-linguistic) physical world by attaching labels to physical objects. In physics, and in linguistics, it seems to be much more natural to start by *describing* the object under study, not just marking parts of it with labels. From this point of view, the TagLog approach is a *radical* alternative to the traditional approach, and it is also more in accordance with scientific practice in other fields.

Another possible way of looking at the 'Tagging in Logic' approach, and a way to make more sense of the 'Tagging by Labelling' approach, is to see the former as a *logical reconstruction* of the latter. The traditional corpus linguists have always treated the labels as a way to express statements

1. To say of the *string* "loves" that it is a verb is different, and could be expressed as follows: *syncat("loves",v).* For more on this, see Section 4.7.

about the text; they just haven't been very explicit about it, one might argue. This is how I choose to regard the 'Tagging by Labelling' approach in this thesis.

Just as traditional mark-up descriptions do, TagLog descriptions group entities into classes, the motivation being that the entities share common characteristics. For example, the segments denoted by *1-2* and *3-4* are grouped into a class denoted by 'pn', on the basis of the fact that both are names.

Segments may also be classified according to which string types they belong to:

(36) bstring(2-3,"loves").

Note that it is clear that it is not strings that are classified; rather *segments* are classified as instances of certain strings, just as they are classified as instances of certain word classes, such as *noun*, *verb*, etc. It would, of course, be possible to speak of "string tokens" instead of "segments", but that would obscure the fact that not only are segments instances of strings, but also of various other classes. Although it might be easier to see that a segment is an instance of "saw" than to see that it is an instance of (say) a noun, in principle these are descriptions on the same level. So why speak of them as "string tokens" rather than as "noun tokens"? Referring to them as segments avoids the problem.

A classification is perhaps the simplest, most primitive form of theory, but also the most basic. No theoretical work can be carried on without classification. In my view, not even the most radically quantitative empiricist approaches – such as clustering approaches to word classification (see e.g. Brown et al., 1990) – can do without classes and classifications. These approaches deal with string tokens (segments, that is), and it is characteristic of such tokens that the string class they belong to can be read off the tokens themselves – it is written on their back, so to speak. So the systems of classes that are generated in these approaches do not emerge out of raw text soup; they too are built on the basis of other classes.

A set of sentences such as these, classifying segments of a text, forms a description of the text, a kind of theory of the text.

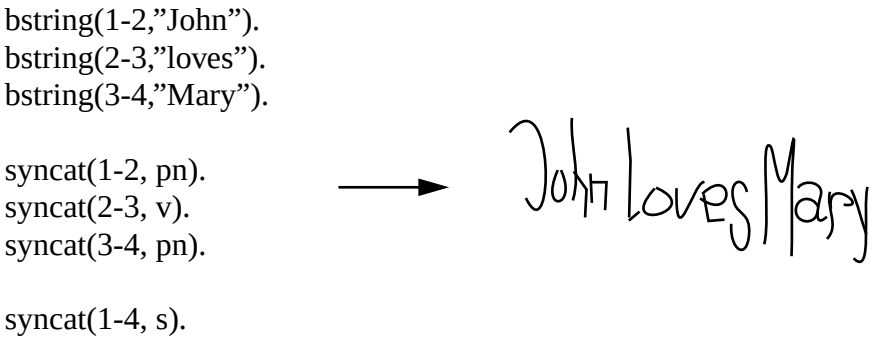


FIGURE 5. A text and its description

I have already made some comparisons with the mark-up descriptions discussed in Chapter 1. Let me continue to point out some similarities and some differences.

The point expressions making up a segment expression do not have to name adjacent points, and a name of a point may be used more than once, in different sentences. Thus, TagLog logic allows descriptions of segments longer than a word (e.g. the sentence *syncat(1-4,s)*), it allows simultaneous descriptions at several levels (e.g. *syncat*, *function*, *string*). It allows also descriptions of relations between segments, using such terms as *adjacent*, *embedded* and *overlapping*.

Together, the sentences in (37) do actually *claim*, not only that *1-2* is a proper noun and that *2-3* is verb, but also, since the second point of the first segment is identical to the first point of the second segment, that they are *adjacent* segments.

(37) syncat(1-2,pn).
syncat(2-3,v).

We know that we have succeeded in describing them as adjacent, since, with respect to the sentences in (37), the following theorem can be proved, with *S1* and *S2* bound to *1-2* and *2-3*, respectively:

(38) !- syncat(S1,pn), syncat(S2,v), adjacent(S1,S2).

In TagLog, there is no doubt as to what we mean by assigning two tags to one segment. It is absolutely clear that we must *not* think about that as forming a disjunction. The sentences in (39) claim that the segment *2-3* is a noun *and* a verb.

(39)
```
syncat(2-3,n).
syncat(2-3,v).
```

If we want to describe that segment as a noun *or* a verb, then we cannot, since this is something that the language does not allow.[2] We will strongly resist any temptation to 'hack' disjunction (e.g. by 'keeping in mind' that 'under certain circumstances' something like (39) 'really' means noun *or* verb). I certainly have nothing against the use of disjunction, but to use it, we will have to change the language first. Without disjunction, what we can try to show is that there is really no need for it, and that phenomena such as uncertainty and ambiguity can somehow be dealt with anyway.

All in all, it seems that we have at least the kind of expressive power that we expect from a mark-up language, except that it is much clearer that this *is* a language, and that it is a language with a well-defined meaning and an explicit inference method.

So far, we have only used this language to describe simple one-sentence texts. In this thesis, I often use examples that are deliberately oversimplified, so that major ideas can be presented in a simple way. However, just to show that we are not limited to 'laboratory sentences', made-up texts, and 'toy theories', let me give a preview of the TagLog system in action with one of the texts from the Brown corpus (Francis & Kučera, 1982) loaded in one window, and descriptions of this text lined up in the windows beside it.

---

2. The formal specification of the TagLog formalism on page 48 may have given the impression that disjunction *is* allowed. Indeed, disjunction is allowed, but only in a query or in the antecedent of an implication. Thus, we cannot say about a segment that it is a noun *or* a verb, but we can say that *if* it is a noun *or* a verb, *then* it is something else.

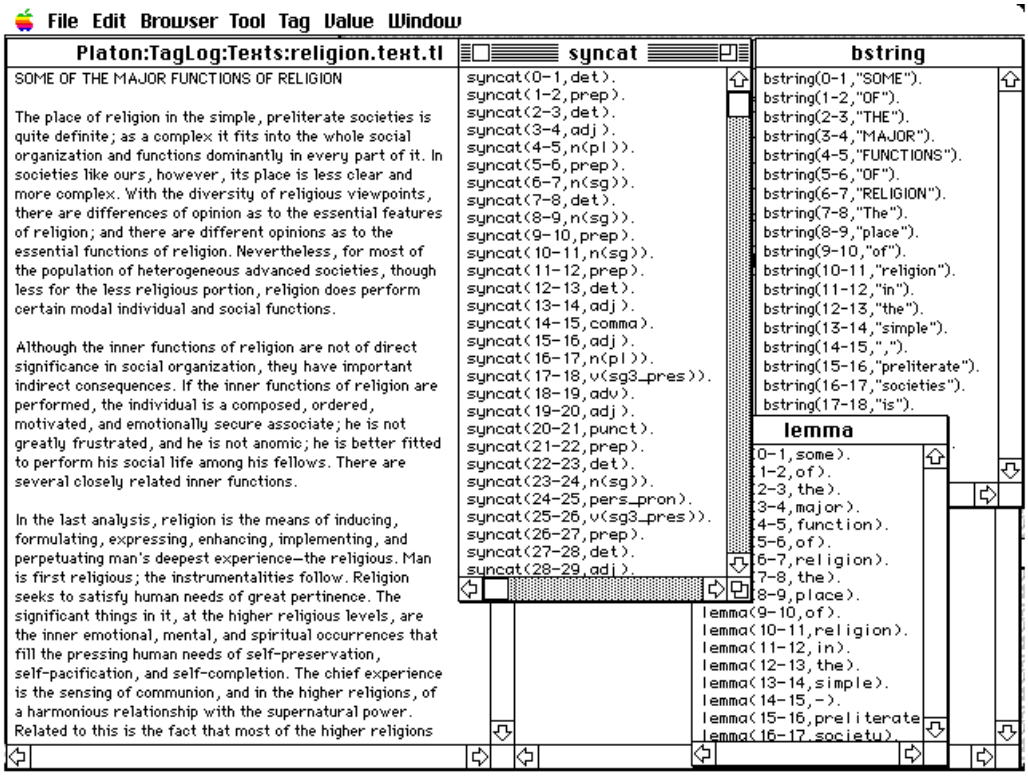


FIGURE 6. Text and theory in the TagLog system

As can be seen in Figure 6, TagLog ‘labels’ may be general terms, and thus they may be structured. To take a very simple example, to mark the proper nouns as singular, we introduce a *number* feature, and a value *sg* (for *singular*).

(40) syncat(1-2,pn(sg)).

Moreover, by means of general terms it is possible to represent *syntactic analysis trees*:

```
(41)    syncat(1-4,s)/
                (syncat(1-2,np)/
                        syncat(1-2,pn)/true,
                syncat(2-4,vp)/
                        (syncat(2-3,v)/true,
                        syncat(3-4,np)/
                                syncat(3-4,pn)/true)
```

In Chapter 10, I will introduce a couple of extensions to the expressive means of the basic TagLog formalism. Disjunction will not be among them, but (a limited form of) ‘classical’ negation will. Thus the extended TagLog formalism will allow us to express negative statements about seg-

ments. We will then be able to say, for example, about a segment, that it is not an adverb.

(42) ¬ syncat(1-2,adv).

To explicitly state that a segment is not an adverb is different from just leaving that information out. Thus, a clear distinction between negative and unknown facts is drawn. More importantly, the possibility of *contradiction*, i.e. the simultaneous derivation of a statement and its negation, is recognized. (Another extension that will be introduced is a certain capability for meta-description and reasoning, and a check for consistency will be introduced as part of this.)

The formulas of TagLog seem hard to read and write in comparison with mark-up languages. But the mistake is to think that we *need* to make a compromise between expressive power on the one hand and readability and writeability on the other. A set of TagLog clauses may be unsatisfactory from a presentational point of view and terrible from an entering point of view, but it is easy to see that there are ways to map the information contained in a set of clauses to an adequate presentation, and that there are other ways to enter clauses than to write them down, character by character, in a text editor. Later chapters will describe, in more detail, the TagLog implementation of some of these ideas.

4.3 Situation Descriptions

It is illustrative to compare descriptions of texts in the sense of Section 4.2 with descriptions of other things, other parts of the world. Also, it can be regarded as a useful preparatory exercise, because in Section 4.8, I will present a framework in which a description of a text and descriptions of the situation(s) that the text is relating to are integrated.

I will call such a theory, describing a situation, a *situation theory.* When the theory consists of singular statements only, it will be referred to as a *situation description.*

As a very simple and concrete example, consider how the clauses on the left in Figure 7 describe the particular situation depicted on the right.

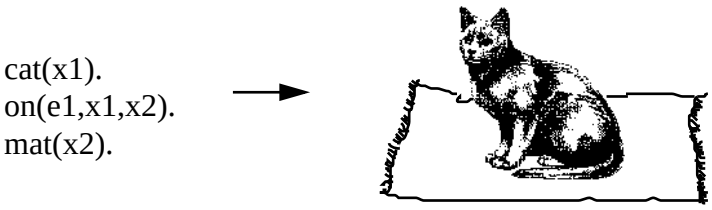


FIGURE 7. A situation theory and a situation

There are two ‘objects’, a cat and a mat, and there is an on-eventuality (or ‘state’, or ‘situation’) that they take part in.

It is not hard to see the similarities between a theory about a text and a theory about a situation in the world. There are names/identifiers referring to objects/segments, there are singular statements describing states of affairs, and so on.

4.4 Taxonomies

Up to this point, we have been concerned only with sets of singular statements, i.e. with *descriptions*. In the following, we will concentrate on *universal theories*, i.e. on sets of universal statements.

One form of classification that we often meet with in the traditional literature of grammar is the one between *open* and *closed* word classes, i.e between word classes that can be added to, and word classes which never (or very seldom) get any new members. In TagLog, classification of a segment as belonging to an open word class can be done as follows:

(43) syncat(2-3,open).

It is common practice to arrange classes in a *class hierarchy*, or a *taxonomy*. A taxonomy can be displayed graphically, as a tree (or, more generally, a graph). A very simple taxonomy of word classes can be displayed as in Figure 8.

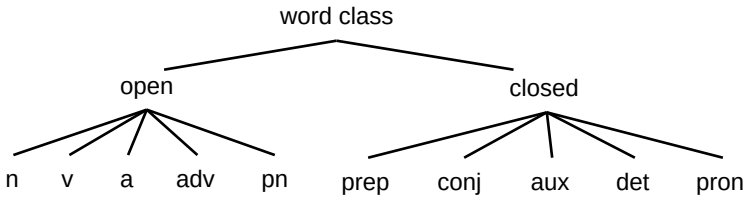


FIGURE 8. A taxonomy of word classes

It is well-known that a link in such a graph can be rendered in logical terms as a universally quantified implication. In Horn clause logic, and in case of the link between *v* and *open* above, it may look as follows:

```
(44)    syncat(S,open) :- syncat(S,v).
```

Now, it suffices to describe a segment as in (35); the sentence in (43) will follow by deduction.

The whole taxonomy of word classes in Figure 8 may be modelled as follows:[3]

```
(45)    syncat(S,open) :-
              syncat(S,n)
            ; syncat(S,v)
            ; syncat(S,a)
            ; syncat(S,adv)
            ; syncat(S,pn).

(46)    syncat(S,closed) :-
              syncat(S,prep)
            ; syncat(S,conj)
            ; syncat(S,aux)
            ; syncat(S,det)
            ; syncat(S,pron).
```

Note that the distinction between closed and open word classes is only one possible distinction out of many related ones.[4] Indeed, the user of the TagLog system is free to encode in a TagLog theory any distinction and any hierarchy of classes that he or she thinks would be interesting and/or

---

3. The reader is reminded that the character ';' is the TagLog disjunction symbol, so that the first of these rules should be read: "for every segment *S* holds that if *S* is an instance of a noun *or* verb *or* adjective *or* adverb *or* name, then *S* is an instance of an open word class".

4. A related distinction is the one between *content words* and *function words*, or between *categorematic* and *syncategorematic* words.

useful to have as working parts of the developing theories of text. No such distinctions or taxonomies are hard-wired into the system.

4.5 Definitions

As has already been mentioned, universal statements can be used to capture conceptual relationships in a domain. Since there are important conceptual relations holding between some of the built-in predicates of TagLog, in the sense that some of them can be defined in terms of others, we use them to illustrate this use of universal statements. Also, these are clear cases of conceptual relations. Note that we do *not* need to actually submit these definitions to the system, since they are already part of what the system knows. In fact, the system would not even allow us to, since they are built-in predicates and therefore cannot be redefined by the user.

One of the most important relations holding (or not holding) between three segments is the *concatenation* relation. The TagLog predicate *segment_concat/3* can be defined simply as:[5]

(47) segment_concat(P0-Pi,Pi-Pn,P0-Pn).

The *prefix* relation and the *suffix* relation can be defined in terms of *segment_concat/3*:

(48) segment_prefix(S3,S1) :- segment_concat(S1,S2,S3).

(49) segment_suffix(S3,S2) :- segment_concat(S1,S2,S3).

A subsegment of a segment can be defined as a prefix of a suffix of that segment (or, alternatively, as a suffix of a prefix of that segment):

```
(50)  subsegment(S1,S2) :-
                segment_suffix(S1,S3),
                segment_prefix(S3,S2).
```

5. To allow different instantiation patterns, the actual system definition of *segment_concat/3* is actually slightly more complicated.

4.6 Grammars

One particularly important kind of universal theory in TagLog is a *grammar*, although I hasten to add that I do not consider the distinction between grammars and non-grammar theories to be of any fundamental importance. As a matter of fact, one point that I would like to make here – and that point will be returned to in the chapters to come – is that theories of texts (of which a grammar is an example) and theories of situations (conceptual models, world models etc.) are very similar. Moreover, we can do very similar things in them: description, prediction, explanation, hypothesis testing, and so on.

4.6.1 Logic grammars

TagLog grammars are just one particular form of TagLog theories. Grammar 'rules' are not really rules, but rather universal statements expressed in logic. The term *logic grammar* has come to refer to the use of logic to formalize grammar rules. Pereira and Shieber (1987, p. 3) explain the basic idea as follows:

> [Phrase structure] rules have a very simple expression in first-order logic:
>
> $(\forall u,v,w)\ NP(u) \wedge VP(v) \wedge conc(u,v,w) \Rightarrow S(w)$
>
> where *NP* represents the class of noun phrases, *VP* the class of verb phrases, *S* the class of sentences, and *conc* holds of any string *u*, *v* and *w* such that *w* is *u* followed by *v*, that is, the concatenation of *u* and *v*. This expression in first-order logic thus states that any noun phrase *u* and any verb phrase *v* can be concatenated to form a declarative sentence *w=uv*.

In TagLog, then, a rule like S --> NP VP can be given the formulation

```
(51)    syncat(S,s) :- syncat(S1,np), syncat(S2,vp), segment_concat(S1,S2,S).
```

which is equivalent to (and can be partially evaluated to) the formula

```
(52)    syncat(P0-Pn,s) :- syncat(P0-Pi,np), syncat(Pi-Pn,vp).
```

Such a rule translates into English as "every instance of the syntactical category noun phrase followed by an instance of the category verb phrase is an instance of a sentence". The reading "a sentence can consist of a noun phrase followed by a verb phrase" is implied.

As mentioned in Chapter 3, a special purpose *grammar formalism* can be used instead of native TagLog syntax to express grammar rules. It is vital to realize that the grammar formalism is only a notational variant of ordinary TagLog. The mapping is simple: the universal statement in (52) can be written as in (53).

(53) `s --> np, vp.`

4.6.2 A grammar as an axiomatic system

The term *formal grammar* usually refers to the uses of formal techniques of logic and mathematics in the analysis of language. One particularly ambitious version of formal grammar is an axiomatic system in the sense of Chapter 3. An *axiomatic grammar*, then, consists of three parts: a language, a finite set of axioms (or axiom schemata), and a finite set of inference rules.

One way to see a grammar as an axiomatic system is to regard grammar rule statements, along with singular statements about particular segments of text, as axioms expressed in the language of Horn clause logic, and to repeatedly use just one rule of inference – the resolution principle – to derive other claims about the belonging of particular segments of text to particular grammatical categories, in the form of theorems.

4.6.3 What is a TagLog grammar about?

We have come to regard a grammar rule as a kind of statement, and a grammar as a theory consisting of a set of such statements. But if a grammar rule is a statement, what is it a statement *about*? Again, to a great extent, this is a matter of *intended interpretation.* That is, it is up to you and me to decide what grammar statements are about. Many grammarians, following Chomsky, have chosen to see grammars as descriptions (or models) of grammatical *competence*, or as descriptions of abstract, decontextualized entities called *sentences*. Logic grammar proponents have been no exception in this respect, but we do not have to buy that part of the story.

In this thesis I will try to defend the view of *grammars as universal theories of texts*; that is, as statements about segments (or about facts involving segments and grammatical categories) rather than about strings, about utterances rather than about sentences.

I believe there are several important methodological advantages to be gained by regarding logic grammars as empirical statements about the facts of linguistic reality. I also believe that certain difficulties will arise due to this view.

The most important advantage arising as a consequence of this view is that a theory of grammar becomes an 'ordinary' kind of empirical theory, no different from a theory of other (linguistic or non-linguistic) phenomena. First of all, that means grammar rules (and thus grammars) can be treated as true or false. Thus the grammarian's quest for correct grammars becomes a special case of the scientist's quest for true theories about the world.

Also, it means that what can be done in theories in general can be done in a grammatical theory. Standard methods and assumptions as we know them from other sciences apply. In particular, notions such as 'prediction', 'explanation' and 'hypothesis testing' become applicable and relevant.

A difficulty that might arise, however, is that as a consequence of the view of segments as the subject matter of corpus linguistics, *context* cannot easily be disregarded. Segments *qua* segments are *immersed* in context, and that's an *essential* fact about segments, and an inescapable fact about linguistic reality. Thus, grammars developed without regard for context, that is, most traditional grammars, might turn out to be *false*, if they are interpreted as statements about segments of text.

But then again, an advantage with grammars regarded as complex statements is that such problems may turn out to be solvable. Conditions can be logically strengthened, or weakened, and thus, the truth values of grammar rules can be changed. As we shall see, it is even the case that conditions referring to segments and linguistic categories – conditions on linguistic facts – can be interleaved with conditions referring to entities from other domains – conditions on extra-linguistic fact.

4.6.4 Grammars and grammar theories

One traditional distinction worth looking at is the distinction between descriptive grammar and *theoretical grammar*. Whereas a descriptive grammar tries to account for the facts of linguistic usage as they exist in particular languages, theoretical grammar goes beyond analysis of indi-

vidual languages to the theoretical foundations of such language, e.g. to explications of the notions of *category* and *rule*, the connection between lexical items and phrases, the relations between syntax and semantics, etc. In my view, it is better to conceive of this distinction as a distinction between *grammar* and *grammar theory*. (Of course, a grammar is also a theory, which makes grammar theories a kind of meta-theories – theories about (among other things) theories.)

TagLog is geared towards descriptive practice. Foundational 'claims' made by grammar theory are *assumed* but not explicitly *asserted* in particular TagLog grammars, and cannot be tested in the usual TagLog way. A false instance of a grammar rule does not necessarily prompt us to change our view of the structure of a particular category, for example. Also, needless to say, TagLog *as such* embodies certain claims of that sort, and thus the extent to which TagLog can function as a testbed for other such claims is clearly limited. However, one nice thing about TagLog, and something that contrasts it with most other tools, is that many of the structures and relations assumed by current schools of theoretical linguistics *can* be represented. In this way, TagLog can be seen as an attempt to strengthen the communication between the theoretical grammarians and descriptive grammarians.

In Section 4.6.5 – 4.6.7, I will give a brief overview of some of the kinds of grammars that can be written in the TagLog framework.

4.6.5 Phrase structure grammar

Basic in some respects to (at least generative) grammar is the pure context free phrase structure grammars: being fully recursive, they are perhaps the simplest devices that are still general enough to be interesting and relevant from a grammarian's point of view. In (54), a simple example is given.

(54)
```
syncat(P0-Pn,s) :- syncat(P0-Pi,np), syncat(Pi-Pn,vp).
syncat(P0-P1,np) :- syncat(P0-P1,pn).
syncat(P0-Pn,vp) :- syncat(P0-Pi,v), syncat(Pi-Pn,np).
```

In the grammar rule notation provided in TagLog, (54) can be expressed as (55):

(55)
```
s --> np, vp.
np --> pn.
vp --> v, np.
```

For example, given the ‘text’ in (56),

(56) John loves Mary

and given that *1-2*, *2-3*, and *3-4* denote the first, second and third word token, respectively, these universal statements, together with the statements

```
(57)  syncat(1-2,pn).
      syncat(2-3,v).
      syncat(3-4,pn).
```

form a small but still useful theory about syntax in which, for example (58) is a theorem.

```
(58)  syncat(1-4,s)
```

4.6.6 Categorial grammar

An interesting alternative to phrase structure grammar is *categorial grammar* (Lambek, 1958; Woods, 1993). In contrast to phrase structure grammars, categorial grammars are *radically lexicalistic*: nearly all grammatical information is encoded in lexical categories, and only a very small number of rules (or ‘cancellation schemata’) are needed.

The two categorial grammar rules in (59), and the utterance in (56) tagged with lexical categories as in (60), allow the derivation in (61).

```
(59)  X --> X/Y, Y.
      X --> Y, X\Y.

(60)  syncat(1-2,np).
      syncat(2-3,(s\np)/np).
      syncat(3-4,np).

(61)  !- syncat(1-4,Sign).
            Sign = s
```

4.6.7 Unification-based grammar

A trend arose a decade ago in theoretical and computational linguistics towards unification-based grammar formalisms – FUG, GPSG, LFG, DCG, UCG, PATR, HPSG – and is still with us today.[6] The TagLog sys-

6. Two good introductions to unification-based grammars are (Kay, 1992) and (Shieber, 1986).

tem contains a unification-based grammar system, similar in most respects to DCG. Although some of the other formalisms are based on feature-structures rather than terms, the first-order terms of Prolog (and TagLog) provide expressive power equivalent to feature structures, so as to ensure that most theoretical grammar models can be implemented (or at least partially emulated) in DCG, and therefore in TagLog. In this section we look at some examples of how different solutions to typical problems of grammar can be implemented in TagLog.

First, consider the grammar in (62) where the use of variables and unification guarantees *number agreement* between the subject and the verb. A description of the utterance in (56) as in (63) makes derivations like the one in (64) possible.

(62)
```
s --> np(Num), vp(Num).
np(Num) --> pn(Num).
vp(Num) --> v(Num), np(_).
```

(63)
```
syncat(1-2,pn(sing)).
syncat(2-3,v(sing)).
syncat(3-4,pn(sing)).
```

(64)
```
!- syncat(1-4,Syn).
      Syn = s
```

As a second example, the grammar in (65) implements a simple scheme for *verb subcategorization* (only non-subject complements are subcategorized for). Given (56) tagged as in (66), the derivation in (67) can be performed.

(65)
```
s --> np, vp.
np --> pn.

vp --> v(SubCat), vcomp(SubCat).
vcomp([]) --> [].
vcomp([X|Xs]) --> X, vcomp(Xs).
```

(66)
```
syncat(1-2,pn).
syncat(2-3,v([np])).
syncat(3-4,pn).
```

(67)
```
!- syncat(1-4,Syn).
      Syn = s
```

In our third example, we use a *gap threading* technique to handle *long-distance dependencies*, for instance, as it is introduced by the utterance of a WH-question in (68).

(68) Who does John love?

(69)
```
s(GapIn,GapOut) --> wh, s([np|GapIn],GapOut).
s(GapIn,GapOut) --> aux, np(GapIn,Gap), vp(Gap,GapOut).
np([np|Gap],Gap) --> [].
np(Gap,Gap) --> pn.
vp(GapIn,GapOut) --> v, np(GapIn,GapOut).
```

(70)
```
syncat(1-2,wh).
syncat(2-3,aux).
syncat(3-4,pn).
syncat(4-5,v).
```

(71)
```
!- syncat(1-5,C).
      C = s(X,X).
```

Next, the grammar in (72) implements a simple unification-based *compositional semantics*. The colon is used to separate the syntax from the semantics. The utterance in (56) described as in (73) allows the derivation in (74).

(72)
```
s:Sem --> np:X, vp:X^Sem.
np:Sem --> pn:Sem.
vp:Sem --> v:X^Sem, np:X.
```

(73)
```
syncat(1-2,pn: j).
syncat(2-3,v: Y^X^loves(X,Y)).
syncat(3-4,pn: m)
```

(74)
```
!- syncat(1-4,SynSem).
      SynSem = s: loves(j,m)
```

Finally, (75) gives the forward and backward application rules in a simple *unification-based categorial grammar* with compositional semantics. Here also, the colon is used to separate the syntax from the semantics. The description of (56) in (76) allows the derivation in (77).

(75)
```
C:S --> C/(C1:S1):S, C1:S1.
C:S --> C1:S1, C\(C1:S1):S.
```

(76)
```
syncat(1-2,np:j).
syncat(2-3,(s\(np:X))/(np:Y):loves(X,Y)).
syncat(3-4,np:m).
```

(77)
```
!- syncat(1-4,Sign).
      Sign = s:loves(j,m)
```

We see that the TagLog grammar formalism is quite expressive. More suggestions on how to solve particular problems in the area of syntax (and in the syntax-semantics interface), as well as information on how to straightforwardly combine the techniques for handling agreement, subcategorization, semantics and long-distance dependencies in one single

grammar, can be readily found in the literature on unification-based grammar (see for example Shieber, 1986; Pollard & Sag, 1995).

4.7 Lexica

Another common form of body of linguistic knowledge is the *lexicon*. In standard phrase structure grammar, for example, a lexicon might be written as follows:

(78) John: pn
loves: v
Mary: pn

On the face of it, this looks very similar to a segment of text that has been tagged with lexical categories. But the semantic difference is great, given the assumptions made so far. As we saw in Section 4.2, a tagged segment of text is 'logically reconstructed' as follows.

(79) syncat(1-2,pn).
syncat(2-3,v).
syncat(3-4,pn).

That is, (79) says of *particular* segments (referred to by the segment expressions) that they are proper nouns, verbs, etc. That is, it is important to realize that by the introduction of the sentence *syncat(2-3,v)* in the description we have said something about that particular *instance* of the word "loves". We have said nothing at all about the string "loves" *as such* (except, implicitly, that at least one of its instances is a verb).

A lexicon, on the other hand, makes much more general claims. For example, a lexical entry may say of a word that *all* of its instances are of a certain category. Thus, a logical reconstruction of a lexicon, if it is going to be able to account for the relation to tagged corpora, must involve universal statements. A lexicon is really a *universal* theory of (certain aspects) of text, with sentences quantifying over word occurrences.

Indeed, in the logic grammar tradition, lexical entries are usually encoded as universally quantified conditionals, roughly as in (80).

(80) syncat(S,pn) :- bstring(S,"John").
syncat(S,v) :- bstring(S,"loves").
syncat(S,pn) :- bstring(S,"Mary").

Lexical lookup is performed by deduction. Given an encoding of a string as follows:

(81) bstring(2-3,"loves").

the following sentence is derived as a theorem, and thus, the lexical category is tied to the particular segment in question.

(82) syncat(2-3,v).

It is tempting, and very much in accordance with current terminological practice, to say about a word, all the instances of which are of a certain category, that the word in *itself* is of that category. Thus, in the case of “loves”, if it is true that all of its instances are verbs (and for the sake of the example, let us assume that it is), we say that “loves” is a verb (and that “John” and “Mary” are proper nouns):

(83) lexicon(“John”,pn).
lexicon(“loves”,v).
lexicon(“Mary”,pn).

Then again, we must not let this kind of representation obscure the fact that a lexicon is a universal theory; and it is indeed the fact that for a lexicon of this kind to have any bearing on segments of text, and to capture the relation between (81) and (82), we need an axiom along the lines of (84), that is, a universally quantified sentence.

(84) syncat(S,C) :- bstring(S,W), lexicon(W,C).

So we see that with the ambition of attempting to account for the relation between part-of-speech tagged text and a lexicon, the logical structure of a lexicon is slightly more complicated than the list of pairs of words and lexical categories might suggest.

As soon as we realize that the logical structure of the lexicon really is universal, we might begin to see some further possibilities for experimentation: we might want to introduce other kinds of conditions, we might want to reverse the directions of the implication arrows, we might want to play around with negative information, with default versions of the logic, and so on.

For example, instead of formulating conditions on the belonging of segments to a certain lexical category in terms of the string classes to which the segments belong, we might want to introduce *lemmas* into our ontol-

ogy,[7] and condition the belonging of segments to a certain lexical category on lemmas rather than on strings.

(85) bstring(S,"saw") :- lemma(S,see1), tense(S,past).
syncat(S,v) :- lemma(S,see1).

This gives us a kind of lexicon which may be used to deduce *string* and *syncat* given knowledge of lemma, but also, as we will see in Chapter 13, may be used to *abductively* infer ('abduce') the lemma of a certain segment, given information on *string* and/or *syncat*.

Or by introducing meta-logical conditions on the consistency of the current theory, and by defining negative facts, a lexicon sensitive to the local co-text can be implemented.

(86) syncat(S,C) :- bstring(S,W), lexicon(W,C), consistent(syncat(S,C)).
¬ syncat(P1-P2, v) :- syncat(P0-P1,det).

This could perhaps be conceived of as a move towards a kind of lexicon the purpose of which is not only to passively serve the needs of the grammar, a 'smart' lexicon, that could reason about context. And/or, it could be seen as a move towards a 'word grammar', making no reference to any segment longer than a word.

Both of these approaches could conceivably be extended to include the kind of objects that typically appear within a lexicon, such as senses, definitions, antonyms, and so on. Regardless of which approach we eventually choose to follow, there is nothing very special about lexica; they too are just theories.

4.8 Syntax, Semantics, and the Modelling of Context

Semantics studies the relation between language and the world, between linguistic reality and extra-linguistic reality. For a grammar to 'handle' semantics, it must be able to relate a text to its semantic content, in virtue of the meanings of the expressions used and other facts of the text.

7. Or perhaps *lexemes*, or something else along that line.

4.8.1 A situation theory as a content/context model

At least for a narrative text, it makes sense to think of its content, and its situational context, in terms of situations described and presupposed. Thus for a narrative text, a suitable situation theory, in the sense explained in Section 4.3, may serve as a model of the content and context for the text.

As a very simple and concrete example, consider how the situation theory from Section 4.3 models the content of an utterance of "The cat is on the mat".

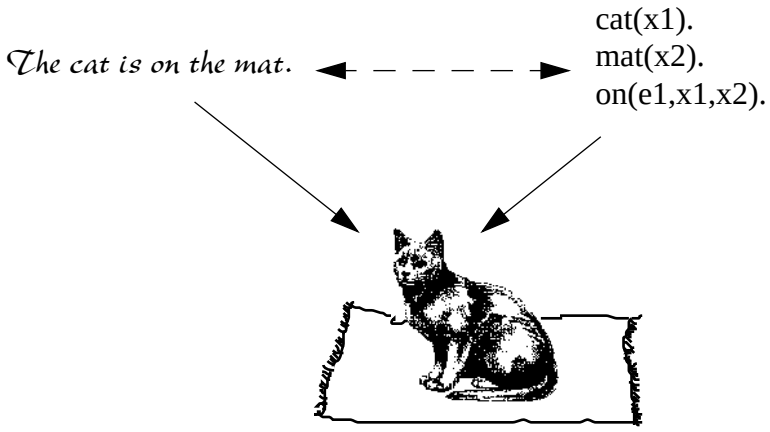


FIGURE 9. A text, a situation theory, and a situation

As can be seen in the diagram, the situation theory can be characterized as a *translation* of the natural language text, expressed by a logical language.[8]

4.8.2 Integrated grammar

A perhaps very desirable form of grammar would generalize over both form *and* content *and* context, and would thus integrate syntax, semantics, and pragmatics. What would such a grammar look like?

Hobbs et al. (1993) points to a way of performing a thorough integration of syntax, semantics, and pragmatics, by combining the idea of *interpre-*

8. Note that I make no distinction between content and (situational) context (that is, I do not put them in different boxes). This is natural, since we are not concerned here with the dynamics of text comprehension, and since the content of a particular utterance will typically serve as the context for the utterances that follow. Hans Kamp refers to this as the 'principle of the unity of content and context' (Kamp, 1985).

tation as abduction with the idea of *parsing as deduction* (Kowalski, 1979; Pereira & Warren, 1980).[9] The axioms of a logic grammar are augmented with predications referring to content and context in appropriate places.

In this section, we will concentrate on the declarative aspect of the integrated approach. We will return to the use of abduction, and to interpretation as abduction, in Chapter 13.

Consider the following grammar:

(87)
```
s:E --> np:X, vp:X^E^A, {provable(A)}.
np:X --> det, n:X^A, {provable(A)}.
vp:A --> cop, pp:A.
pp:A --> p:X^A, np:X.
```

Again the colon is used to separate the syntax from the semantics. The meta-predicate *provable/1* is true if whatever sentence *A* gets bound to is provable in the current situation theory.

Let us translate some of the rules of this grammar into logic and see what they tell us. The statement in (88) says that if we have a noun phrase from *P0* to *Pi* referring to *X* and a verb phrase from *Pi* to *Pn* denoting *X^E^A*, if there is an eventuality *E* such that *A* is true of *E* and *X*, then there is an interpretable utterance from *P0* to *Pn* describing eventuality *E*.

(88)
```
syncat(P0-Pn,s:E) :-
        syncat(P0-Pi,np:X),
        syncat(Pi-Pn,vp:X^E^A),
        provable(A).
```

The statement in (89) says that if *P0-P1* is an instance of a determiner,[10] and *P1-P2* is a noun denoting *X^A*, and *A* is true of X, then the segment *P0-P2* is an instance of a noun phrase referring to *X*.

(89)
```
syncat(P0-P2,np:X) :-
        syncat(P0-P1,det),
        syncat(P1-P2,n:X^A),
        provable(A).
```

---

9. In particular, see section 6 in Hobbs et al. In fact, they attribute this idea to Stuart Shieber.

10. The fact that the determiner does not get any semantics here is of course a gross oversimplification. Decades of research in formal semantics has shown that determiners have a lot of very important semantics relating to quantification.

Thus, these rules categorize segments of text on the basis of how their 'constituent' subsegments are categorized, how the semantics of the categories combine, and how the result of evaluating the result of semantic processing in the content/context model comes out.

In Figure 10, we have put the pieces together: a grammar, a text, and a situation theory.

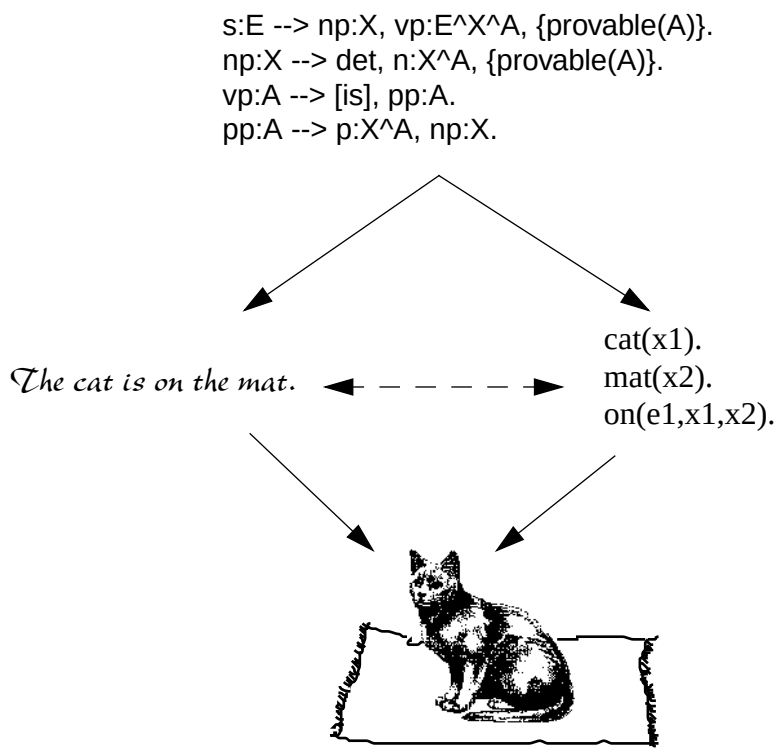


FIGURE 10. Integrated grammar

We will now be able to perform the proof in (90).

(90) !- prove(syncat(1-7,C)).
C = s:e1

Thus, we are able to deduce the fact that the text segment *1-7* is a sentence denoting an eventuality *e1*, of which we know that it is an *on*-eventuality in which a cat and a mat participate. We might *explain* the conclusion as follows. The segment corresponding to "The cat is on the mat" is a sentence denoting eventuality *e1*, since the segment corresponding to "The cat" is noun phrase denoting *x1* and the segment corresponding to "is on the mat" is a verb phrase denoting *X^E^on(E,X,x2)*, and *on(e1,x1,x2)* is true. The instance of "The cat" is a noun phrase denoting

x1, since “The” is a determiner, and “cat” is a noun denoting *X^cat(X)*, and *cat(x1)* is true. And so on...

A few remarks are in order here. First, the reader should be warned against seeing this in terms of interpretation processes. All we are doing here is trying to describe, on a general level, a relation between a text and its content and context, when all these factors are known in advance. The problem of determining the content, when the content is not known, is a different problem, to which we will turn in due time.

Still, the above theory seems to capture the following facts about the linguistic reality:

- The correct grammatical classification (*s* in the above example) of a text segment may depend on the content and context of that segment.
- The content of a text segment (*e1* and the associated information in the above example) may depend on its context. This context, in turn, may include the content corresponding to surrounding (in particular preceding) segments.

So far, we have really only discussed sentences, not texts. A *discourse grammar* is in many respects like a sentence grammar, but it crosses the sentence boundaries, and connects sentences into discourses. Once again, let me borrow a small example from Hobbs et al.:

(91)
```
discourse:E --> s:E.
discourse:E --> discourse:E1, discourse:E2, {coherence_rel(E1,E2,E)}.
```

The rules in (91) claim that a sentence with the interpretation *E* is a discourse with the same interpretation, and that two adjacent discourses form a larger discourse if they stand in some kind of coherence relation (such as explanation, cause, motive, etc.).

4.8.3 Now, what’s the point of all this?

There are at least three reasons why one would like to use integrated grammar in the manner sketched here. First, if theories about text are extended to content and context, then the scope of the tools that employ these theories are extended to content and context as well. As will be seen, TagLog tools can indeed be used to investigate the relation between form, content, and context, of text.

Secondly, it may well possibly be that including descriptions of content and context in our theories is the only way (or at least the most 'correct' way) in which the properties of text segments can be successfully predicted, and the facts of linguistic reality correctly explained. Some observations that we make in Chapter 14 tend to indicate that this is indeed the case.

Thirdly, on the side of usefulness, determining the content of text is also the key to the success of many potential natural language processing applications. The purpose of a message understanding system, for example, is to map text into some representation of its content.

So much for increase of coverage, truthlikeness, and usefulness. Are there any reasons why we should *not* use integrated grammar? Yes, for the average corpus linguist, there may be many such reasons. Constructing an integrated grammar for even a very small text is a very difficult and time consuming work. Integrated grammars are something to strive for, but in practice, for most tasks, we will probably have to be content with less. A simple syntax grammar (a couple of context free rules, say) can also count as a corpus grammar, although one must not forget that it describes only a certain *aspect* of linguistic reality – a text cut off from its context – and only certain properties of this text that relate to its form and to the ordering of its elements. In other words, it tries to describe text-in-context, without really relating to this very context. As we will see in Chapter 14, this might lead to problems.

4.9 Dialogue Grammars

Grammarians working with sentences are certainly not the only linguists that have been attracted by context free grammars, rewriting systems and the like. Some researchers have claimed that the structure of dialogue too can be fruitfully described with the help of such devices. These grammars are usually called *dialogue grammars*.

Some of the attempts to construct such grammars are based upon the concept of *adjacency pair*. This concept is one of the corner stones of the *conversation analysts*, the school of sociology that for the last couple of decades has investigated recurring patterns in natural conversation, on the

basis of recordings and transcriptions of such conversations (Levinson, 1983).

Adjacency pairs are sometimes claimed to be the fundamental units of conversational organization (cf. Coulthard, 1977, p. 70). A typical adjacency pair consists of a greeting followed by another greeting,

(92) A: Hi there!
B: Hello

a question followed by an answer,

(93) A: What's your name?
B: My name is Hillevi

or an offer followed by an acceptance, a summons followed by a response, etc.

Using the TagLog grammar formalism, such interactions can be described as follows, with just a few simple rules:

(94)
```
adj_pair --> greeting, greeting.
adj_pair --> question, answer.
```

However, as noted by for example Levinson (1983, p. 304), the requirement of strict adjacency of the pair is too strong, and *insertion sequences* often occur where, for instance, one question-answer pair is nested within another, or where the level of nesting can be even deeper, as in the following example:

(95) A: Can I speak to your daddy?
B: Who's asking?
A: Is he at home?
B: Yes
A: Is he working?
B: Yes, as usual
A: This is his boss
B: Okay, hold on a second

For capturing these kinds of dialogue structure, we will adapt a dialogue grammar from (Gilbert, Wooffitt, & Fraser, 1990):

(96)
```
adj_pair --> greeting, greeting.
adj_pair --> question, answer.
adj_pair --> summons, response.
adj_pair --> offer, acceptance.
```

```
adj_pair --> question, insertions, answer.
adj_pair --> offer, insertions, acceptance.

insertions --> insertion, insertions.
insertions --> insertion.
insertion --> question, insertions, answer.
insertion --> question, answer.
```

As will be demonstrated, such a system of rules makes it possible to search for instances of adjacency pairs, to count them, to make concordances of them, to explain and to predict.

4.10 Summary and Conclusion

In this chapter, the expressive power of the language defined in Chapter 3 was explored. Data description, the TagLog alternative to mark-up, was exemplified, and some comparisons with mark-up were made. Also, a description of a particular text was compared with a description of a particular (non-text) situation.

Combining the notions of ‘logic grammar’, ‘descriptive grammar’, and ‘corpus data’, we arrived at the notion of ‘logic grammar for describing corpus data’. It was shown that many different approaches to grammar theory can in principle be supported by TagLog. We also noted that the TagLog notion of grammar is indeed very broad. In the broadest possible case, a grammar is a theory, not only of text, but also of the *relation* between text, its content, and its context. Furthermore, an approach to the logic of the lexicon was suggested, in which the relation of the lexicon to descriptions of particular texts is seen as just as important as the relation to the (rest of the) grammar.

It is extremely important to emphasize that regardless of whether they are grammars, lexica, situation theories, or text descriptions, these are all theories. This lends a good deal of uniformity to the approach. Finally, it is important to emphasize again that these theories are nothing but *examples* of what may happen when the expressiveness of the TagLog formalism meets the creativeness of the corpus linguist.

CHAPTER 5

TagLog System Overview

5.1 Introduction

In this chapter, the design of the TagLog system – a computer program implementing the logical approach to corpus linguistics and supporting the activity of corpus linguistics – will be discussed.

The TagLog system is a scientific instrument, and by definition, science is about exploration and creativity. Thus, TagLog can be classified as what Shneiderman (1992) calls an 'exploratory, creative system'. This should perhaps not be taken to imply that the system in itself is creative, or that it is able to explore anything by itself, but rather that it is able to take care of some of the non-creative tasks of science – storing, counting, searching – in order to create an environment to support the user's own acts of exploration and creativity.

This chapter also presents the Macintosh implementation of the TagLog system used in this thesis to illustrate the logical approach to corpus linguistics. This is just an overview, however, and more detailed descriptions of the TagLog analysis tools are given throughout the thesis, in connec-

tion with the sections that describe the theory behind the tools, and that give examples of their use.[1]

TagLog has been *designed*, in the sense that due consideration has been given to the principles and rules of thumb that always apply to the design of software, independently of its domain of application (cf. Shneiderman 1992). We will touch upon this slightly.

In this chapter, we will also spend a few pages looking away from the particular implementation of TagLog that we use to exemplify the approach. In particular, we will be looking at questions of architecture and efficiency at a more general level.

5.2 TagLog for Interactive Theory Revision

The TagLog system is designed as a system for *interactive theory revision*. I will explain in this section what this means, and I will contrast this design with a more traditional design of corpus tools. In particular, I will contrast it with what Sinclair (1992) has to say about his "strategy for planning and designing the next generation of analytical software" (cf. Section 1.2.2). I have chosen Sinclair here, because his views are so different from mine.

5.2.1 TagLog is a theory revision system

I have devised TagLog as a theory revision system. It is nearly always the case that the researcher approaches a corpus with some kind of theory in his luggage, obtained from a textbook, from interviews with an expert, from his own intuitions, etc. It is usually the case that this theory does not quite fit the data. It may be false, it may lack coverage, it may be too general, or too specific. The problem then is to revise this imperfect theory to make it fit the facts under study. This, indeed, is a theory revision problem, and therefore I have devised TagLog as a theory revision system.

1. The current implementation of the system is not complete with respect to the specification put forward in this thesis. The functionalities involving abduction, situation theories, and automatic tagging have been implemented, but have not yet been fully integrated with the rest of the system. Consequently, some figures showing the system in action have been fabricated.

5.2.2 TagLog is an interactive system

I have devised TagLog as an interactive system. Humans and computers are good at different things. Corpus linguistics involves searching, observation, discovery, counting, understanding, storing, and many other things. All this interacts in complicated ways, that simply do not allow implementations as programs to be run in batches. Therefore, TagLog does not try to replace human intelligence or expertise. Rather, it applies computer intelligence in ways that seek to bind the user and the system closer together. In the TagLog approach, the computer focuses on those tasks at which it excels. Observation, understanding, and creative imagination is left to the human expert.

Contrast this with Sinclair's view: "analysis should be restricted to what the machine can do without human checking, or intervention". It is understandable that Sinclair takes this position: he is interested mainly in statistical methods for corpus work; statistical methods usually demand very large corpora (in Sinclair's words: "operations should be designed to cope with unlimited quantities of text material"); we don't want to do manual work on very large corpora, it is far too slow ("speed of processing should take precedence over ultimate precision of analysis").[2]

This idea of corpus work as a one-shot process where the user prepares a corpus and the required knowledge and then runs an algorithm on the data (e.g. an algorithm for part-of-speech tagging) is very different from the view of corpus work that I am arguing for here. In my view, interaction with the human expert is necessary, and should be supported.

This does *not* mean that no support is given for automatic processing. On the contrary, as will be seen, TagLog implements several automatic tagging tools, and machine learning tools, that perform parts of the work automatically. However, I am aiming at a design in which these automatic processes always inform the user what they are doing, and report the results of what they are doing as soon as such results become available.

---

2. Normally, software unguided by a human expert will make lots of errors. Tagging errors and parsing errors abound. Sinclair's solution: "Software will be designed to operate at more than one level of discrimination, so as to bypass doubtful decisions." I am advocating a 'bolder' approach in this thesis. Let errors arise. Let us observe them. Let us take care of them by changing the theories. After all, this is what makes theories change into better theories.

This is feedback. And the user may always interrupt, break in, and add or delete a couple of clauses, to which the system should always be prepared to accommodate. This is incrementality. But if something goes wrong, the system should always be prepared to back up and undo. This is reversibility. Feedback, incrementality, and reversibility – that's the essence of what it means to be an interactive system.

5.2.3 TagLog does this, user does that

Given that we do divide the work between the user and the system, *how* do we divide it? What can a system do all by itself, what can it support, and what must be left entirely to the user?

In the following, in order to say something about what is supported by TagLog, and what is not, I will make use of the division between the three kinds of activities from Section 2.5.1, and the analysis of the dynamics of these activities, that were provided in Section 2.5.2. There I distinguished development of descriptions of particular texts from development of descriptive universal theories of (sets of) texts, and from development of theoretical principles. Furthermore I distinguished, with the help of Popper, a theory development cycle consisting of problem detection, proposal of tentative solution, and error elimination.

TagLog supports balanced cooperative development of descriptions of particular texts – a classification of (parts of) text data according to a classification schema. Often, it is up to the user to do the observation and the classification, and it is the responsibility of the system to present the user with the text in a way that supports observation. Also, it is up to the system to check the user-supplied descriptions for consistency, and to remember them. But sometimes, as we have already indicated, TagLog is able to perform automatic classification. This does not mean that the system 'takes over'. The user may very well break in and add a clause or two by hand, and this new information is to be taken into consideration by the system.

Note that classification of particular segments of text presupposes the existence of a system of classes. The development of appropriate systems of classes and taxonomies is indeed a difficult and important theoretical meta-level task. As the system now stands, this task has been left entirely

to the user. In a future implementation, the system might be able to help the user with this too.

What about the development of descriptive universal theories? In support of the development of *quantitative* descriptions, based on the classifications, TagLog implements counting, and tabling, and various sorts of statistical calculations. But above all, the TagLog system is designed to support development of explicit representations of *qualitative*, descriptive theories. First of all, with respect to the development cycle as given above, there are tools in TagLog that may assist the researcher in finding facts that are not covered by a certain theory. That is, tools to identify some of the problems that may be there.

In general, the initial 'conjecture' concerning a particular linguistic phenomenon as it occurs in a text is a truly creative part of scientific work, even when it comes to descriptive theories. Nevertheless, this aspect of scientific work can be supported, in at least an indirect way, by a system providing means of *information exploring*. Efficient information exploration has two important requirements: quick ways of navigating through data, and powerful means of visualizing data and analyses. TagLog supports both, for both texts and theories.

The TagLog *Search Tool* and the *Concordance Tool* provide means of exploration of the texts under study. These tools are even able to use the theory under development to support navigation. Presentation (visualization) of search results is done in various ways.

To support exploration of theories under development, there are browsers that implement various aids for navigation in theories, and various modes of presentation of theories. The generation and presentation of explanations are supported as well.

Some aspects of the theory testing activity have been taken over by the TagLog system. The system handles the search for confirming or falsifying instances of universal statements, or the generation of predictions given such statements, followed by the checking of predictions against data, in order to confirm or disconfirm the predictions.

TagLog is very open to the incorporation of tools that implement methods for learning (certain aspects of) descriptive theories automatically. I will present some such methods further on in this thesis.

On the other hand, the task of discovering and formulating theoretical principles (in contrast to ‘shallow’ and ‘low-level’ descriptive theories of data) is a truly creative part of scientific work. It is extremely unlikely that we will ever be able to provide a system with tools capable of doing much useful work here. In this, we should let ourselves be guided by a general ‘quest for truth’ rather than by tools. But one thing seems clear: to discover, one must explore. Hence, tools for exploration will be useful in this context too.

The division of labour between the user and the system is summed up in Table 1.

User	**System**
Navigating linguistic reality	Providing the means of navigating linguistic reality
Observing facts	Presenting facts
Describing facts	Remembering descriptions
Summarizing data	Providing the means of summarizing data
Questioning statements	Explaining statements
Testing statements	Detecting inconsistencies and other errors

TABLE 1. Division of labour between user and system

5.3 Against Tools!

Now let us turn to a discussion of *tools*. To begin with, let us think of different tools as corresponding to different tasks within the process of developing and revising a description of a text.

In Chapter 1, we saw Sinclair (1992) give an impressive-looking list of tools believed by him to be needed by the corpus researcher: a parser, a boundary marker, a tagger, a collocator, a lemmatizer, a compounder, a disambiguator, a lexical parser, a phrase finder, an exemplifier, a setter, a classifier, a typologizer, etc.

Many of the tools in this collection (as well as many of the other tools listed in Chapter 1) have the general architecture depicted in Figure 11.

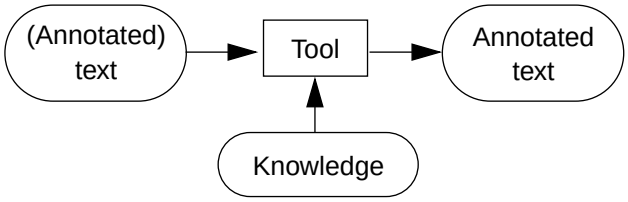


FIGURE 11. From text to text with traditional corpus tools

That is, tools map (possibly annotated) text into annotated text. In order to perform this mapping, the tool needs to be fed some kind of 'knowledge'. A parser needs to be fed a grammar. A tagger needs lexica, transition probabilities, rules, or constraints.[3]

As the picture indicates, no clear separation between the texts and the descriptions of the texts (i.e. the annotations) is made in traditional systems. Tagging (or parsing, or lemmatizing) means *changing* a text, by annotating it. In TagLog, by contrast, tagging, syntactic analysis, etc. means *changing what follows from the theory describing the text*. The text is not changed at all. As a matter of fact, the text is not even editable.[4]

In contrast to traditional systems, TagLog has very few tools, but can still realize very many functionalities. The most important tool is a *theorem prover*. This is a very general tool. Throughout this thesis, we will see this theorem prover work as a parser, as a tagger, as a search engine, as a concordancer, a lemmatizer, etc. Apart from this, there are very few tools.

---

3. Figure 11 is slightly misleading in suggesting that the 'knowledge' used by this kind of tool is clearly separated from the tool itself. Granted, the 'knowledge' is usually stored separately, but an *understanding* of what the 'knowledge' means can usually only be gained in relation to how it effects the behaviour of this particular tool. As will be seen, it is different with TagLog: the semantics of the TagLog formalism is independent from any processing considerations.

4. Hence TagLog, but not the traditional approach, *guarantees* adherence to the principle advocated by Leech (1991) in the following:

> [A]n annotated corpus should never totally replace the corpus as it existed prior to annotation. The original 'raw' corpus [...] should always be available, so that those who find the annotations useless or worse can recover the text in its virgin purity.

TagLog does, however, have tools in a different sense, on a different level, as well. TagLog ‘tools’ are more on the level of update tools, browsers, query interfaces, and presentation tools. In fact, most TagLog ‘tools’ are not absolutely essential; they are there for the sake of user friendliness only. In principle, we could call the theorem prover directly, from a command line interface, in order to parse, search, tag, or whatever.

In the TagLog approach we prefer to think in terms of *theories* rather than in terms of tools. TagLog has no separate parser, no phrase finder, no collocator, no tagger, no lemmatizer. Instead, different functionalities are realized by means of the same general tool – a theorem prover – either by using different sub-theories (e.g. a grammar for parsing; a lexicon for part-of-speech tagging), or by using the same body of knowledge in different ‘directions’, with different variables uninstantiated (e.g. a grammar can be used for *searching* for grammatical constructions, as well as for parsing).

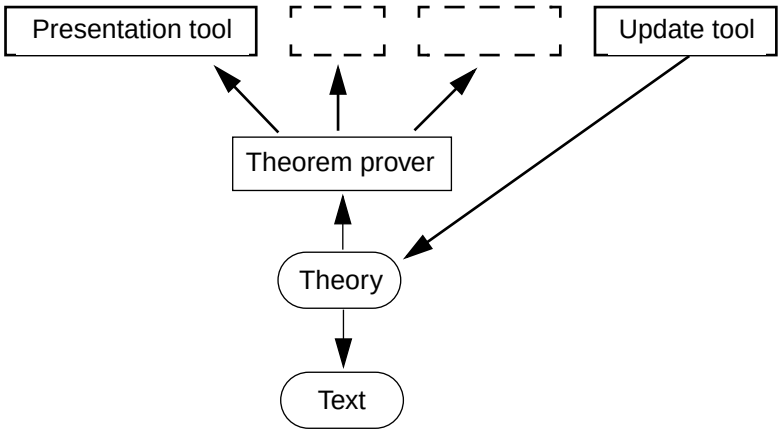


FIGURE 12. TagLog: tool, theory, text

So instead of invoking a parser, we add grammar rules to the current theory. Then sentences, describing the syntactic properties of text segments, will follow as theorems in the theory. This is a logical relation, so it happens instantly, as it were. The theorem prover is there so that we, when the need arises, can generate explicit representations of the theorems.

5.3.1 Cooperation between tools

In this section, we focus on the distribution of work between several tools. Can one tool use the result of another tool? Can several tools use the same knowledge? Can one tool call another tool?

Sinclair: “I am advocating a modularised kit of software tools. Each will specify input requirements and a variety of output conventions. Complex routines can be built up for particular applications by organizing several tools together”. This idea of *piping* between tools, illustrated in Figure 13, seems to be the standard way of implementing cooperation between corpus tools.

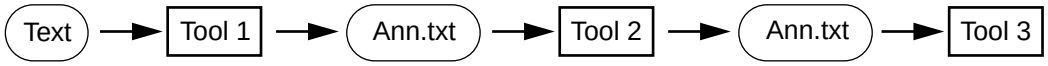


FIGURE 13. Piping between tools

In contrast, TagLog is a system that integrates several tools (in the sense of functionalities), using a *uniform representation*.

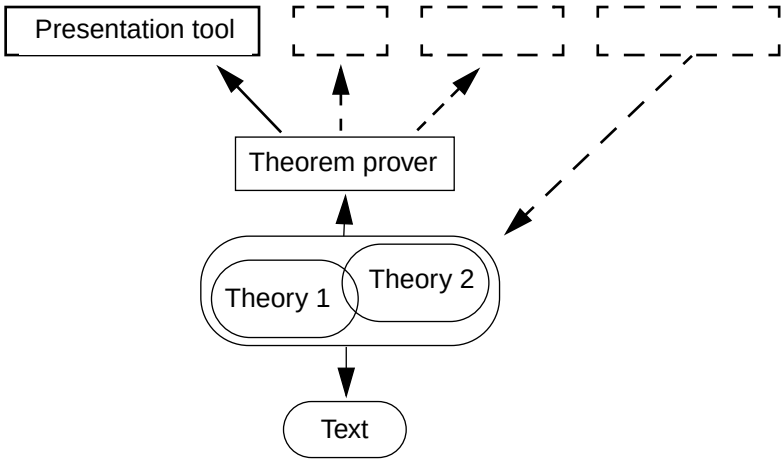


FIGURE 14. Tools making use of uniform representation

Thus, the need for interfaces is greatly reduced. Composing functionalities is a matter of *logically* interfacing different ‘sub-theories’. Whenever one theory (e.g. a grammar) takes as primitive what another theory (e.g. a lexicon) generates, communication between the theories is there. Or composing functionalities is a matter of composing expressions in the query

language. In either case, this is as it should be! The evolving theory is placed in focus, and we, as designers of corpus tools, are relieved of the burden of designing 'input requirements and a variety of output conventions' of different tools.

Composing functionalities in TagLog is sometimes not a logical matter at all, in case it is a matter of initial update to a theory, or final output to a presentation tool. Consider a situation in which we want to generate a concordance of (say) all the nouns in a particular corpus text. In TagLog we would just load a theory that purports to make correct predictions about the lexical categories of segments. We would then open the Concordance Tool in order to monitor the theorems of a certain form, in the form of a concordance.

Now, suppose a certain tagging error is detected during inspection of the concordance, an error that we would prefer to correct on the spot. In TagLog we just select the offending row in the concordance. The corresponding segment in the text is then automatically selected, and the *Statement Browser*, a tool to be introduced in more detail in Chapter 8, shows the formulas describing it. These formulas may then be selected and subsequently deleted or modified. The Concordance Tool will be updated accordingly.

The behaviour of the TagLog system is consistent with a certain view of searching and concordancing as tools for *navigation* in a corpus. An important aspect of a descriptive theory is that it works like a *map*. It greatly extends our means of navigating in the corpus. By so navigating, by observing and exploring, we may be able to correct the errors, and extend the coverage, of the theory/map even further. We will then be able to explore further, and so on. In exactly this manner, TagLog allows extensions and corrections of the map incrementally, as the user moves along, and they take effect immediately.

In a traditional setting, we would probably have to feed the text into a part-of-speech tagger, and then pipe the resulting annotated text to a concordance tool.



FIGURE 15. From text to annotated text to concordance

Such a set-up could hardly support the kind of connectivity, incrementality, and interactiveness that I am arguing for here.

5.4 Macintosh TagLog User Interface

The kernel TagLog user interface consists of a set of windows, a menu bar with menus, and a set of dialog boxes. These are of course highly dependent on the particular TagLog approach to description and analysis, but, except for two of the menus, they are meant to be independent of the user‘s choice of descriptive levels, his way of organizing theories, his choice of coding categories, and so on.

5.4.1 Menus

The Macintosh implementation of the TagLog system provides a standard menu bar and familiar pull-down menus. Apart from the usual File, Edit, and Window menus, there is a Browser menu, a Tool menu, a Tag menu, and a Value menu.

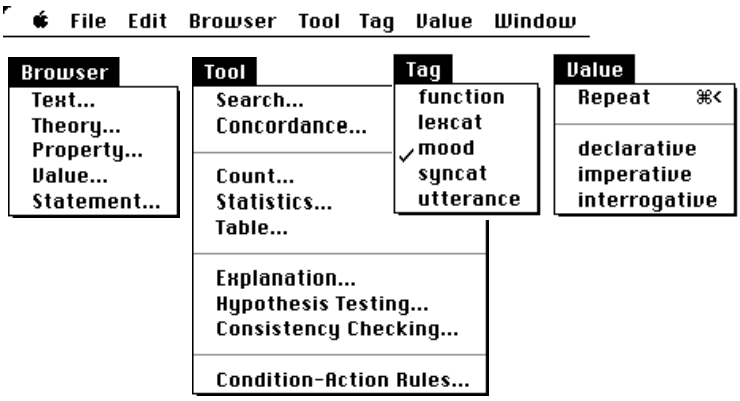


FIGURE 16. The TagLog system’s menu bar and some of its menus

The browsers provide ways to administer texts, theories, tags and tag values, and ways to access the tools in an ‘object oriented’ way. The Tool menu contains tools for searching, counting, concordance building, tabling, explanation, hypothesis testing, and consistency checking. The Tag menu contains tags and the Value menu contains values for these tags.

5.4.2 Windows

The TagLog system consists of the following kind of windows: the Text window, the Theory windows, and the Dialog window. All windows can be moved, resized, zoomed, scrolled, and closed, in the usual Macintosh way.

5.4.2.1 The Text window

The Text window contains the current text, (partially) described by the current theory. This text is not editable, but portions of it can be selected, in the usual Macintosh manner. By selecting a segment in the text, selecting a tag in the Tag menu, and selecting a value from the Value menu, a statement is introduced into the current theory, describing this segment. Or, by selecting a segment of the text and entering a query of the right form in an appropriate window, the properties of the selected segment may be queried. Also, the TagLog system is able to select a segment of the text programmatically, and thus *point* to the segment, in response to a request posed by the user, for example.

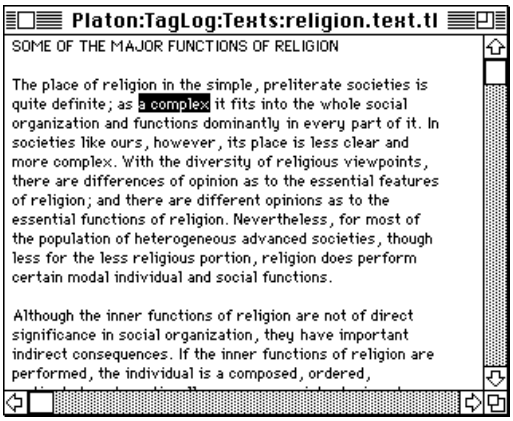


FIGURE 17. The TagLog Text window

5.4.2.2 Theory windows

The idea is to have one collection of statements (ordinary TagLog statement or grammar rules) in each window, and to give the window a symbolic name, under which the content of it can be referred to. These collections are modified manually, by editing them. The content of a Theory window may be saved to a file.

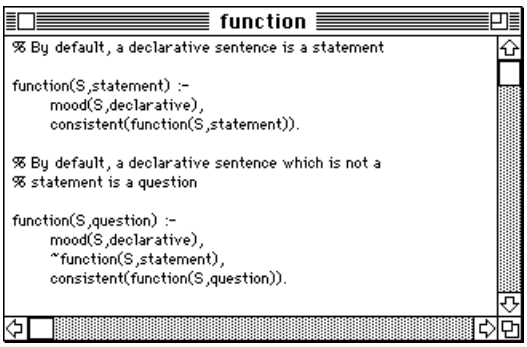


FIGURE 18. A TagLog Theory window

A particular kind of Theory window is the *Grammar window.* This is where grammars, expressed in the grammar notation, are written. An example is given in Figure 19.

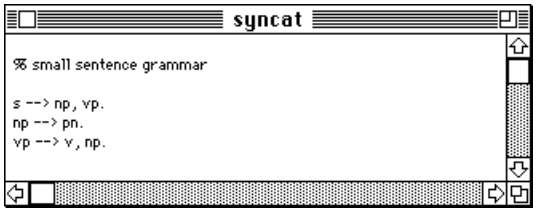


FIGURE 19. A Grammar (theory) window

The *name* of a grammar window plays a special role, in that rules of the form given in (97) are translated by the system into statements of the form in (98).

(97) $C \text{ --> } C_1,...,C_n.$

(98) $\text{<name>}(P_0\text{-}P_n,C) \text{ :- } \text{<name>}(P_0\text{-}P_1,C_1),...,\text{<name>}(P_{n-1}\text{-}P_n,C_n).$

The grammar in Figure 19, for example, is translated into the clauses in (99).

```
(99)    syncat(P0-P2,s) :- syncat(P0-P1,np), syncat(P1-P2,vp).
        syncat(P0-P1,np) :- syncat(P0-P1,pn).
        syncat(P0-P2,vp) :- syncat(P0-P1,v), syncat(P1-P2,np).
```

5.4.2.3 Situation Theory windows

Typically, a Situation Theory window contains singular statements that model the situation(s) to which a text, or a collection of texts, relates. These too, can only be modified by hand.

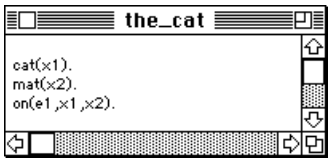


FIGURE 20. A TagLog Situation Theory window

The TagLog environment supports the integration of syntax, semantics and pragmatics in the sense given in Section 4.8. Statements describing the content and context are introduced in the Situation Theory window during the modelling process. The symbols that are parts of expressions written in this window can be highlighted the usual way, for example in order to point out a relation between a segment of text and some entity in the content/context model. As will be shown, many TagLog tools can be used to analyse the relation, in both directions, between a text and a content/context.

5.4.2.4 The Dialog window

The TagLog Dialog window is a so called ‘command line interface’ to TagLog. By calling the built-in predicates from this interface, all the functionality of TagLog can be accessed; nothing is buried behind the buttons and menus in the user interface. On the other hand, this command line interface is not particularly ‘user friendly’.

The only thing that meets the eye of the user is a prompter that invites him to formulate his request. To distinguish the TagLog prompter from the ordinary Prolog prompter (?-), the TagLog prompter consists of an exclamation mark, followed by a dash (!-).

Typically, the user writes his request after the prompter and presses the return key, and it is normally the case that when the system has computed

an answer to the query, it presents the result, and waits for input from the user.

Referring back to Figure 17, the loading of the window, and the highlighted area in it, is an effect of making the two commands in Figure 21.

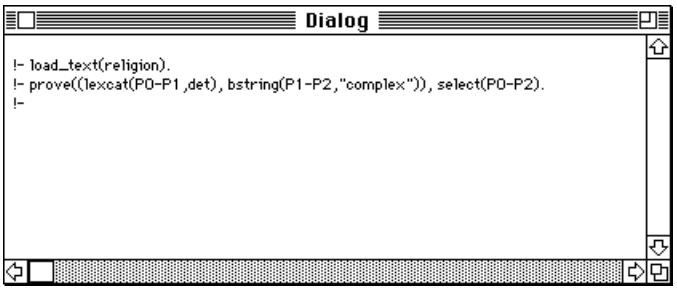


FIGURE 21. The TagLog Dialog window

To save space, many of the examples in the present thesis will be formulated in this simple environment. This does not mean that they cannot be formulated and executed in any other way; on the contrary, the TagLog tools most often provide a much more accessible interface.

5.5 The TagLog Schema

The role of the *schema* is to unify and coordinate the elements of a TagLog corpus 'project'. Creating a schema is usually the first step in the process of developing a corpus theory. The user specifies the texts that are to be included and how they are to be *tokenized*, i.e. divided up into basic segments. The user also specifies what coding categories will be available, what universal theories will be drawn upon, etc. However, the schema will usually be further refined in the course of theory development.

A schema, like many other things in TagLog, is a kind of theory.[5] One group of schema predicates is used for telling the system about the texts that are available in the corpus defined by a particular schema, and which

5. Formally speaking, the schema is a *meta-theory* with respect to other TagLog theories. As a meta-theory, however, it is very limited. It determines the behaviour of certain aspects of the system, and it influences the object theories, but no proper meta-level reasoning is performed in it.

theories are associated with them. Each text is associated with an abstract identifier (ID). A table maps IDs to file names. One ID may correspond to many different files, one for storing the text itself, others for storing descriptions of the texts, as well as various indices that map between the text and the descriptions (e.g that map file positions onto point expressions). Universal theories too, are associated with identifiers.

Another group of schema predicates is used to inform the system about the current tag set, by declaring, for (some of) the predicates of the theories, the possible values of their arguments. For example, if the clause *tag(syncat)* and the clauses *tag_value(syncat,n)* and *tag_value(syncat,v)* are added to the current schema, then the menu command *syncat* is automatically added to the Tag menu, and the menu commands *n* and *v* are added to the Value menu.

An overview of the TagLog schema predicates is given in Appendix A.

5.6 The Choice of Interaction Styles

An important step in the process of designing an interactive system is the choice of interaction style. In his 1992 textbook, Shneiderman distinguishes the following interaction styles:

- menu selection
- form fill-in
- command language
- natural language
- direct manipulation

I will argue that the TagLog user interface is best characterized as a hybrid system, where three different interaction styles – direct manipulation, menu selection and ‘command language’ – are carefully blended.

In what sense is TagLog a direct manipulation system? If it is true that “presenting a visual representation of task objects and actions is the heart of the direct manipulation approach to design” (Shneiderman, ibid. p. 65) then surely TagLog is a direct manipulation system. Texts, theories, statements, properties, etc. (abstract task objects) are represented visually in

the system. The pointing to these objects by means of the mouse is a kind of direct manipulation action.

5.6.1 The pragmatics of human-computer interaction

Since I realize that a blend of different interaction styles may seem to run the risk of being confusing, I will spend some effort on the defence of this choice. It is my contention that the branch of linguistics called pragmatics is able to throw light on this kind of mixed system. In fact, it is my conviction that notions from pragmatics – such as communicative act, turn taking, feedback, deixis, presupposition, cooperativeness, and relevance – ought to play a much more important role in theories of computer system usability than they currently do.

For example, *speech act theory* can be used to argue that the term *command language* is misleading, when it is meant to denote all uses of artificial (as opposed to natural) languages in user interfaces. TagLog, for instance, uses a formal language which we do not want to describe merely as a command language, but also as a declarative description language and a query language – a language for making statements, for giving commands, and for asking questions. In speech-act theory, statements, questions and commands are regarded as different kinds of communicative acts. A natural language example of the differences would be the following:

(100) *The door is closed.*
Is the door closed?
Close the door!

Only an utterance of the last sentence above deserves to be called a command. I think that this distinction is a very important one, and that the failure to make it is a sad reflection of the state of pragmatics in the field of human-computer interaction research today.

Another notion from pragmatics that we will make use of in the current analysis is the notion of *deixis*. In human face-to-face conversation participants often specify elements of their common visual world by means of a combination of deictic expressions – e.g. pronouns such as *this*, *that* and *there* – and body movements, mainly pointing gestures. Typical natural language examples would be the utterance of “Close that door!” while

pointing to a door, or the utterance of “This is a noun.”, or “What kind of word is this?”, while pointing to a word in a text.

In TagLog, ‘talking about text’ typically involves deictic reference to particular segments of text, by selecting with the mouse, and the making of statements, commands or queries, by means of choosing from menus or by means of the formal language. In this sense, deixis establishes a link between expressions in the TagLog formalism and the graphical representation of text.

5.6.2 Talking with TagLog tools about texts and theories

In an interesting paper, Brennan (1990) advocates a pragmatics approach to human-computer interaction. Her view is that:

> ...thinking of user interfaces in terms of real conversation leads to more specific predictions about how people will behave and to more informative explanations of why interface features succeed or fail. The hope is that employing the conversation metaphor will lead to insights in the kinds of architectures and features that are necessary to support cooperation between people and computers.

In accordance with these ideas, we will regard the user’s interaction with the TagLog system as a kind of conversation, a conversation about texts and theories, a conversation involving both the use of a declarative formal language, direct manipulation, and menu selection.

For example, in TagLog, the process of hand coding, i.e. the process of describing a segment of text as belonging to a certain category, can be regarded as a kind of dialogue, comprising two turns: the user selects a segment in the text (direct manipulation), chooses a tag in the Tag menu (menu selection), and chooses an appropriate value from the Value menu (menu selection). In the second turn, the system gives feedback (typically by using language) to indicate whether it accepts the description or not.

As another example, a query from the user to the system typically involves the user writing a formal declarative specification in a text entry field (formal language use), choosing from menus to set additional parameters (menu selection), and clicking a button in order to execute the query (menu selection). The system response is sometimes given as statements in the formal language, and sometimes it involves the setting of a selection of a segment in the current text (direct manipulation). This turn-

taking scheme could be extended with turns where the user demands an explanation of a statement (by pointing to it and clicking a button) and the system gives such explanation, which in turn would imply more interaction with the user in the same manner.

The notion of conversation may seem to stand in some kind of opposition to the notion of direct manipulation. However, as studies of face-to-face communication clearly show, conversation usually involves not only words, but also pointing gestures, nods, eye-to-eye contact, *um humns*, sign language, etc. Moreover, communication most often takes place in the context of some activity involving non-communicative acts as well. I agree with Brennan when she claims that:

> ...the dichotomy between direct manipulation and conversation is a false one [..] First, direct manipulation interfaces succeed because they share important features with real conversations. Second, when so-called "conversational" interfaces fail, it is because they lack these pragmatic features – that is, words alone do not a conversation make. Third, real conversations actually fulfil many of the criteria for direct manipulation.

The conversational view is a powerful design metaphor for a corpus theory development system. By looking at the interaction between the user and the system as a conversation, we are free to apply maxims of cooperative communication (Grice, 1975; Allwood, 1976), and on such grounds design the most efficient and cooperative dialogue possible. Besides, it is reasonable to believe that a potential user of the system will feel quite at home with this view, especially if the user is a linguist.[6]

To sum up, the TagLog working environment integrates a logical approach to description and analysis of texts, and a theory revision approach to scientific method, with a conversational view on human-computer interaction, while still exploiting high-resolution graphics, interface objects like menus and windows, and a mouse. The interaction between

---

6. To talk about ourselves as having conversations with our tools is not as strange as it may sound. In particular, this view does not imply that we must regard the tools as intelligent, alive, human, or conscious; and this is fortunate, since I agree with Shneiderman that such anthropomorphism should be avoided in most circumstances. It may be charming for a while, but it is very tiring in the long run. I am confident that we can regard TagLog as a set of tools with which the user interacts, without throwing away the insight that interaction with computers is a kind of communicative behavior – a kind of conversation.

researcher and system is analysed in terms of statements, questions and commands, and deixis is seen as an important link of integration between expressions in the TagLog formalism and the graphical representations of text, and between menu commands and such representations.

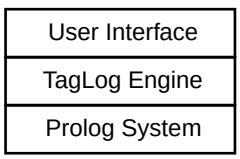
The principles underlying human conversation are very powerful, and I strongly believe that a theory which is able to explain why some interfaces work while others do not, will have to relate to these principles in one way or another.

5.7 Architecture

Logic is good for achieving machine independence, implementation independence, even proof strategy independence. Indeed, the logical approach to corpus linguistics does not have to involve an Apple Macintosh; it does not have to involve a von Neumann computer; it does not have to involve the Prolog programming language; it does not have to involve a depth-first search strategy, nor does it have to involve the resolution inference rule. Now, it just so *happens* to involve all that in the implementation reported on here, but it does not have to.

5.7.1 The current system

TagLog for the Macintosh is built as an *extension* of a Prolog system. Around the Prolog kernel, a layer of TagLog specific procedures are built, which the user can run by manipulating the menus and buttons in the user interface, or by calling them from the command line interface.



Thus, TagLog is implemented *on top of* – rather than *in* – Prolog. There is (almost) never any level of interpretation between TagLog and Prolog. Most of the time, TagLog uses the *raw* power of Prolog to perform its reasoning; TagLog theories are ‘run’ as Prolog programs.

5.7.2 Alternative backbone systems

To base TagLog on Prolog is not the sole option. As can be seen in Figure 22, there are a number of alternative backbone systems around.

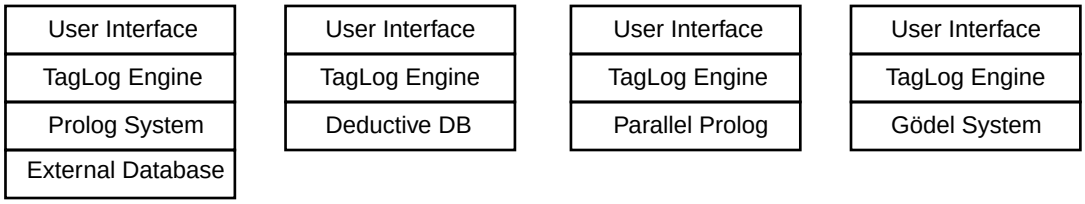


FIGURE 22. Four ways of building a TagLog system

By coupling a Prolog system to an external database, e.g. a relational database, it is possible to lessen the RAM requirement of TagLog. The cost in terms of speed will be great, however, and besides, RAM is cheap these days.

A *deductive database* is a generalization of a relational database, where not only facts, but also rules, just like in Prolog, can be used. In fact, the coupling of Prolog with a relational database described above can be regarded as a 'poor man's' version of a deductive database. Since deductive database technology aims at very large fact bases and rule bases, they seem to be very good candidates for a TagLog backbone.[7] However, the technology has not really matured yet, and as far as I am aware, no commercial deductive database management system (DDMS) exists.

The prospects for fast *parallel Prologs* are very good. As a matter of fact, at the time of writing this, Sicstus Prolog version 3.0 comes with library procedures for running Prolog in parallel (running a particular example program with 5 instead of 1 processor is reported to speed up the execution with a factor of 3.8!).

Gödel (Hill & Lloyd, 1991) is one particular example of a logic programming language that addresses some of the shortcomings of Prolog, in particular logical shortcomings. *Mercury* (Zomogyi, Henderson & Conway,

---

7. In an interesting paper, (Dahl, Huang, & Tarau, 1994) show that logic grammars may very well be implemented within a deductive database framework, and that some problems that Prolog has with logic grammars, in particular the difficulty to handle left-recursive grammars, disappear within this framework.

1994) is another logic programming language – efficient and purely declarative – that looks promising.

5.7.3 Client-server architecture

The current Macintosh implementation of TagLog is a rather traditional single-user application. A *client-server* architecture is an interesting alternative. In a client-server TagLog system, a central TagLog server would handle all the hard-core reasoning and other processing. It would also maintain single copies of texts and theories and make them available to all users, resulting in a cut down on data redundancy and potential inconsistency. A protocol would serve as the sole interconnection between the server and the clients.

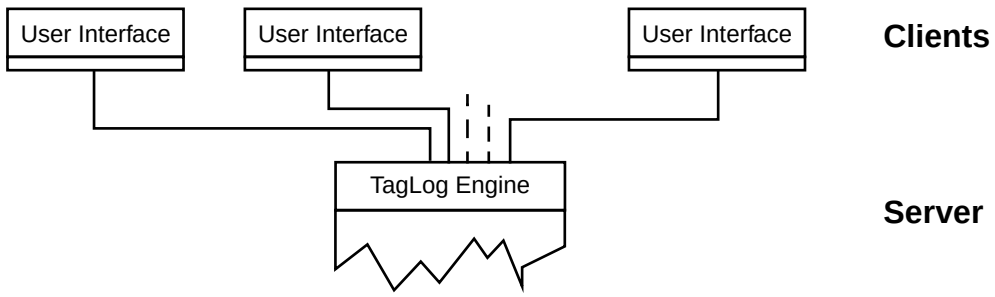


FIGURE 23. TagLog as a client-server system

When a client tool needs information from a theory, it would form a request for it and send that request to the server. After the server has performed the desired reasoning or processing, it would send the resulting information back over the net to the client. A single-user setup is merely a special case of this schema, where the server and one client reside on the same machine.

Since the user interface code – always the most difficult code to port – is restricted to the clients, a client-server architecture also has the potential of making porting between different platforms easier.

5.8 Efficiency

It follows from the fact that TagLog theories are run as Prolog programs that the performance of TagLog is dependent on the performance of Prolog. Prolog has a reputation of being slow that probably dates back to the years when Prolog was interpreted. But many modern Prolog systems of today come with both an interpreter and a compiler. Some systems even allow for compilation to machine code. Reasoning in a compiled theory is very efficient.

Still, as we shall see, in particular the logical approach to automatic tagging, and lemmatization as abduction, have efficiency problems which might even render them practically useless until these problems are solved. Moreover, certain other operations in TagLog might turn out to take a very long time, if one is not careful when formulating queries and specifications.

Fortunately, fairly general techniques exist for optimization of Horn clause queries: dynamic reordering of goals so as to constrain the search for solutions as early as possible (see e.g. Rayner, 1993), memo functions to save the results of sub-computations to be used later in computations (see e.g. Sterling & Shapiro, 1986), etc. Furthermore, optimization techniques that use knowledge of invariants in the domain of text in order to 'rewrite' queries into forms more suitable for efficient computations would probably be possible, although this would need to be investigated in more detail.

Finally, certain kinds of reasoning in TagLog theories are probably more efficient to implement as text crunching procedures instead. This does *not* mean that we are back to square one: we have realized that text crunching is something that belongs on the level of implementation, not on the conceptual/logical level.

5.9 Summary and Conclusion

In this chapter, the design of the TagLog system – a computer program implementing the logical approach to corpus linguistics and *supporting*

the activity of corpus linguistics – was discussed. Since the development of corpus theories is best described as a theory revision problem – the corpus is usually approached with a theory that will have to be tailored to fit that particular corpus – I have designed the system as a theory revision system, a balanced cooperative system.

What emerges is a picture suggesting that the computer supported corpus research envisioned in the present work is quite a complex activity consisting of very many, *in essence*, highly interrelated subactivities: the user observing the (parts of a) text presented by the system; the user asking the system to present (a new part of) the text (in a new way); the user describing the segments of the text and the system storing the descriptions; the user browsing a theory created by him and stored by the system; the user demanding an explanation of a statement and the system giving it to him; the system testing a hypothesis supplied by the user; the system generating a prediction, and the user checking and confirming (or disconfirming) it (by adding to the theory), the system finding a contradiction and the user resolving it (i.e. by removing statements from the theory); the system suggesting different (and competing) hypotheses that explain a fact and the user selecting among them; etc.

I have argued that throughout this process it is the evolving theory, in relation to the texts that it purports to describe, that ought to be focused on. The interaction between user and system is there to support this. The goal is a system that vanishes as users become completely absorbed in thinking about theories about texts.

But not only do the tools of the system interact with the user. Tools interact with tools as well. I have argued that piping is not the best way of implementing this. Rather, the use of a common uniform representation – the explicit representation of the evolving theory – is crucial here.

In this chapter, a particular implementation of TagLog for Apple Macintosh computers has also been introduced. This system will be used, in this thesis, to provide a concrete illustration of the approach. But it was emphasized that this implementation, and indeed any Prolog-based implementation, is not essential to the approach; alternatives, such as deductive databases and logic programming languages other than Prolog, were mentioned as well.

CHAPTER 6

Description

6.1 Introduction

In Chapter 2, *description* was characterized as a stage of scientific linguistic study consisting in categorizing data obtained by observing the facts of linguistic reality. Moreover, description was seen as a preparatory stage of transition to the development of descriptive universal theories. Description and explanation are rather closely connected: without a description of facts it is impossible to explain them. On the other hand – and this will be one of the themes in Chapter 13 – description without explanation is not enough for science.

In Chapter 3, I presented the TagLog formalism as a language in which we are able to express theories about texts. In the first part of Chapter 4, we looked at particularly simple kinds of theories, namely descriptions. In this chapter, the focus will be on the *process* of description, or *tagging*, as it is sometimes called in the context of corpus linguistics. Where traditional approaches suggest the use of a word processor for entering tags, or a program that is able to tag a text automatically, the TagLog approach takes as its point of departure a rather different view of what tagging con-

sists in. From the point of view of logic and knowledge representation, the process of describing a text must involve *changing what follows from the theory purporting to describe the text*, i.e. *updating* this theory. I use the slogan 'Tagging with a Theory' to designate this approach.

Within this framework, there are kinds of tagging that are possible to perform in an automatic or semi-automatic fashion, and we will return to some such methods in Chapter 11. But we will begin by considering tagging as a manual process. I call this *hand coding*.

6.2 The Importance of Hand Coding

For some reason, 'tagging' has become almost synonymous with 'automatic tagging'. Moreover, a lot of work has been done on automatic tagging, whereas hand coding has received very little attention, if any at all.

The present thesis takes the issue of hand coding rather more seriously. In my view, there are many reasons for not neglecting this subject:

- Hand coding is linked to *observation*, which is an extremely important notion of empirical science.
- Through his senses, a human observer can, in principle, have full access to the (local and global) contexts and co-texts of segments of text.
- Many coding tasks cannot, at present, be performed automatically, and *must* therefore be done by hand.
- A description often needs to be revised in the light of the findings when using an analysis tool.
- There is no clear-cut division between manual and automatic tagging and there are many opportunities for semi-automatic tagging.
- Automatic taggers often need manually coded corpora in order to learn to perform their tasks automatically.
- An automatically tagged corpus often needs to be corrected manually.

6.3 The Elements of Hand Coding

Analytically, one can distinguish at least the following steps in the complex process of a user **U** describing a fact **F** to a system **S**, by stating **P** by means of a sentence *S*:

1. **U** observes **F**
2. **U** decides to describe **F**
3. **U** forms a sentence *S* by grouping together certain syntactic primitives in a language
4. By asserting *S*, and perhaps also by *pointing* to a segment or to another object, **U** states **P**
5. **S** checks that **P** is consistent with what **S** already knows
6. **S** stores **P** (by storing *S*, or something equivalent to *S*)

In the following sections, under the headings of Observing, Deciding, Framing, Stating, Checking, and Storing, each of these steps will be discussed.

6.3.1 Observing

Empirical science is based on observation. But in corpus linguistics, what is it we observe? In general, and this is how I like to see it, knowledge about a text is obtained through observation of the *facts* that make up the text. The facts that form the content and the context, and the facts that relate the text to its content and context, can sometimes be observed as well, and are important.

For the purpose of the present discussion, observation is to be regarded as perception of the outside world for the purpose of providing the primary data for scientific research. In addition, observation may be classified as simple or complex, direct or indirect, aided or unaided, etc. In many sciences, observation implies the use of various instruments (such as microscopes and telescopes) to compensate for the natural limitations of human sense organs. The use of various vehicles (such as space craft and submarines) to support transportation to advantageous points of observation is also common. Texts are made to be read, so the corpus linguist does not

need sense-amplifying instruments. The need for transportation is there, however, since the textual reality is immense, almost endless, so that the lack of complete overview must be compensated for by support for mobility. We will return to this point in Chapter 7.

Obviously, *reading* may be regarded as one particular form of 'observation' of texts. The particular form of knowledge obtained through reading, namely knowledge of the content of the text, is what we mean by *understanding*. Now, the purpose of (corpus) linguistics is *not* to read and understand the content of texts, but to *describe*, *analyse* and *explain* them (and then often other aspects than their content). So, in this respect, scientific observation of text is different from reading.

How does scientific observation work in corpus linguistics? Let us begin with some simple observations (no pun intended):

- A linguist *seems*, at least as long as we are not too philosophical about it, to be able to *directly* observe (at least some of) the facts of a text, even in cases where these facts appear to be very high-level and complex. For example, think of someone observing the fact that a particular segment of text is a sentence, or that it functions as a question, or that it relates to context in a particular way. Usually, there is no trace of conscious 'derivations' of high-level, complex facts from more low-level, basic ones. In fact, the linguist may not even be able to observe the more low-level/basic facts!
- Seemingly without effort, and usually without even noticing any alternatives, researchers seem to be able to disambiguate between different senses of words, different interpretations of sentences, different possible communicative purposes of uttering a sentence, and so on.
- However, to its full extent, the above holds only if the linguist in question is properly *trained*. Moreover, observation is not infallible, and once in a while, even a trained linguist does make observational *mistakes*, seeing something which is not there, or missing the obvious.

Clearly, what makes all this possible is the researcher's in principle more or less unrestricted access to the contexts of text segments under investigation, together with his theoretical and common sense knowledge, tacit and explicit. It would be far beyond the scope of this thesis to attempt to unpack all this, however, and we will content ourselves with the basic insights as they stand: trained researchers are able to observe the facts of

linguistic reality, and although the observations are usually correct, mistakes are certainly possible.

Just a few remarks to link all this to some intensely debated questions in the philosophy of science: What is observable? Popper claims that 'observability' is a notion that can only become sufficiently precise in *use*, in the context of a particular scientific practice (cf. Popper, 1980, p. 103). The assumption that I make in the present connection is that all facts of texts are in principle observable. I simply do not know of any convincing attempt to demarcate between what is observable and what is not, that would deem some facts of text observable, but exclude others. Now, it is true that we cannot directly observe what goes on in the minds of the writer or the reader of a text, so either we could decide that what goes on in people's minds is not what corpus linguistics is about anyway, or we could decide that it is appropriate for a corpus linguist to make *assumptions* about (rather than to observe) what goes on in people's minds, and to build his theories on these.

Besides, we may not be able to determine, just by looking at a particular segment of text, the nature of the beliefs and attitudes that made the writer produce it, and we may not be able to determine what reading this segment gave rise to in the mind of a particular reader, but we are presumably in the position to say that a certain phrase in a text *presupposes* certain beliefs or attitudes on the parts of the writer and the reader, and to say that a particular phrase has a certain content, just by looking at that segments in context. But then we are talking about the text, not about the minds of the reader and the writer.

Texts are 'public' objects, no less public than chairs, or football games, or microwave ovens. Researchers, given that they have been appropriately trained, are able to observe facts relating to the syntactic properties of text segments, facts relating to the contents of texts, and other facts relevant to them. Unless their theoretical backgrounds differ considerably, they are normally willing to agree on what they see, and they are usually convinced that if they go away and then come back again, they will be able to make the same observations. As far as I can see, this is what it takes to be observable.

Is observation theory neutral? In contrast to the logical positivists, who used to think of observation as being unaffected by theoretical perspec-

tive of the (trained) observer, most philosophers of today tend to think of observation as (necessarily) *theory impregnated*. As should be evident from the above, I see no reason to think otherwise. However, knowing that observation is theory impregnated certainly does not make observation less important for empirical science; it is still the only tie we have to reality, be it texts, be it other parts of the world.[1]

We cannot resolve these questions here, nor even discuss them in any depth. My point is that I think that there is hope in corpus linguistics for a notion of 'observability' that can become 'sufficiently precise in use'.

In conclusion, it seems obvious to me that when designing a system for description and analysis of texts, the remarkable capability of a trained linguist to observe what is really there in a text should not be allowed to go to waste. This capability should be exploited and supported to the greatest possible extent, and nothing should be allowed to hamper it.

6.3.2 Deciding

Needless to say, not every fact that happens to be observed by the linguist is considered worth describing. The linguist is likely to decide to describe only facts that he thinks will matter to his (future) theory.

6.3.3 Framing

After the observation of a fact, and the decision to describe it, comes *framing* of the description. Description of a fact **F** involves the act of composing a sentence *S* (that expresses a statement) that describes **F**. The symbols of *S* refer to the elements of **F** by virtue of the intended interpretation of the description language. Segment-expressions, predicate symbols and terms refer to segments, properties and property values, respectively.

For example, given the observation of a particular segment in the current text as a noun, and given that the intended interpretations of the symbols *lexcat* and *n* are lexical category and noun, respectively, and that the

---

1. Let me also say that I oppose the view, which has been held, that a theory about texts should take as its point of departure – its foundation – the printer's ink, the laserwriter's toner, or the pixel pattern on the screen of a computer. This may be an appropriate point of departure for a theory of our perception of the outside world, but it would hardly serve the needs of corpus linguistics.

intended interpretation of the term *3-4* is that it denotes that particular segment, the sentence in (101) is composed.

(101) lexcat(3-4,n)

6.3.4 Stating

In this thesis, stating something means adding it to the current theory. The TagLog built-in predicate *add/1* will add a unit clause (expressing a singular statement) given in the argument to the current theory, e.g, the call in (102), will add the clause *lexcat(3-4,n)* to the current theory.

(102) !- add(lexcat(3-4,n)).

Clearly, since the 'names' of the segments are not very 'user friendly', a better way to refer to them is often the kind of demonstrative reference that the use of the built-in predicate *selection/1* allows.

(103) !- selection(S), add(lexcat(S,n)).

The execution of the *selection/1* procedure binds *S* to (the expression denoting) the current segment, i.e. to the segment highlighted in the current text, and the execution of the *add/1* procedure adds the resulting statement to the current theory.

There are other ways to assert a statement than to write a sentence down, character by character, in a text editor, or in a command line interface, and then issue a command. In TagLog, the most important alternative is to compose statements by pointing to segments, and selecting from menus.

The segment which is selected (highlighted) in the Text window is called *the current segment*. In order to tag the current segment with a value for a tag selected from the Tag menu, a value is chosen from the Value menu. This action will add a statement of the form *P(S,V)* to the current theory, where *P* is the tag, *S* the current segment and *V* the chosen value.

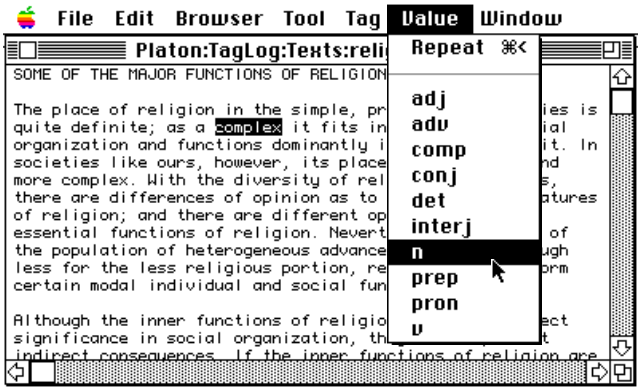


FIGURE 24. Hand coding in TagLog

Technically, what happens is that the following TagLog goal is called:

(104) !- selection(S), add(P(S,V)).

Again, note that even though it would also be accurate to say that we are pointing to a *string*, we are not *saying* anything about that string. (Only if selecting from the Tag menu were to introduce a sentence *syn-cat(“loves”,v)* would it be correct to say that we were saying something about a string.) Hand coding means speaking about segments, not about strings.

6.3.5 Checking

A ‘yes’-response from the system, following directly after a statement from the user, indicates that the statement was accepted and will be stored accordingly. In other words, it works like a kind of feedback.

(105) !- selection(S), add(lexcat(S,n)).
yes

There is at least one occasion when an attempt to update the current theory should not lead to an actual update, namely when the statement gives rise to logical inconsistency. Therefore, *add/1* implements a consistency checking step, and the next step – the storing step – is taken only if the consistency checking step succeeds. If it fails, the current implementation will notify the user and refuse to update the current theory.

6.3.6 Storing

The storing step is where the system makes itself 'remember' the sentence composed in the framing step, and asserted in the stating step. Given a Prolog backbone to TagLog, the storing step is implemented by the Prolog database manipulation predicates, where *assert/1* is the most important one.

The most important effect of asserting a clause is that it becomes active in the sense that it becomes visible to the TagLog prover, and thus that it can take part in reasoning, as a premise in a deductive step. The clause is stored in the internal memory of the computer (at least this is true for the current implementation), and is indexed for efficient access. Efficient access to knowledge is of course an important requirement for a system like TagLog, but it will not be further discussed here.

6.4 Coding as a Dialog

In TagLog, the process of coding can be regarded as a kind of dialogue, in which the user makes statements about segments by highlighting the segments and selecting from menus. The role of the system in this dialog is to accept (or, in certain cases, refuse to accept) the statements, and to remember them, e.g.

(106) U: *This is a noun phrase [pointing to a segment]*
S : *Ok*

Furthermore, I would like to promote the view of description and *analysis* as two aspects of the same ongoing 'conversation', the purpose of which is 'talking about text'. The following dialog illustrates the idea:

(107) U: *Please find me a* sentence *in this text*
S : *Ok, here's one [pointing to a segment]*
U: *WHY is this a sentence?*
S : *It consists of a noun phrase followed by a verb phrase, etc.*
U: *I would regard this as an instance of a* question *then*
S : *Ok*

In simple terms, this dialog can be seen as a request for search, followed by an ostensive demonstration by the system of what was found. Then a

request for explanation, followed by an explanation. Finally, a description followed by a kind of feedback indicating acceptance of the description.

There are at least two good reasons for integrating the functionalities for description and analysis in one single system, and to build support for cooperation between them, or at least for quick and effortless switching. First, a description often needs to be revised in the light of the findings that the researcher made when using an analysis tool. Secondly, analysis can often lend a substantial amount of support to the process of description.

6.5 Hand Coding in the TagLog System

Hand coding is a time-consuming and error prone process. It therefore lies in the interest of the researcher to make coding as fast and accurate as possible. The TagLog system supports various methods for speeding up the process of coding, means of providing feedback in order to ensure accuracy, and simple ways to correct an error, should it arise anyway.

6.5.1 Ways to speed up coding

An important way to speed up hand coding is to use the Search Tool or the Concordance Tool in order to set the current selection more or less automatically to segments denoted by a set-expression given by the user. More on this in Chapter 7.

To support quick and easy access to menus, the TagLog system features so called ‘tear-off’ menus. The Value menu can be torn off, and be made to ‘float’ at any position of the screen. This is often a faster way of coding since the pointer does not have to be moved all the way up to the menu bar all the time.

The Repeat command of the Value menu repeats the assertion of the most recently added statement, but for the current segment. This command has a keyboard equivalent which allows the user to select segment after segment and just press these keys in between, in order to assign tags to these segments. This is a way to speed up the tagging process considerably when there are many segments that are to be assigned the same tag value.

Although not implemented yet in the current system, it is easy to imagine a facility for assigning tags and tag values to keys and key combinations on the computer's keyboard. For some tasks, this might be the fastest way to code a text.

6.5.2 Feedback when coding

The Statement Browser is able to display the statements describing the segment currently selected in the Text window. Updates to these statements will always be dynamically reflected in the Statement Browser. (See Chapter 8 for more details about the Statement Browser.)

Furthermore, as already mentioned, the system is able to make various consistency checks. The current way to inform the user of an inconsistency is to display an error message in the usual Macintosh way.

6.5.3 Undoing statements

When coding, mistakes are easily made. There are currently two ways to withdraw a statement, in order to repair such a mistake:

1. The most recently asserted statement may be 'taken back' by means of the Undo command found under the Edit menu. The actual wording of the Undo menu item indicates if there is a statement that can be taken back.
2. The Statement Browser allows you to delete any singular statement about a segment from the current theory. Place the insertion point on the statement you want to delete and press the Backspace key, or choose Clear from the Edit menu.

6.5.4 Macros

Many tag values – and this is especially true for terms representing the kind of categories that modern grammar theory talks about – can be rather unwieldy, and they can be very many, too large and too many to fit in a menu. The *macros* in TagLog address both of these problems. The TagLog formalism allows the user to define macros which expand into property values at 'tagging time'. For example, (108) defines a macro named *v4* and (109) declares *@v4* to be a possible value of *syncat*.

(108) `v4 macro v(finite,[np(nom),np(obj1),np(obj2)]).`

(109) `tag_value(syncat,@v4).`

Then, if a segment (say *5-6*) is selected in a text, and if *@v4* is selected from the Value menu, then the statement *syncat(5-6,v(finite, [np(nom), np(obj1),np(obj2)]))* is added to the current theory.

The result of the expansion of a macro can be made to depend on what 'state' the system is in, e.g. on the current selection. For example, (110) defines a macro named *iv*, and (111) declares *@iv* to be a possible value of *syncat*.

```
(110)   iv macro (s\(np:X):P) :-
            selection(S),
            string(S,W),
            string2atom(W,F),
            P =.. [F,X].

(111)   tag_value(syncat,@iv).
```

Then, if a segment (say *23-24*) is selected in a text, and if this segment is an instance of the string "walks", and if *@iv* is selected from the Value menu, then the statement *syncat(23-24,s\(np:X):walks(X))* is added to the current theory.

The TagLog macro facility is hardly of any theoretical interest, and it does not increase the expressive power of the formalism. Macros are just one of those things that makes TagLog practical, in particular for handling the large and complex structures dealt with in unification-based grammars.

6.6 Summary and Conclusion

I have argued that from a TagLog point of view, the process of tagging a text means *changing what follows from the theory purporting to describe the text*, i.e. *updating* this theory. This can be done by hand, and in some cases it can be done automatically. This chapter dealt with hand coding.

The TagLog system was shown to support the view of hand coding as an interactive dialog between the system and the user, where the user makes statements about segments by highlighting them and selecting from a menu. Various means of speeding up coding, ways to monitor the tagging process in order to provide feedback, and the use of macros, were explained.

CHAPTER 7

Exploring Linguistic Reality

7.1 Introduction

The distinction between data and theory assumed in this thesis is reflected in a division of labour between two kinds of browsers in the TagLog system: there are browsers for browsing text, and there are browsers for browsing theories. In this chapter we will be looking at text browsing tools, and in Chapter 8 we return to tools for theory browsing.

As was noted in Chapter 5, there are two important aspects to the linguist's exploration of texts: navigation and observation. A system has to support both of these aspects: it has to be able to provide quick and accurate ways of 'transporting' the linguist to where he wants to go in the text, and it has to be able to give him a nice view of the surroundings when he gets there. In this chapter we look at *searching* and *concordancing* from this perspective.

The TagLog *Search Tool* is able to take a user-supplied (partial) description of a segment, find an actual segment in the current text that matches

the description, point it out to the user, find the next segment that matches, point *that* out, and so on.

The *Concordance Tool* accepts a set-expression denoting a set of segments, finds each of these segments and presents the corresponding string on one line in a special window, surrounded by some of the co-text of that occurrence. (To extend our 'geographical' analogy, the Concordance Tool sends postcards from the linguistic reality!)

Both of these tools are able to use a *theory* (e.g. a grammar) in order to extend their scope. Thus, they provide a clear example of a theory being used not only for the purpose of answering queries, but also for navigation. In this respect, theories are the maps and guidebooks to the linguistic reality.

7.2 Searching

For the purpose of the present discussion, *searching* is an attempt to find segments of text that satisfy a certain (partial) description. The segments being searched for will be referred to as the *target segments*.

In the Artificial Intelligence literature, where "searching" is usually used in a different sense, it is sometimes claimed that deduction is nothing but searching (cf. Nilsson, 1980). However, as will be shown, one might just as well think of it the other way around: searching as deduction.

7.2.1 Searching as deduction

It may not be obvious that the choice of logic as a representation formalism will allow searching in the required sense. But, as a matter of fact, from a logical viewpoint, and given an appropriate representational strategy, searching is not different from other kinds of deduction.

As regards the representational strategy, the most important requirement is that the segments are made *explicit*, i.e. that they are recognized as members in the domain (in the sense of model theory), so that they can be quantified over. This is catered for in TagLog: at least one argument in a singular statement about text should be a segment-expression, denoting a segment in the text.

As a first attempt at searching as deduction, suppose we tried to prove an existentially quantified sentence formulated in ordinary predicate logic notation, as follows:

(112) |- ∃s[*syncat(*s,*conj)*]

The problem here is, of course, that we normally would not be satisfied by just getting to know that there *exists* a segment in the current text such that it is an instance of a conjunction; we would also want to learn the exact *identity* of that segment. Thus, we need to be able to perform *constructive* proofs. Fortunately, the TagLog prover is (basically) a constructive theorem prover, that will (almost) always bind variables in the course of a proof.

Thus, since TagLog is explicit about segments, and since the TagLog proof procedure is able to constructively infer the existence of segments satisfying a particular description, search is possible in TagLog. For example, to search for instances of the syntactic category (conj)unction, all we have to say is:

(113) !- provable(syncat(S,conj)).

Remember that the explicit existential quantifier in the predicate logic formula is implicit in the corresponding TagLog query.

Note that a certain amount of under-specification of property values corresponding to properties of the target segments is possible in those cases where terms denoting property values are complex. Thus, if we are interested in both singular and plural nouns, we write our query as in (114).

(114) !- provable(syncat(S,n(_))).

Furthermore, if we want to find the segments which are instances of *some* syntactic category, a ‘total under-specification’ is possible:

(115) !- provable(syncat(S,W)).

A *complex* search query is a search query involving several atomic formulas linked by ‘connectives’ like conjunction, disjunction or negation. Such queries will also often be complex in the sense that they will involve several levels of analysis. Consider the query in (116), for example:

(116) !- provable((utterance(S,customer), mood(S,interrogative),
function(S,question), segment_prefix(S,S1),
string(S1,"do"), subsegment(S,S2), syncat(S2,conj)).

What we are looking for here is an interrogative question, uttered by ‘customer’, which begins with an instance of “do” and which contains a conjunction. It is easy to imagine more complex queries, many of which are probably impossible to formulate with ‘traditional’ corpus search tools.

7.2.2 Searching for syntax

In Chapter 4, an overview of different ways of writing declarative grammars in the TagLog environment was given. As is often claimed, the appeal of a declarative grammar is its *bi-directionality*, i.e. its usability for generation as well as parsing (cf. Shieber, 1986).

Parsing has in fact been demonstrated already. The derivations in (64), (67), (71), (74), and (77) all demonstrate parsing in the sense that given a segment, the syntactical (or in some cases the syntactico-semantic) category of which it is an instance is derived. While parsing is certainly something that the corpus syntactician wants to be able to do, this is not so with generation. The corpus syntactician is usually interested in *searching* for certain grammatical constructions in text, rather than generating them from a grammar and checking them against his intuitions of grammaticality.

Fortunately, the way we have encoded the grammar gives us a way to search for segments. If we formulate a query, with the first argument uninstantiated, and the second argument instantiated to *np*, TagLog will find the position of the first instance of a noun phrase for us and instantiate the first argument to it:

```
(117)    !- provable(syncat(S, np)).
              S = 1-2
```

Then if we force TagLog to backtrack, it will find the next instance, and subsequently all the instances, of noun phrases:

```
(118)    !- provable(syncat(S, np)).
              S = 1-2 ;
              S = 3-4
```

It is easy to see that this search process relates to generation in a certain way. It works in the same direction, in that we start out with a category and end up with a string. The difference consists in the extra constraint that this string must be found in the text. Instead of asking, for a certain

category “is there a string in this *language* that is of this category?” we ask “is there a segment in this *text* that is of this category?”.

In the literature (e.g. in Voutilainen, 1993), one sometimes finds references to a kind of tool that goes under the name of a *phrase detector* (or *phrase finder*). The idea is that a phrase detector would be able to take a specification of a kind of phrase and then find all occurrences of it in a text. Clearly, searching as deduction might be seen as one possible, and nicely principled, step towards an implementation of a phrase recognizer. In Chapter 14, however, we will see that a phrase detector would have to be able to use more than just the traditional kind of grammar rules; some kind of constraints on the context is necessary as well.

7.2.3 Pointing to what is found

To make searching more practical, we use the TagLog built-in predicate *select/1* to select (i.e. highlight) the segment found, as in (119).

(119) `!- provable(syncat(S,conj)), select(S).`

Calling this goal will bind *S* to a segment expression and the subsequent call to *select/1* will select the segment denoted by this expression.

In case the argument of *select/1* is not instantiated to a segment when it is being called, but to another term (a constant, say), a free text search in the *Situation Theory window* is performed instead, for occurrences of that term. For example, suppose the Situation Theory window contains the clause in (120), modelling the existence of a person in the described or presupposed situation.

(120) `person(x34).`

If the call in (121) is performed, the symbol *x34* in the Context window will be selected.

(121) `!- provable(person(X)), select(X).`

As we will see in Section 7.4.3, this is how we can use an integrated grammar in order to investigate (and navigate) the relation between the segments of a text, and its content and context, represented as a situation theory.

7.2.4 Searching as a dialog

Note that the query in (119) can be regarded as a formalization of the natural language query: "Where is (the first instance of) a conjunction?" or as a formalization of the request: "Please find, and point out, (the first instance of) a conjunction!". The system replies by pointing to a segment. Thus, looking at the whole thing as a dialog, we have something along the lines of (122).

(122) U: *Please find, and point out, a* conjunction *in this text*
S : *Ok, here's one [pointing to a segment]*

As argued in Chapter 3, looking at particular user commands and system responses as turns in an ongoing conversation may help to clarify their connection to other user commands and system responses.

7.2.5 Searching and prediction

The notion of *prediction* relates to searching in the sense that the consequent of a universally quantified conditional can be said to predict a property of what we are going to find if we search for what is specified by the antecedent.

As an example, consider the statement in (123) again:

(123) `syncat(P0-P2,np) :- syncat(P0-P1,det), syncat(P1-P2,n).`

In Chapter 4, the meaning of this statement was said to be: "a determiner followed by a noun is always a noun phrase". What the statement *predicts* can be paraphrased as follows: "the next time you find a determiner followed by a noun, the two together will form a noun phrase."

In Chapter 12, we will see how the TagLog *Hypothesis Testing Tool* allows us to *check* a prediction made by a universal statement, by exploiting the fact that the consequent of such a statement amounts to a specification of a segment to search for, select and thus present to the user.

7.2.6 Precision and recall

When searching for segments matching a particular description, by means of a grammar for example, the system often finds too much, and/or too little. That is, it finds segments not in the target set of segments, and/or it fails to find segments that *are* in the target set. These problems are usually

referred to as lack of *precision*, and lack of *recall*, respectively (cf. Salton, 1989).

The exact definition of these measures is not crucial in this thesis (since no actual evaluation of searching is performed), but we can think of precision as the ratio between the number of target segments that were found and the total number of segments that were found, and by recall is meant the ratio between the number of target segments that were found and the total number of target segments.

Thus, a precision of less than 100% means that the system found something that is not regarded as a correct result, whereas a recall of less than 100% means that the system missed some of the desired segments.

We will return to these notions again in Chapter 14, as part of the discussion of certain errors that some TagLog tools seem to make, but which will turn out not to be errors for which the tools can be blamed.

7.3 The TagLog Search Tool

The TagLog *Search Tool* is a tool which combines the use of the *provable/1* and *select/1* predicates as presented in Section 7.2.3. The tool makes it easier to enter and execute a search specification, and provides dynamic links to both the Statement Browser and the TagLog Explanation Tool.

7.3.1 Search Tool layout

The Search Tool dialog box, depicted in Figure 25, consists of one text entry field and four buttons.

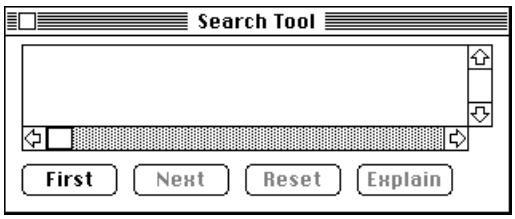


FIGURE 25. The TagLog Search Tool

Initially, the only button enabled is:

First: Find the first element in the set specified by the set-expression in the text entry field and select it in the Text Window.

But besides finding a matching segment, clicking this button enables three other buttons:

Next: Find the next instance. Subsequent clicks on this button will find the rest of the specified segments, one at a time.

Reset: Return to top level.

Explain: Activate the link to the TagLog Explanation Tool (see Section 7.3.4).

7.3.2 How to use the Search Tool

To search for a particular set of segments in the current text, proceed as follows:

1. Select the Search command from the tool menu.
2. Enter the specification {*S* | *Description*} in the Search Tool's text entry field, where *Description* describes each of the target segment *S*. (Section 7.3.3 defines a simplified notation that may be used instead).
3. Click the Find button.

The first matching segment will then be selected and highlighted in the text. In order to see the next matching segment:

4. Click the Next button.

To exit the search, e.g. in order to enter a different search expression:

5. Click the Reset button.

7.3.3 A simplified notation

The current version of the TagLog system implements a shorthand notation for a certain class of target set of segments. For example, instead of the set-expression

(124) {P0-P2| (syncat(P0-P2,pn) ; bstring(P0-P1,"the"), syncat(P2-P2,n))}

to be read "the set of segments that are either proper nouns *or* consists of "the" followed by a noun", we may write

(125) (syncat = pn ; bstring = "the", syncat = n)

Moreover, when values are unique to one property, i.e. when there are no other properties for which these values are possible, then we may write the specification as the even shorter:

(126) (pn ; “the”, n)

This is an example of how we might define a simple query notation ‘on top of’ the TagLog formalism. On the other hand, it is clear that there are set-expressions for which this particular notation cannot provide us with equivalent shorthand expressions.

7.3.4 Linking to other tools

There is a link from the Search Tool to the Statement Browser such that if the Statement Browser is open, it will automatically display the statements describing the segment selected by the Search Tool. For more information on the Statement Browser, see Chapter 8.

Clicking the Explain button activates the link to the TagLog Explanation Tool, i.e. if not already open, it opens this tool, and it produces an explanation, in the mode determined by the settings of the relevant controls in the Explanations Tool’s dialog box, of why the system considers the segment selected by the Search Tool to be an instance of the kind of segment searched for. See Chapter 13 for a more detailed description of the Explanation Tool.

7.4 Exemplifying Searching

7.4.1 Searching for instances of a basic string

Suppose that we want to search for an instance of the basic string “the” in the current text. We select the Search command from the Tool menu, enter the specification {*S* | *bstring(S,"the")*} in the Search Tool’s text entry field, and click the Find button.

The first instance of “the” (if such an instance exists) will then be selected and highlighted in the text. In order to see the next instance, we click the Next button, until we have seen enough, and then we click Reset.

It is true that this task could be performed by any word processor program, probably in a faster and simpler way. The generality and flexibility of the TagLog approach does not show until we try to find instances of properties other than strings.

7.4.2 Searching for adjacency-pair segments

In order to find instances of adjacency pairs, as given by the theory in Section 4.9, we only have to change the specification, the other steps are to be done exactly as before. So the search specification will now have to be {*S | d_syn(S, adj_pair)*}. By clicking the First button, the result we get display as in Figure 26.

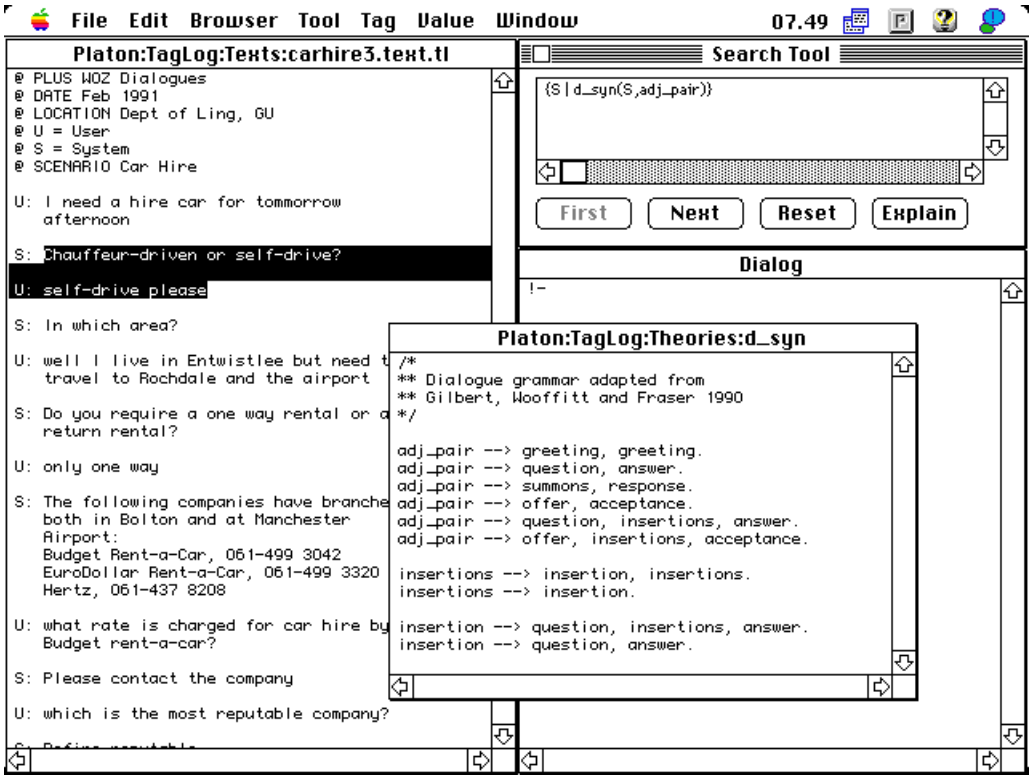


FIGURE 26. Finding an adjacency pair

7.4.3 Searching for errors and incompleteness

Suppose we have defined a couple of clauses like the following:

(127) inconsistent(S) :- syncat(S,C1), syncat(S,C2), C1 \= C2.
inconsistent(S) :- bsegment(S), unprovable(syncat(S,_)).

Then the following set-expression would allow us to search systematically for ambiguities or incompleteness.

(128) {S | inconsistent(S) }

The descriptions for the segments that are found are displayed in the Statement Browser – if it is open – where we may try to correct or complete them.

7.4.4 Tracing the Text-Content/Context relation

Many TagLog tools can be used to analyse the relation, in both directions, between text and content/context. I will exemplify this here with the use of the Search Tool. Suppose we have, as part of the current theory, the integrated grammar presented in Section 4.8. Suppose we select a segment in the Text window, and that we enter the set-expression in (129) in the Search Tool's text entry field.

(129) {X | selection(S), syncat(S,_:X) }

This expression should be read as follows: the set of entities (e.g. objects, events, states) *X* such that *X* is the interpretation of the segment *S* selected in the current text.

With a click on the Find button, the Search Tool is able to find and highlight, in the Situation Theory window, each occurrence of a symbol denoting the entity that is singled out as the interpretation of the segment selected in the Text window. This is illustrated in Figure 27, where the first token of *x18* has been selected. Subsequent clicks on the Next button will highlight the other occurrences of *x18*.

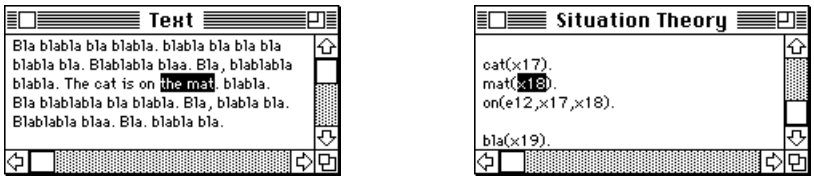


FIGURE 27. Tracing the text-content/context relation

Suppose, on the other hand, during inspection of the statements in the Situation Theory window we observe a name of an entity (*x17*, say), for which we would like to know which segments (if any) in the text that are connected to it by a relation of interpretation. We enter the set-expression

in (130) in the appropriate field in the Search Tool, and click the First button.

(130) {S | syncat(S,_:x17)}

The segment corresponding to the string “The cat” will become highlighted. Thus, both directions are accounted for.

7.5 Text in Co-text – Concordancing

In TagLog, searching is the basis for concordancing. In fact, a concordance can be described as a way of presenting the result of an exhaustive search.

7.5.1 The logic of concordancing

A concordance is declaratively specified by a set-expression denoting a set of segments, and is further determined by the setting of parameters that specify sorting order and where to print the concordance.

This approach to concordancing is much more powerful than traditional approaches: set-expressions may involve logical connectives like conjunction, disjunction and negation; subformulas in the set-expressions may refer to any value for any property (i.e. not only text strings, but also lexical categories, syntactic or functional categories, etc.); the statements expressed by the subformulas do not have to be explicitly given in the tagging process, but may be defined by universal statements given in a general theory (e.g. a grammar).

7.6 The Concordance Tool

The TagLog Concordance Tool, like other TagLog system tools, supports declarative specifications, is very general, and is dynamically linked to some of the other tools and presentation areas in the system.

A declarative specification of a concordance is given by writing a set-expression denoting a set of segments in the Concordance Tool’s text entry field. A TagLog concordance can be sorted in alphabetical order, or

in order of appearance, and can be written to a Live window, the Dialogue window, to the Macintosh clipboard, or to a file. These choices are made by setting the values of pop-up menus.

7.6.1 Concordance Tool layout

Selecting Concordance from the Tool menu brings up the dialog box in Figure 28, containing one text entry field, two pop-up menus, and one button.

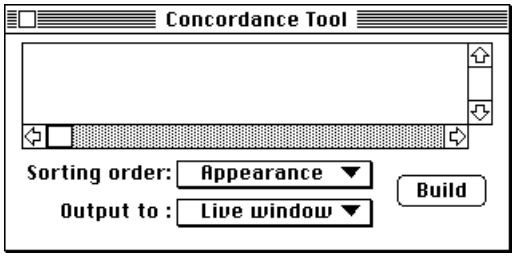


FIGURE 28. The Concordance Tool dialog box

The set-expression denoting the target segments is entered in the text entry field. The controls are as follows:

Sorting order: This pop-up menu has two possible values:

Alphabetical: The concordance is sorted in alphabetical order. This is the default value.

Appearance: The concordance is sorted in the order that the segments appear in the text.

Output to: This pop-up menu has four possible values:

Live window: The concordance is presented in the Live window. This is the default value.

Dialogue: The concordance is printed in the Dialogue window.

Clipboard: The concordance is sent to the clipboard, from which it can be pasted into other applications, such as a word processor.

File: The concordance is printed to a file specified by the user.

Build: This button will start building the concordance specified in the text entry field, sorted according to the setting of the Sorting Order menu,

and printed to the target specified by the current value of the Output To menu.

7.6.2 How to use the Concordance Tool

In order to build a concordance of segments in the current text that satisfy *Description*, proceed as follows:

1. Select the Concordance command from the Tool menu.
2. Enter the specification {*S* | *Description*} in the Concordance Tool's text entry field. (The simplified notation defined in Section 7.3.3 may be used here instead).
3. Make sure that the pop-up menus of sorting and output options are set according to your requirements.
4. Click the Build button.

7.6.3 Linking with other tools

The Concordance window is dynamically linked to the Text window and the Statement Browser. By clicking a particular line in the Concordance window the user can locate the corresponding segment in the main text, thus seeing a larger co-text than is possible in the concordance, which is limited to one line per target segment. The segment selected in this way can be further described in the usual way. As well as this, the statements describing the segment thus selected are displayed in the Statement Browser.

7.7 Concordancing Examples

7.7.1 A concordance of word form tokens

In order to build a concordance of the word "the" with respect to the current text, and present it in the Live window, we select the Concordance command from the Tool menu, enter the specification {*S* | *bstring(S, "the")*} in the text entry field, and click the Build button. The result of these steps is depicted in Figure 29.

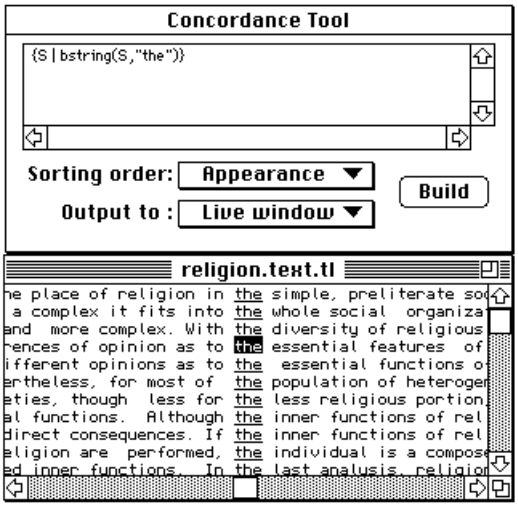


FIGURE 29. A concordance of “the”

Again, the task is trivial and could be performed by any concordance program; the TagLog approach does not reveal its power until we try more difficult things.

7.7.2 Concordancing based on non-string properties

Given a hand coded or automatically obtained description of the segments of a text, a concordance can be specified that picks out some of the segments. Figure 30 exemplifies this. Note the use of the logical connective *disjunction* in the specification as well, and note that the sorting order now has been set to Alphabetical.

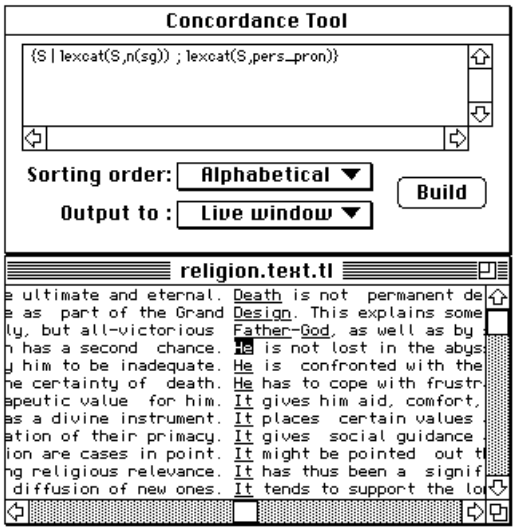


FIGURE 30. A concordance of nouns

7.7.3 Using a theory to build a concordance

A noun phrase grammar can be used to build a concordance of noun phrases. We select the Concordance command from the Tool menu, and enter the specification {*S* | *syncat(S,np)*} in the Concordance Tool's text entry field. We set the menu of output options to Live Window, and the Sorting order menu to Alphabetical. Finally, we click the Build button.

Note that this was done in *exactly* the same way as with strings (the ordinary type of concordance) and lexical categories, except that the specification was different. In this way, we would also be able to produce a concordance of adjacency pairs, as defined by the dialog grammar given in Section 4.9, a concordance of segments functioning as questions, as defined in Section 10.5.1, etc.

7.7.4 The Concordance Tool – Statement Browser link

As we learned in Section 7.3.4, the Concordance Tool and the Statement Browser can be dynamically linked via the Live Window. The link is established automatically as soon as the Concordance Tool's Live window and the Statement Browser are open simultaneously. For instance, when the Statement Browser is open in the context of the example in Section 7.7.2, we get the display depicted in Figure 31.

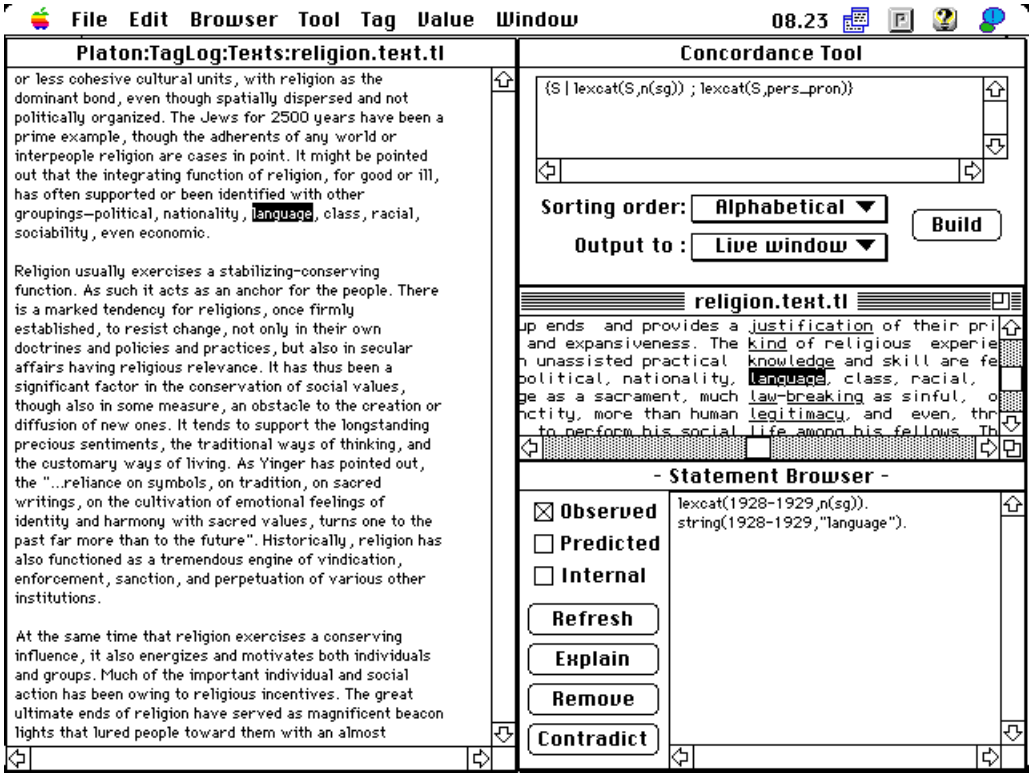


FIGURE 31. The Concordance Tool – Statement Browser link

Selection of a row in the Live Window concordance (by clicking on it with the mouse, or by using the Up- and Down arrow keys) will automatically update the Statement Browser so that the statements describing the segment corresponding to this row are displayed.

7.7.5 Concordances and stop lists

In order to reduce the size of concordances produced, it is not unusual for a traditional concordance program to be fed a so called *stop list*, i.e. a list of words (or categories of words) which the program is instructed *not* to include in the concordance. This works, of course, but the intended effect is also easy to obtain, in a more declarative and high-level way, by using negation in the specification of the concordance. For example, the specification in (131) produces a concordance of all the basic strings in the current text, except for all strings classified as form words.

(131) `{S | bstring(S,W), not form_word(W)}`

More correct – but useful only if the text has been tagged with syntactic categories – the specification in (132) uses the taxonomy given in Chapter 4 to exclude all closed class words from a concordance.

(132) `{S | syncat(S,open)}`

Thus, we see that at least some of the extra machinery and input requirements that is usually built into a concordance program is not needed if the language for specifying the target concordances is expressive enough.

7.8 Concordancing for Hand Coding

Sometimes we wish to describe (a subset of) some easily circumscribed set of segments in a particular way: instances of a particular string, for example, or instances of a particular syntactic category. It may then be more expedient to do this coding in a concordance corresponding to this set, instead of directly in the Text window. The result, in terms of updates to theories, is exactly the same, but the concordance provides a very good overview of the potential targets for coding, and thus increases the chance of assigning the codes in a well-informed and consistent way. Moreover, due to the dynamic link between the Concordance Tool and the Statement Browser, the latter tool can be used to monitor the coding process.

In Figure 32, the user has entered a set-expression in the Concordance Tool's text entry field, and has clicked the Build button. A concordance has been built, and the user has selected a line in it, resulting in the selection of the corresponding segment in the Text window and the display of the statements describing this segment in the Statement Browser. Next, the user is about to select a tag from the Tag menu, in order to elaborate the description of the current segment, perhaps by saying something about its function.

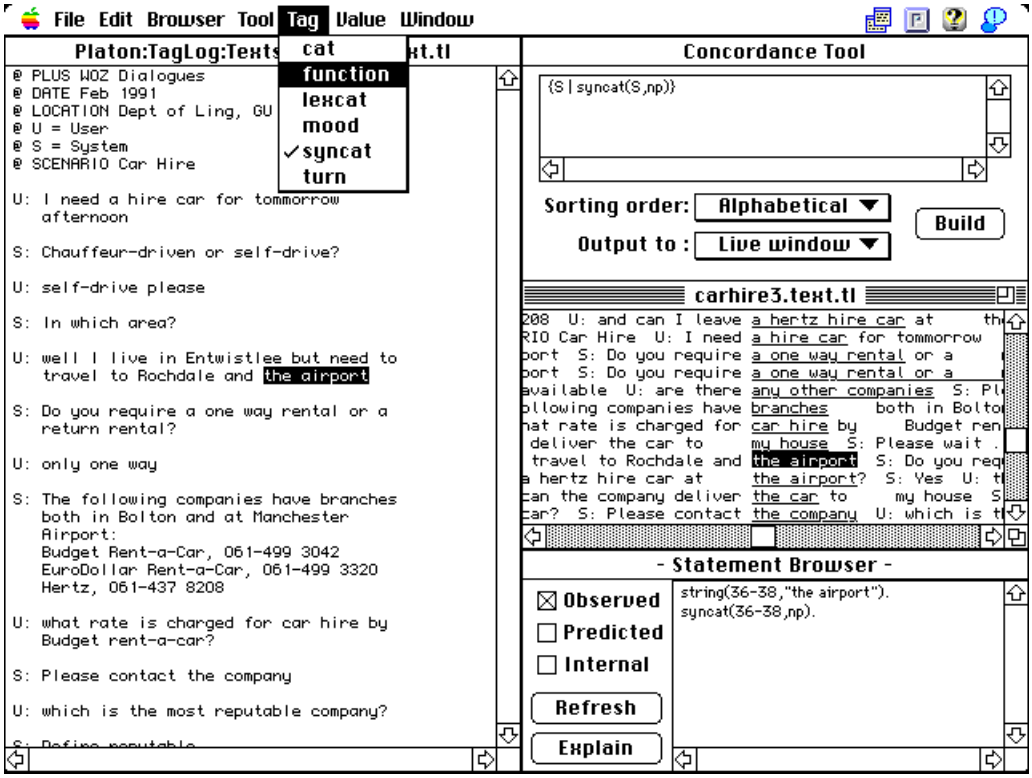


FIGURE 32. Concordancing for hand coding

7.9 Concordancing for Grammar Development

The set-expression in (133) denotes the set of segments in the current text that are instances of the syntactical category *np*.

(133) { S | syncat(S,np)}

By building a concordance on the basis of this specification, the user of the TagLog system can easily observe the implications of the grammar rules responsible for the description of noun phrases. If some of the

strings in the concordance are observed not to be noun phrases, he may have to consider changing the grammar.

But the user may also observe the impact of a grammar rule without actually adding it to the grammar first. The specification in (134) generates a concordance listing of all the instances of a preposition followed by a noun phrase in the current text.

(134) { P0-Pn | syncat(P0-Pi,prep), syncat(Pi-Pn,np)}

So here, if the grammar writer is not happy with what he sees in the corresponding concordance, he may decide that the addition of the rule *pp --> prep, np* to the grammar is not such a great idea after all.

Sometimes we want to be able to distinguish statements that have been added explicitly, on the basis of observation during a manual tagging process, from statements that follow deductively from a theory by means of one or more applications of rules. Fortunately, TagLog contains built-in predicates: *observed/1*, *unobserved/1*, *predicted/1*, and *unpredicted/1*, that will let us do exactly that. In (135) these predicates are used to build a concordance of all segments that are described manually as noun phrases, but where the grammar fails to recognize that they are noun phrases.

(135) { S | observed(syncat(S,np)), unpredicted(syncat(S,np))}

Another group of TagLog built-in predicates that are convenient for the corpus grammarian to know about is the *tree predicates*. Here I just exemplify the use of the predicates *parse_tree/2*, *node/2* and *parent/3*, and I will have to refer the reader to Appendix A for a closer presentation of these and other predicates in this family.

Suppose we want to build a concordance of all the verb phrases that contain at least one noun phrase. This set is specified as in (136).[1]

(136) {S | parse_tree(syncat(S,vp),Tree), node(Tree,np)}

Example (137) finally, specifies the set of segments *S* such that *S* is a phrase according to a particular grammar, and this phrase contains a verb phrase which stands in the *parent* relation to a noun phrase.

(137) {S | parse_tree(syncat(S,_),Tree), parent(Tree,np,vp)}

---

1. An alternative formulation, avoiding reference to a tree:
{S | syncat(S,vp), subsegment(S,S1), syncat(S1,np)}

7.10 The Treebank Approach

Before closing this chapter, let me say a few words on the subject of *treebanks*. A treebank is a corpus that has been annotated for skeletal syntactic structure. Examples of treebanks include the *Penn Treebank*, mentioned in Chapter 1.

A treebank can be implemented in TagLog simply by 'tagging' all segments in a text corresponding to sentences with trees, i.e. by using predications of the form *tree(Tree)*, where *Tree* encodes the syntactic structure of the sentence.[2] An example is given in (138).[3]

```
(138)   tree(syncat(3-7,s)/
                    (syncat(3-4,np)/
                        syncat(3-4,pn)/true,
                    syncat(4-7,vp)/
                        (syncat(4-5,v)/true,
                        syncat(5-7,np)/
                            (syncat(5-6,det)/true,
                            syncat(6-7,n)/true)))).
```

To find, to count, or to make a concordance of (say) all the noun phrase segments in such a treebank we may specify the target set of segments as follows:[4]

```
(139)   {S | tree(Tree), node(Tree,syncat(S,np))}
```

That is, what we are looking for here is the set of segments *S* such that *S* is a segment corresponding to a *np* node in a tree in the treebank.

Here, *node/2* is a built-in predicate in TagLog, defined as follows:

```
(140)   node(Node/Subtree,Node).
        node(Node/Subtrees,Node1) :-
            conj_member(Subtree,Subtrees),
            node(Subtree,Node1).
```

One advantage with the treebank approach is that it does not suffer from the imprecision problem mention in Section 7.2.6; provided the descrip-

2. But note that proof trees rather than parse trees are used here.

3. I am certainly not claiming that this is the most economical way to encode the tree.

4. Thus, this simple TagLog query could be compared with a tool for pattern matching against a skeletally parsed corpus such as the Penn treebank *tgrep*.

tion of the tree structures corresponding to segments is true, then what we find will always be what we were looking for.

Now the pre-processing part of building a treebank is not something that we would like to do manually. The treebanks referred to above have been constructed by means of an automatic parser, followed by a phase of manual disambiguation. In TagLog, a treebank can be obtained by 'caching' the results of *proof_tree/2*, and then manually removing all trees except the correct ones.

7.11 Summary and Conclusion

This chapter was concerned with two important aspects of the exploration of linguistic reality, namely *navigation* and *presentation*. (It was also shown how navigation and presentation in this sense is linked to observation and description in the sense of Chapter 6.)

It was shown that the TagLog system, through an implementation of the notion of *searching as deduction*, and by means of a built-in predicate that implements a kind of *deixis*, is able to find segments satisfying partial descriptions provided by the user and to point them out to him. The search functionality of TagLog is very general and powerful. In principle every segment that can be described by means of a TagLog expression can also be searched for.

Also, it should have become clear from this chapter that one of the traditional tools of the corpus linguist – the concordancer – can be designed on principles that are compatible with a logical approach to corpus linguistics, and that the result is a tool more powerful than traditional concordance programs. In the case of hand coding, syntax studies, conceptual modelling work, etc. such a concordance tool is indeed a useful thing to have.

CHAPTER 8

Browsing Theories

8.1 Introduction

As was explained in Chapter 2, a rather sharp division between theory and data is assumed in this thesis. With respect to this division, there are basically two ways to set the focus when using the TagLog system.

Either – and this was the theme of Chapter 7 – we start with a (possibly partial) description and ask for a matching piece of linguistic reality: “find this segment, take me there!”. Or else – and this will be the theme of this chapter – our point of departure is a particular segment of text: “what does the current theory predict about this segment?”. These questions can be combined, of course: “given this partial description, for each of the matching segments, how is it described by the current theory?”.

The descriptions referred to above are descriptions of *local* properties of text segments. But TagLog also deals with descriptions of *global* properties, i.e. properties of a whole text, or a whole corpus. Typically, the latter descriptions are of a quantitative nature. The topic of quantitative descrip-

tion is sufficiently different, however, to warrant a chapter of its own. Thus, it will be the topic of Chapter 9.

8.2 Retrieval by Deduction

To ask the TagLog system to store a sentence describing a particular segment, as explained in Chapter 6, makes sense only if the system is able to *retrieve* that description again on a later occasion, when the user asks it to do so.

To retrieve the value for a certain attribute for a particular segment in a traditional tagging system (assuming a tier-based mark-up language), we need only to look below the segment in question, on the tier corresponding to this attribute. Thus, for example, we can easily determine that the lexical category of the instance of the string “Mary” in Figure 33 is *pn*, by just looking immediately below that segment.

```
          ... John and  Mary ...
lexcat: ... pn   conj pn   ...
```

FIGURE 33. Traditional retrieval of tags

In the logical approach to corpus linguistics, not only searching, but also retrieval in the above sense, is performed by deduction. But – as was the case with searching – finding out whether a certain existentially quantified first order logic formula, such as the formula in (141), is provable or not is not enough:

(141) `|- ∃x[syncat(2-3,x)]`

Rather than a *yes* or a *no*, we need the values of the variables; thus, we need to use a constructive prover here too. In (142) the prover from Chapter 3 is used in order to prove an existentially quantified sentence. It succeeds, and thus binds the variable bound by the quantifier to the value of the property *syncat* for the segment *2-3*.

(142)
```
!- provable(syncat(2-3,X)).
      X = conj
```

So we see that two seemingly very different operations – searching for segments satisfying particular descriptions, and retrieval of the descriptions of particular segments – are supported by the same procedures, using the same declarative theory.

To make retrieval of the value of a certain property of a particular segment of text practical, it is of some import that the segment in questions can simply be pointed to, rather than referenced ‘by name’. As we have already seen, TagLog provides a built-in predicate for this purpose: *selection/1* binds a variable to the segment expression denoting the currently selected segment, and that expression can be used in a succeeding call, as follows:

```
(143)     !- selection(S), provable(syncat(S,X)).
                X = conj
```

8.3 The TagLog Statement Browser

The Statement Browser is designed to make it easy to inspect the current description of a particular segment of a text. The Statement Browser allows the user to select a segment in the text, and by clicking a button, or pressing a key, get a full description of this segment.

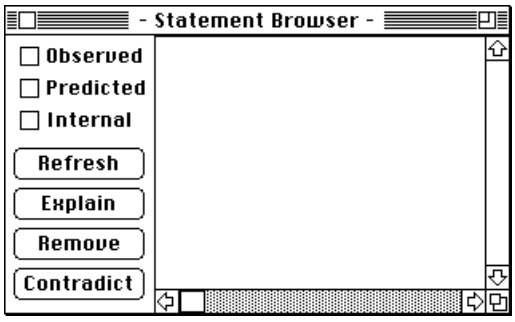


FIGURE 34. The TagLog Statement Browser

The Statement Browser can also be used for monitoring updates of the current theory (i.e. additions and deletions of sentences); it can be used for deleting statements, negating statements, demanding explanations of statements, and so on.

The Statement Browser is dynamically linked with the Search Tool and the Concordance Tool, and can thus be used to inspect descriptions of segments found by means of these tools.

The Statement Browser contains one display area, three checkboxes, and four buttons. The display area displays the sentences that describe the currently selected segment, in the mode determined by the settings of the checkboxes the meaning of which are given as follows:

Observed: Select this checkbox to display the observation statements that describe the currently selected segment.

Predicted: Select this checkbox to display derived statements about the currently selected segment.

Internal: Select this checkbox to display the statements describing every *subsegment* of the currently selected segment.

The following buttons perform various operations on the sentences in the display area:

Refresh: Clicking this button updates the information in the display area, for the segment currently selected.[1]

Remove: To remove the statement selected in the display area from the current theory, click this button (or push the Back Space key). The sentence becomes *unknown* in a sense to be explained in Chapter 10. This operation is currently possible only on explicitly represented sentences.

Contradict: To add a sentence which is the explicit negation of the selected sentence to the current theory, click this button. Hence, if the selected sentence has the non-negated form A, a sentence $\neg A$ is added to the current theory. If the selection is $\neg A$, the sentence A is added. In many cases, an inconsistency will arise, although not always, as will be seen in Chapter 14: the effect might also be to remove the contradicted sentence from the current theory (in the sense that it will no longer be derivable).

---

1. A short-cut: Pressing the Return key when the Text window is active has the same effect. So by selecting a segment, pressing the Return key, selecting a new segment, pressing the Return key, and so on. the descriptions of many segments in succession can be browsed, with no need to deactivate the Text window in between.

Explain: Clicking this button opens the Explanation Tool with a conjunction of the sentences selected in the display area, or, if the Explanation Tool is already open, produces an explanation in the currently selected mode.

8.3.1 Inspecting the statements describing a segment

As a simple example of the use of the Statement Browser, consider the text fragment:

(144) a complex

Suppose that we have described this text fragment as follows,

(145)
```
syncat(22-23,det).
syncat(23-24,n(sg)).
```

and that we later on want to recapitulate what we have said about it. In order to inspect the statements describing the segment corresponding to *complex* we select the segment and click the Refresh button. The Statement Browser will now look as in Figure 35.

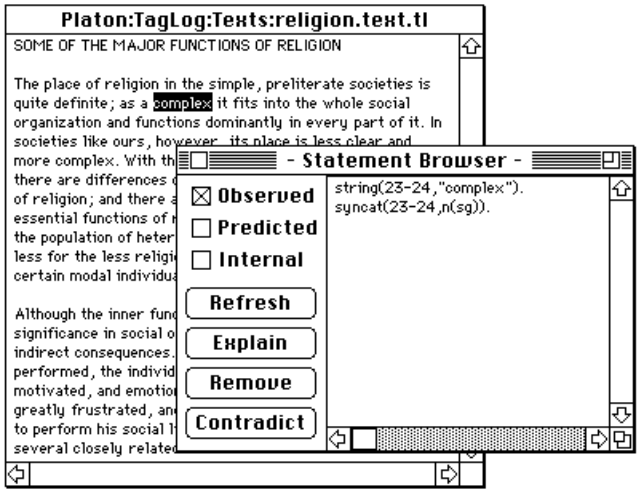


FIGURE 35. The Statement Browser in action

8.4 Parsing as Deduction

The idea of *parsing as deduction* was originated by Kowalski (1979), and Pereira and Warren (1983). Parsing as deduction has in fact already been

exemplified in Chapter 4, where the corresponding notion of *grammar as theory* was also explained. The current section will demonstrate how parsing as deduction by means of the Statement Browser is performed.

In the TagLog framework, parsing becomes a special case of retrieval of a description of a segment of text. Parsing a particular text segment is simply a question of asking the system what the current grammar theory *predicts* of the segment.

The Statement Browser is the right tool to use for this purpose: make sure the Predicted checkbox is marked, select a segment to parse in the Text window, and click the Refresh button (or push the Return key). The result is shown in the display area.

To continue the above example, given that we have written a grammar which includes the statement

(146) syncat(P0-P2,np(Num)) :- syncat(P0-P1,det), syncat(P1-P2,n(Num)).

the browser will display as in Figure 36, if the Predicted and Internal checkboxes are selected.

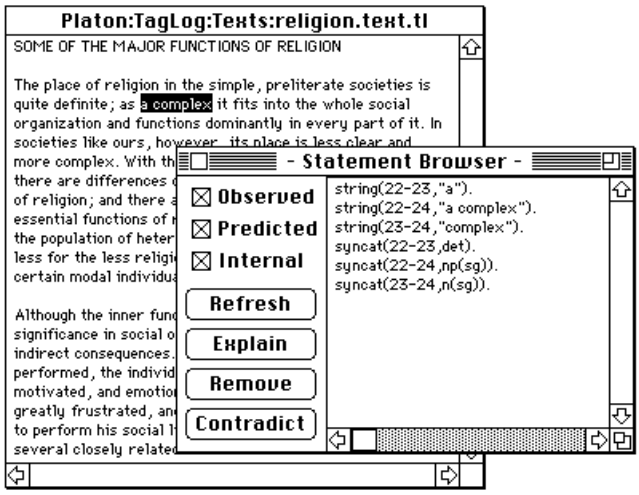


FIGURE 36. Parsing as deduction with the Statement Browser

Note that if we for some reason decide to delete the statement *syncat(23-24,n(sg))* from the current theory, for example by placing the insertion point on it and pressing the Backspace key, the statement *syncat(22-24,np(sg))* will disappear as well, since it is logically dependent on the former statement.

Let me also, by way of another example, introduce one of the greatest problem for parsing, namely *ambiguity.* Suppose we have a text that contains a segment instantiating the string in (147), and that we have selected that segment.

(147) time flies

Assume also we have the theory in (148).

(148)
```
s --> np, vp.
np --> n.
np --> n, n.
vp --> v.

n --> [time].
n --> [flies].
v --> [flies].
```

The Statement Browser will display as in Figure 37.

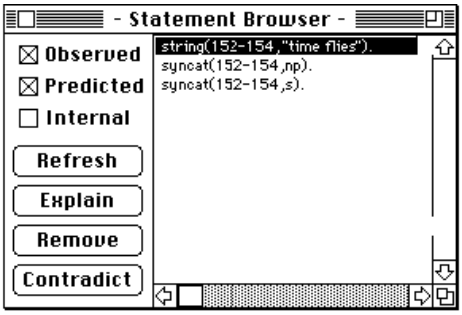


FIGURE 37. Ambiguity in the eyes of the Statement Browser

Thus, there seem to be two statements in the current theory: *syncat(152-154,s)* and *syncat(152-154,np)*, that describe the segment *152-154* in mutually incompatible ways. This is ambiguity.

To distinguish this kind of ambiguity from *structural* ambiguity, I will use the term *categorial* ambiguity. Whereas the term structural ambiguity refers to the case of a particular segment having more than one *analysis tree*, categorial ambiguity is the case of one segment being described as belonging to (at least) two different and incompatible grammatical *categories.*[2]

---

2. It seems to me that the distinction between categorial and structural ambiguity is blurred by the fact that if parse trees are treated as categories, every structural ambiguity is turned into a categorial ambiguity, and by the fact that, strictly speaking, every categorial ambiguity is also a structural ambiguity, albeit a very ‘shallow’ one. Still, I think the distinction is useful.

Presumably, one of the predictions made above must be false, and which one should be easy to determine given the context. As will be seen later, it is in line with the TagLog approach to regard ambiguity as a kind of *error* that a parser (i.e. a parsing procedure plus a particular grammar) makes.

Ambiguity will be discussed more thoroughly in Chapter 14. Let me just end this section by pointing to the possibility of using the Statement Browser to *disambiguate* the segment in question. Since the ambiguity of *152-154* is due to the categorial ambiguity of "flies", the most natural thing to do in this case is to select the segment *152-154*, select the erroneus sentence in the Statement Browser, and remove it by clicking the Remove button. Another possibility is to explicitly contradict the erroneus sentence by clicking the Contradict button. This will either make the inconsistency manifest, or, if the grammar rules are interpreted as default rules, the erroneus sentence will no longer be derivable (i.e. the inconsistency is resolved in favour of the explicitly added sentence). More on this in Chapter 14.

8.5 Alternative Theory Presentation Modes

Logic is extremely useful for the task at representing data, and for reasoning about them, but is not always adequate for presentational purposes. This shows in TagLog too.

One problem is that TagLog formulas expressing TagLog theories tend to be rather long, and what is more, they tend to be numerous. Hence, the Statement Browser is able to show only a small part of the current theory at a time.

Another problem is that it can be hard to trace the relation between the text and the formulas describing the text, e.g. it can be hard to see exactly to which segments particular segment-expressions refer. The TagLog mechanism for highlighting a segment denoted by a particular segment-expression helps, of course, but only for one segment expression at a time. In this respect, the way traditional 'tagging by labelling' approaches use spatial relations as part of their representational means has a certain advantage from a presentational point of view.

Viewing a traditionally tagged text is a bit like looking at a *map* in a direct sort of way, whereas inspecting a theory in the Statement Browser is more like looking at a long list of latitudes and longitudes, through a long and narrow tube. Great detail and precision, but not much overview.

It is important to stress, however, that a user of a logic-based system for corpus analysis is not confined to looking at the result of analyses in the form of sets of logical formulas. The focus of this section is on the use of systematic and consistent mappings between TagLog formulas and more 'visual' forms, that convey the same information as the formulas.

8.5.1 Alternative ways of presenting singular statements

Figure 38 indicates the mapping between TagLog formulas and *table-based*, *tier-based*, and *tail-based* representations. As should be clear from this picture, the mapping is quite straightforward, although it may not be quite so straightforward in cases where we have overlapping tags, for example.

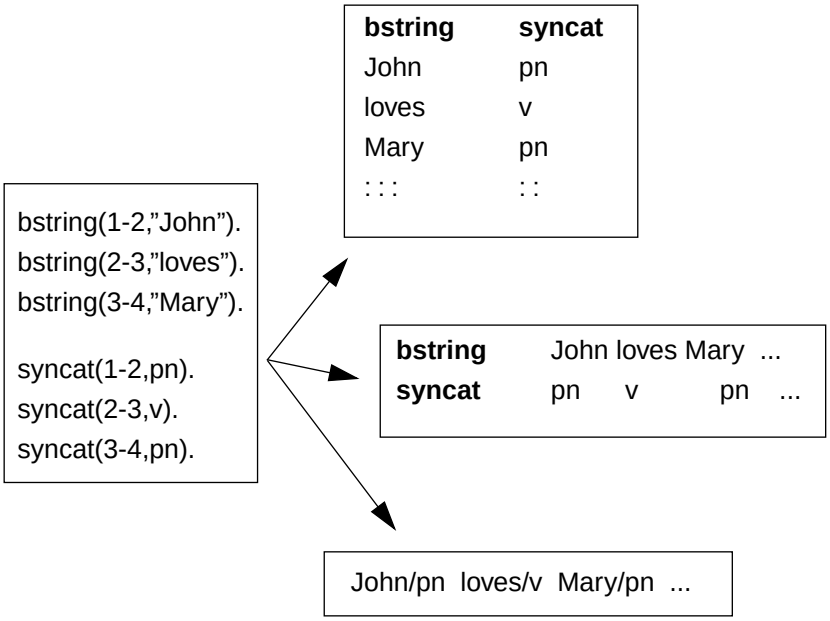


FIGURE 38. Mapping theories to other modes of presentation

In Section 8.6, I will present one of these alternative modes of presentation in detail: table-based presentation.

8.5.2 Alternative ways of presenting general statements

Problems of presentation are not confined to the singular statements. Universal statements can also be hard to read, and alternative ways of presenting the same information are of great interest.

We have already taken advantage of the fact that there is a well-known and natural way to present grammar statements, namely as grammar rules.

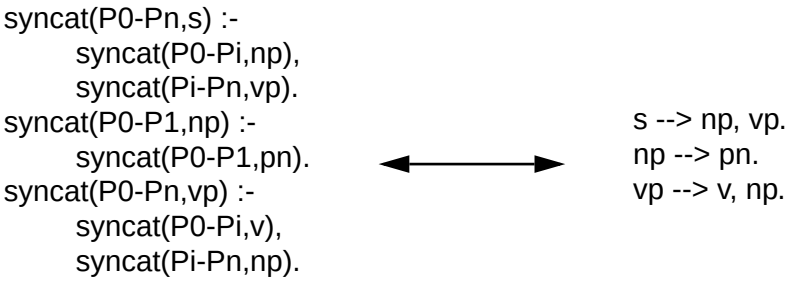


FIGURE 39. Mapping from clauses to grammar

We might also take advantage of the fact that a set of universally quantified sentences of a certain form can be mapped to a graph, and hence, give us a graphical way of presenting a taxonomy.

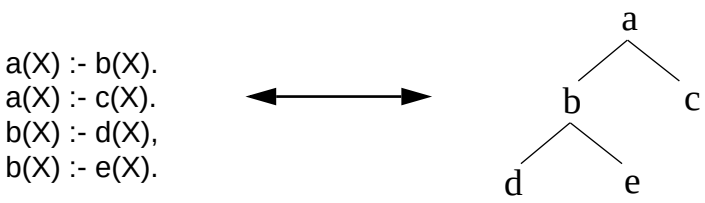


FIGURE 40. Mapping from clauses to a graphically represented tree

Other correspondences might exist, but this is not the place to explore them.

8.6 Table-Based Presentation

In corpus linguistics many uses are found for *tables* and *listings*. In this section, we will be looking at how word listings, tables of collocations, listings of statements, listings of lexica, etc., can be produced in TagLog. The next chapter will demonstrate the building of frequency tables.

Tables, as they are commonly understood, consist of columns and rows. From an *abstract* point of view, a TagLog table is a set of *n*-tuples, where *n* is the number of columns (and where *n*, in principle, is any number). Thus, a TagLog table can be specified by means of a set-expression.

8.6.1 A simple table in the abstract

As a first simple example of how TagLog abstractly characterizes a table, we use the corpus of transcriptions of doctor-patient dialogues prepared by Kós-Dienes (forthcoming), where feedback expression have been described with regard to their position in a larger utterance (‘external feedback position’).[3] Consider the following expression:

(149) {(W,P)| function(S,feedback), string(S,W), ext_fb_pos(S,P)}

This expression denotes the set of pairs *(W,P)* such that *W* is a word type and *P* is a structural position (initial, medial, final, or single) and there exists a segment *S* which is an instance of both *W* and *P* (remember that variables not bound by the matrix are existentially quantified). In more concrete terms, it denotes a two-column table where the first column consists of feedback words and the second column of the structural positions of these words. The expression does not say anything about how to compute this set, or how or where to present it. In order to produce the actual table, such an expression must be placed and put to use in the right environment.

3. See Appendix C for more detailed presentations of this and other corpora.

8.7 The TagLog Table Tool

Selecting Table from the Tool menu brings up the dialog box in Figure 41, containing one scrollable text entry field, two pop-up menus, and two buttons.

The set-expression denoting the set of tuples we are interested in is entered in the text entry field.

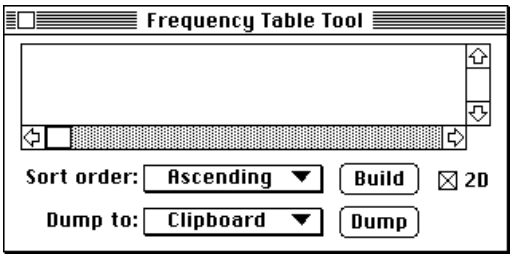


FIGURE 41. The TagLog Table Tool

The meaning of the buttons and menus is as follows:

Sorting order: This pop-up menu has two possible values:

Ascending: The table is sorted in ascending order with respect to the first column. This is the default value.

Descending: The table is sorted in descending order with respect to the first column.

Dump to: This pop-up menu has three possible values:

Dialog: The table is printed in the dialog window.

Clipboard: The table is sent to the clipboard, from which it can be pasted into other applications.

File: The table is printed in a file specified by the user.

Build: Clicking this button will start building the table specified in the text input field, sorted according to the setting of the Sort order menu, and printed to the target specified by the current value of the Output To menu.

We change the order between the columns by just changing the order between the variables in the set expression's matrix. Also, setting the

value of the Sorting order pop-up menu to Descending, gives us a different table.

8.8 Using the Table Tool

8.8.1 Simple table

By placing the following set-expression in the Table Tool's text entry field and then clicking the build button, we express a request for a table of positions of single word feedback items:

(150) {(W,P)| function(S,feedback), bstring(S,W), ext_fb_pos(S,P)}

resulting in the following table:

...	initial
...	single
då	final
ehh	initial
ehh	single
hm	initial
hmm	initial
hmm	single
hmmm	single
ja	final
ja	initial
ja	single
jah	initial
jah	single
jaha	final
jaha	initial
jaha	single
jaja	single
jasså	initial
jasså	single
jo	initial
mja	initial
mjaa	single
nja	initial
nä	initial
nä	single
näh	initial
näh	single
nähä	initial
nää	initial
tack	single
va	final
va	medial
vassa	single
åh	initial

8.8.2 Table with list-valued columns

Rather than having multiple occurrences of one word form in the first column, we may prefer to have only one occurrence, and instead have lists of values for feedback position in the second column. Then the built-in pred-

icate *ext/2* comes in handy; *ext(SetExpression,List)* is true iff *List* is the list of elements in the set denoted by *SetExpression*. The specification:

(151) {(W,L) | ext({C|function(S,feedback), bstring(S,W), ext_fb_pos(S,C)},L)

gives us the following table:

```
...       [initial,single]
då        [final]
ehh       [initial,single]
hm        [initial]
hmm       [initial,single]
hmmm      [single]
ja        [final,initial,single]
jah       [initial,single]
jaha      [final,initial,single]
jaja      [single]
jasså     [initial,single]
jo        [initial]
mja       [initial]
mjaa      [single]
nja       [initial]
nä        [initial,single]
näh       [initial,single]
nähä      [initial]
nää       [initial]
tack      [single]
va        [final,medial]
vassa     [single]
åh        [initial]
```

8.9 Summary and Conclusion

In the logical approach to corpus linguistics, retrieval of information about certain text segments – the current description of the segments – amounts to deduction in a theory encoding all the available information. A special case of this is *parsing as deduction*.

The TagLog Statement Browser is designed to support inspection of the current description of a particular segment of text. The Statement Browser allows the user to browse a full description of a segment selected in a text. It can also be used for monitoring updates, deleting statements, negating statements, and for demanding explanations of statements. The description is always presented in the form of a set of logical formulas.

This chapter has also stressed, however, that a user of a logic-based system for corpus analysis is not confined to looking at the result of analyses in the form of sets of logical formulas. Systematic and consistent mappings between TagLog formulas and more ‘visual’ forms exist, that convey the same information as the formulas, but in a more user friendly way.

CHAPTER 9

Quantitative Analysis

9.1 Introduction

Although it is for the support of qualitative rather than quantitative methods for description that TagLog has something novel to offer, it is my conviction that TagLog has great potential to become a very flexible tool for quantitative analysis as well. In this chapter, it is will be demonstrated how traditional quantitative methods for analysis of tagged corpora, such as simple counting, frequency listing and some common descriptive statistic measures can be accounted for, and in fact generalized, in terms of proof and set theoretic notions, and how they are implemented in the TagLog system.

In a sense, qualitative analysis is logically prior to quantitative description. Counting the instances of a class presupposes a classification of at least some of the segments of a text in terms of this class. Thus, the notion of quantitative description that I am going to argue for in this chapter presupposes the notion of qualitative description that have been developed throughout the preceding chapters.

The TagLog system is not, and will probably never become, a general statistics tool or a tool for creating attractive graphic presentations. But it supports export of data to other such tools, and thus provides a link in the chain of tools needed by the linguist. The exporting capability of the TagLog system will also be demonstrated in this chapter.

9.2 Counting in TagLog

To count in TagLog means to determine the cardinality of a set. So counting, like many other things in TagLog, involves deduction and some very elementary set theory.

The cardinality of a set is the number of distinct elements it contains.[1] To talk about the cardinality of a set, we use the relation *card/2*, which holds between the set and its cardinality, represented by an integer. For example, both *card({a,b},2)* and *card({a,a,b},2)* are true statements, while *card({a,b,c},2)* is false.

In order to count the elements of a particular set, all we need to do is to frame a set-expression denoting the set in question, instantiate the first argument of *card/2* to that expression, and query TagLog about the value of the second argument.

9.2.1 Counting types or counting tokens

As a simple example, consider the utterance in (152).[2]

(152) det är kallt det är det ja

On the one hand, if we count the particular tokens, this utterance contains seven words. The way to count them in TagLog is to ask:

(153)
```
!- card({S | bstring(S,W)},N).
```

which in this case will bind the variable *N* to 7.

On the other hand, if we count the number of different word *types*, the utterance contains only four words. This is the way to ask for it:

1. At least this is true for finite sets.

2. it is cold, that it is, yes

```
(154)    !- card({W | bstring(S,W)},N).
```

which binds *N* to 4.

Note that the difference between the expression in (153) and the expression in (154) is which variable is placed in the set expression's matrix and which variable is existentially quantified.

9.3 The TagLog Count Tool

Selecting the Count Tool from the Tool menu brings up the dialog box in Figure 42, containing one scrollable text entry field, and one button.

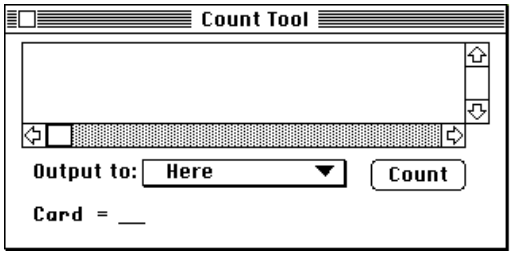


FIGURE 42. The TagLog Count Tool

Count: A click on this button will compute the cardinality of the set denoted by the set-expression entered in the text entry field.

9.4 Frequency Tabling

It often happens that we want to summarize categorial data by giving the *frequency distribution* over classes. This information is often presented in the form of a *frequency listing.*

From a TagLog point of view, a frequency listing with respect to a property *P* is a set of ordered pairs *(C,N)* where *C* is a possible value of *P,* and *N* is the cardinality for the set of instances of *C.*

As a concrete example, consider the expression

```
(155)    {(C,N) | card({S | lexcat(S,C)},N)}
```

This expression denotes the set of pairs *(C,N)* such that *C* is a lexical category and *N* is the cardinality of the set of instances of that category. Thus, what we have here is a word class frequency list.

As another example, the number of word tokens uttered by each participant in a dialogue can be specified as follows:

(156) {(R,N) | card({S | bstring(S,W), utterance(S1,R), subsegment(S1,S)},N)}

As a third example, consider the task of compiling a table of *biclass frequencies* (e.g. for use in a part-of-speech tagger). For our sample Brown corpus text, the specification

(157) {(N,C1,C2) | card({P0-P2|(lexcat(P0-P1,C1), lexcat(P1-P2,C2))},N)}

produces a table of the following kind (only the first fifteen rows are shown):

```
108  prep         det
101  adj          n(sg)
98   n(sg)        prep
85   det          adj
85   adj          n(pl)
82   det          n(sg)
77   n(sg)        comma
57   prep         adj
55   prep         n(sg)
54   n(pl)        prep
52   n(sg)        punct
42   comma        conj
37   adj          adj
36   n(sg)        v(sg3_pres)
35   n(pl)        comma
```

As usual in TagLog, we do not have to have our data *explicitly* tagged in order to refer to them. We may write universal statements by means of which certain facts about segments in our corpus follow deductively from other facts about them. In the case of athe corpus of transcriptions of doctor-patient dialogues prepared by Kós-Dienes (ibid.), certain segments have been described as having a *feedback function*, and certain segments have been described as *utterances*. Since the role of the *position* of feedback item in utterances was felt to raise interesting questions, a relation *ext_fb_pos/2*, exploiting the ordering between points, was defined:

(158)
```
ext_fb_pos(S,single) :-
        function(S,feedback),
        utterance(S,_).

ext_fb_pos(P0-Pi,initial) :-
        function(P0-Pi,feedback),
        utterance(P0-Pn,_),
        Pi < Pn.
```

```
ext_fb_pos(Pi-Pj,medial) :-
        function(Pi-Pj,feedback),
        utterance(P0-Pn,_),
        Pi > P0, Pj < Pn.
ext_fb_pos(Pj-Pn,final) :-
        function(Pj-Pn,feedback),
        utterance(P0-Pn,_),
        Pj > P0.
```

This made it possible to count the number of instances of each type of feedback position:

```
(159)   {(Pos,N) | card({S | ext_fb_pos(S,Pos)},N)}
```

Tables like the following could then easily be generated:

```
final      20
initial    58
medial     10
single     50
```

Sometimes we want to compare the way in which the frequencies of different categories are distributed over two or more groups. A *multi-dimensional table* might be a good solution for this. Such tables do not pose any special difficulties for TagLog. A three-dimensional table just means that the set-expression's matrix must be a triple, a four-dimensional table means it has to be a quadruple, etc.

For example, consider the following frequency list of feedback positions:

```
(160)   {(R,P,N) | card({S | ext_fb_pos(S,P),
            subsegment(S1,S), utterance(S1,R)},N)}
```

The result is a table of the following kind:

```
doctor    final      7
doctor    initial   26
doctor    medial     3
doctor    single    14
patient   final     13
patient   initial   32
patient   medial     7
patient   single    36
```

Often, a better way to *present* such a listing is the following:

```
          doctor  patient

final       7       13
initial    26       32
medial      3        7
single     14       36
```

Fortunately, such *multi-dimensional layouts* of multi-dimensional tables is also supported by the current version of TagLog. In many cases, this is the required form of input to a statistics or graphics program.

9.4.1 Exporting tables from TagLog to other programs

Diagrams can be helpful as a means of presenting a summary version of a collection of data. TagLog excels at collection and tabulation of data, but the current version of TagLog cannot produce high-quality diagrams, nor can it perform inferential statistical computations.

But TagLog tables, such as the ones we have introduced so far, can be used as the basis for contructing attractive graphs, and nice looking tables, or whatever the user wants to do with them, since TagLog tables can easily be *exported* to other programs. There are two ways to export a table: via the clipboard, or via a file.

In Figure 43, for example, the user has entered a set-expression in the Table Tool's query box, denoting the set of tuples *(C,N)* such that *C* is a syntactic category, and *N* is the cardinality of the set of segments that are instances of *C*. The user has then clicked the Build button in order to build the table, dumped it to the clipboard, and exported it to a charting program, where the graph has been produced.

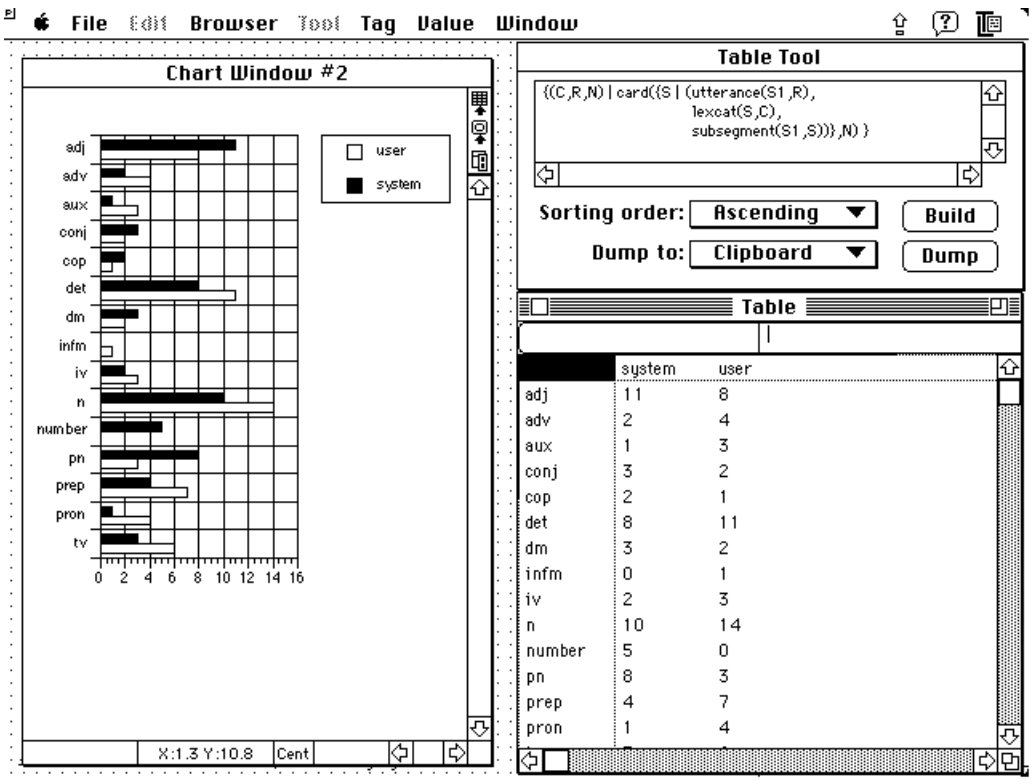


FIGURE 43. Tabling in TagLog

9.4.2 A word length graph

For showing dependencies between numerical values, a vertical bar diagram is usually a good choice. The relation between word length (x-axis) and frequency of words of that length (y-axis) in the doctor-patient corpus referred to above, i.e. the relation

(161) {(L,N) | card({S | bstring(S,W), string_length(W,L)},N)}

which is not particularly clear to present in a table, is easy to apprehend here:

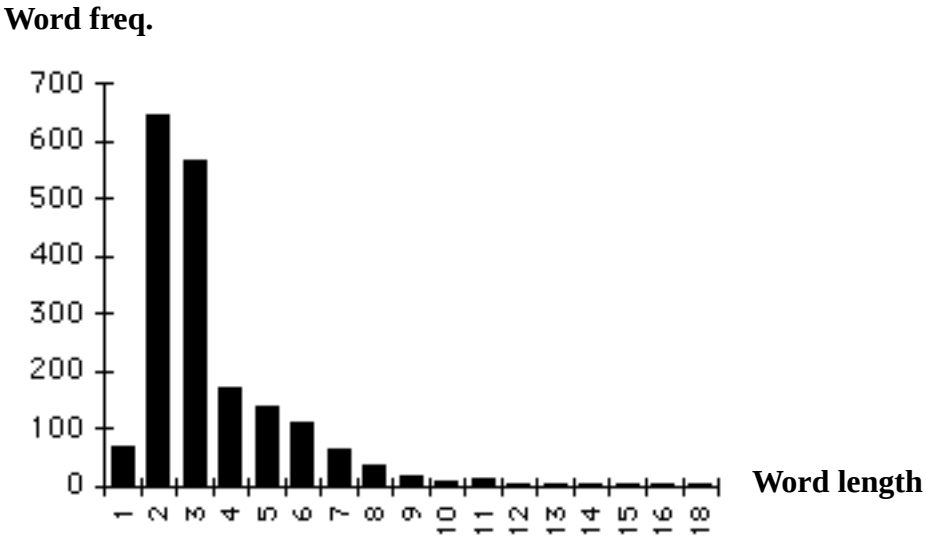


FIGURE 44. Diagram produced by TagLog and a graphing program

The following steps have been taken in order to produce this diagram: The set-expression denoting the table was entered in the Table Tool's text input field; the Sorting Order menu was set to Ascending; the Output To menu was set to Clipboard; the Build button was clicked; the graphing program was launched; the table was pasted into its spreadsheet, and the diagram was produced.

9.5 Descriptive Statistics in TagLog

Once we have a set of numerical data, we usually want to be able to summarize and present it in such a way that if there were any interesting pat-

terns to be found, we would observe them in just one quick glance. This is the business of descriptive statistics.[3]

The current version of TagLog includes a simple but general tool for computing the following descriptive statistical measures:

- Count
- Mean
- Standard deviation
- Median
- Maximum
- Minimum

This tool, like most other tools in TagLog, takes as its input a set-expression, only this time it is not a real set, but what is usually called a *bag*. Given a bag of integers, this tool computes the values for the above measures and presents them to the user in the form of a table.

Selecting the Statistics Tool from the Tool menu brings up the dialog box in Figure 45, containing one scrollable text entry field, one pop-up menu, and one button.

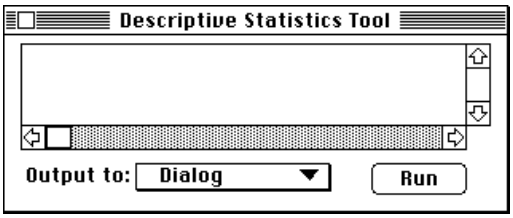


FIGURE 45. The TagLog Statistics Tool

The set-expression denoting a bag of numbers is entered in the text entry field.

Output to: This pop-up menu has two possible values:

Dialog: The table is printed in the dialog window.

---

3. A distinction is often made between *descriptive* statistics, which is concerned with the summary of (numerical) data, and *inferential* statistics, which tries to draw inferences from such data.

Clipboard: The table is sent to the clipboard, from which it can be pasted into other applications.

Run: Clicking this button will compute the statistics mentioned above and print a presentation of them, in the form of a table, to the target specified by the current value of the Output to menu.

9.5.1 Descriptive statistics, an example

The expression

(162) {N | utterance(S,doctor), segment_length(S,N)}

produces the table

Count	Mean	Std.Dev.	Median	Min	Max
83	13.2	13.9	10.0	1	73

and

(163) {N | utterance(S,patient), segment_length(S,N)}

produces

Count	Mean	Std.Dev.	Median	Min	Max
95	5.9	9.0	3.0	1	69

9.6 Other Uses of Tables

In this section, we use the Table Tool in order to generate listings of *high frequency words*, and tables of *collocations*, from a description of the doctor-patient dialogue introduced earlier in this chapter. Thus we add to the already very long list of functionalities that TagLog provides.

9.6.1 Listing high frequency words

It is sometimes useful to be able to determine the set of the most frequently occurring words in a text. Such a set is defined relative to a frequency threshold *F*, so that all words that appear more than *F* times in a text are included in the set. In TagLog, such a set (for *F*=25) can be characterized as follows:

(164) {W | card({S| bstring(S,W)},N), N>25}

```
är
så
och
men
jag
ja
har
du
det
de
att
```

But often, for example when the purpose of the query is to get a rough picture of the content of a text, it is necessary to exclude all *form words* from the list. Given an appropriate definition of *form_word/1*, we can simply say:

(165) {W | card({S | bstring(S,W), not(form_word(W))},N), N>4}

```
väl
viktigt
vet
tror
tack
par
ont
lite
hur
huden
går
gång
doktorn
bättre
benen
```

Now, it might be more elegant instead to define the notion of *content word* more explicitly,

(166)
```
content_word(S,W) :-
        bstring(S,W),
        not form_word(W).
```

and then to use this predicate in order to specify the table:

(167) {W | card({S | content_word(S,W)},N), N>4}

The result will be the same.

9.6.2 Collocations

Collocations can be thought of as arbitrary, often domain-dependent, recurrent word combinations that are lexically ‘clustered’ in the sense that one or several words of the collocations often suggests the rest of the collocation (cf. Smadja, 1993). In order to *find* collocations in a corpus, a reasonable strategy is to look for words that co-occur more frequently

than would be expected on statistical grounds only. For example, if we assume that a pair of words that appear within the span *M* more than *F* times are collocations, then the following expression defines the set of collocational pairs for *M* = 10 and *F* = 2:

```
(168)   {(W1,W2) | card({(S1,S2) |
                    content_word(S1,W1),
                    within_span(S1,S2,10),
                    content_word(S2,W2)}, N),
                    N > 2}
```

The *within_span/3* predicate is a built-in TagLog predicate defined in a way that makes *within_span(S1,S2,N)* true iff the length of the segment determined by the first point in the segment *S1* and the second point in the segment *S2* is less or equal to *N*.

With Kós-Dienes' text loaded, and by using (168) in the TagLog Table Tool, we are able to generate the following table:[4]

verkar	kroppen
verkar	hela
tror	bättre
ibland	finns
hela	kroppen
går	går
går	fint

It would be quite straightforward to work with a more constrained notion of collocation, e.g. to require that *S1* and *S2* stand in a particular syntactic relation. We would just have to add the relevant condition to the expression in (168).

9.7 Summary and Conclusion

In this chapter, 'theory-building', qualitative approaches have met quantitative analysis. Counting, frequency tabling, and descriptive statistics, have been demonstrated, and we have used the Table Tool in order to generate listings of high frequency words and collocations.

[4] Here is a translation into English:

seems	body
seems	whole
thinks	better
sometimes	is
whole	body
goes	goes
goes	well

CHAPTER 10

Logical Extensions

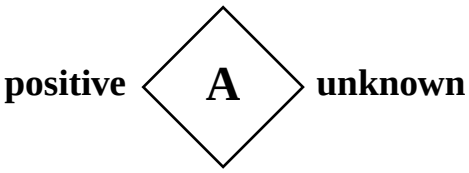
10.1 Introduction

For some of the ideas that we want to develop in the TagLog framework, it turns out that we will need to be able to deal with *negative*, *incomplete* and *inconsistent* information. In this chapter, we will therefore extend the Horn clause logic, very carefully, so that no efficiency is lost in the process, by adding (a limited form of) classical negation and various meta-logical operators. These extensions will provide us with more expressive power, but on the negative side, we will lose completeness, so that a great many conclusions that follow logically from a theory expressed in the language cannot be drawn.

10.2 Hard to Say in Horn Clause Logic

A major problem with the expressive power of pure Horn clause logic is that only *positive* information can be dealt with. Positive information can be explicitly represented and from the representations more positive information can be derived. Information that is *not* derivable is treated as *unknown* or *negative*, depending on whether an *open world* or a *closed world* is assumed.

The result of making the so called *Open World Assumption* can be depicted as follows:

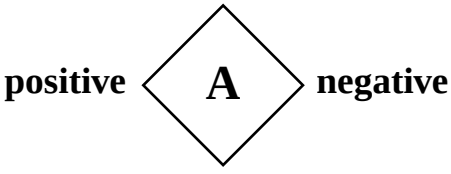


That is, a sentence *A* can either be determined to be true, or to have a truth value that is unknown.

Most ways of dealing with *negative* information within logic programming is related to the *Closed World Assumption* (CWA). The most well-known among these is *negation by failure*.

The CWA can be formulated as follows: Everything is known! A full description of (the relevant part of) the world is present! The knowledge of our chosen domain is complete!

It then follows, under the CWA, that if *A* cannot be inferred, then the truth value of *A* does not have to be treated simply as unknown. Rather, the *negation of A* can be safely assumed to hold. This is negation by failure. Thus, to 'say' that John is not a student, it suffices to say that he is a teacher, or in fact, to say nothing at all. As long as *student(john)* is not provable, the negation of *student(john)* can be assumed to hold. In other words, and with reference to the figure above and to the following figure, we have traded unknown information for negative information.



Under the CWA however, *nothing* can be left unsaid or undetermined, or deemed irrelevant; the unknown no longer exists as an epistemic category. But, as noted by DeRaedt (1992), the CWA is indeed strange in a system for interactive theory revision! Why learn if we know everything? Why update a theory assumed to be complete?

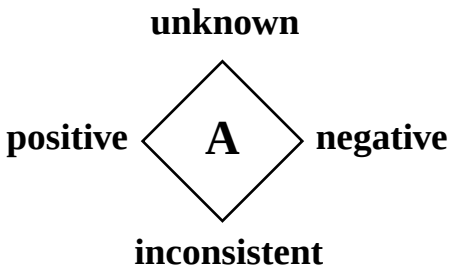
Clearly, in the case of TagLog, the CWA cannot be maintained, at least not in its unrestricted form. Rather, we have to accept that at a certain point in time, a given theory about a text may be incomplete in the sense explained above: there are certain sentences (in our language) that are not part of the theory, either because we do not know the truth value of them (yet), or because we do not care.

There are, however, a couple of problems connected with negation by failure, as well. First, negation by failure, regarded as a kind of negation, is rather limited. Only queries and antecedents of conditional clauses (i.e. rules) may contain negation. Unit clauses (i.e. facts) cannot be negated, nor can conditional clauses have negative conclusions. As a consequence, there are lots of things we simply cannot say.

A second problem is that we cannot capture the *difference* between ‘no’ and ‘don’t know’. As shown above, should *A* not be derivable, we can choose to regard that as a proof of the negation of *A* (if we make the closed world assumption) or we can choose to regard *A* as unknown (if we assume an open world), but we cannot capture the difference. This is unfortunate, since it seems reasonable to assume that a system for interactive theory revision must be able to ‘admit its ignorance’, and be clear and explicit about what it does not yet know.

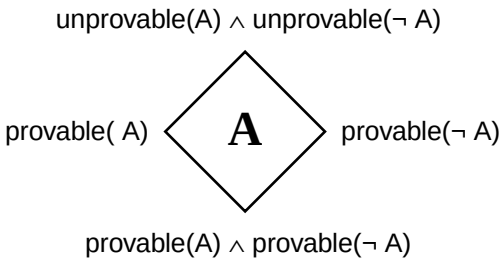
Thirdly, in a framework where the only negation is negation by failure, *inconsistency* cannot arise, since a formula can never be provable and not provable at the same time. That inconsistency cannot arise may sound like a good thing, but it’s not: inconsistency signals error – erroneous predictions, for example – and the detection of error, in its turn, is the most important motivation for revising a theory. Thus, to say no to the possibility of inconsistency is to say no to a major driving force behind theory development.

Hence, it seems that there are four kinds of information that we want the TagLog system to be able to distinguish: positive, negative, unknown, and inconsistent information. What we require can be depicted as follows:



To sum up, the distinction between the negative and the unknown presupposes the availability of explicit negation, as well as making the open world assumption. Moreover, inconsistency is only possible in a framework where explicit negative information is representable, so that a clash with explicit positive information becomes possible. Thus, Section 10.3 introduces a useful notion of explicit negation, and Section 10.4 introduces a battery of meta-predicates to reason about the known/unknown and the consistent/inconsistent. Their uses might be depicted as follows:

.



10.3 Explicit Negation

In this section we enhance the TagLog formalism with a symbol '¬' for a limited form of classical negation – called *explicit negation*. The idea comes from Gelfond and Lifschitz (1990).[1]

1. Gelfond and Lifschitz speak of their proposed extension as *classical negation*. But since the laws of contraposition do not hold for this kind of negation (i.e. *p :- q* and ¬ *p* does not automatically give you ¬ *q*), Kakas, Kowalski and Toni (1993) argue that the term *classical* is inappropriate. They prefer the term *explicit negation* instead, and so do I.

From the theorem prover's point of view, *A* and ¬ *A* are completely unrelated and each must be queried separately. In particular, a failure to prove *A* doesn't mean that ¬ *A* is proven, rather, ¬ *A* can be proven only if ¬ *A* is explicitly given as an axiom, or if there is a clause ¬ *A :- B*, and *B* can be proven.

Gelfond and Lifschitz provide a formal semantics for this extension. Moreover, and this is of great importance for our purpose, they show that the extension "hardly brings any new computational difficulties".[2]

In TagLog, there are at least two cases where the introduction of negated facts fits in quite naturally. First, if we know that a segment *S* has a property value *V1* for a property *P*, i.e. that *P(S,V1)* holds, we are often able to conclude, if *V2* is different from *V1*, that *S* does *not* have the value *V2* for *P*, i.e. that ¬ *P(S,V2)* holds. In particular, this happens when properties are restricted to *unique* values for single segments (for example, we may want to stipulate that a segment cannot be an instance of more than one lexical category). What we need for this kind of conclusion to be drawn formally is a way to express such unique-value restrictions. In TagLog, we solve that problem by introducing denials of the following kind:

(169) ¬ lexcat(S,V1) :- lexcat(S,V2), V1 \= V2.

Given such a sentence, ¬ *lexcat(2-3,n)* is derivable if, for example, *lexcat(2-3,det)* is derivable.

Second, it is clear that if a property *P* involved in a statement *A* is declared as being *closed*, and *A* is not derivable from the current theory, the negation of *A* can be derived. In TagLog we say:

(170) ¬ A :- current_text(T), functor(A,P,N), closed(T,P/N), unprovable(A).

For example, if we feel that we have ascribed a lexical category to every segment in a text 'text1' that can possibly have a lexical category, we may declare the property *lexcat* to be closed. In order to do so, we add the following statement to our theory:

---

2. Gelfond and Lifschitz (1990, p. 581). In fact, they show that under rather general conditions, simply by replacing each occurrence of ¬ P with a new predicate P', a theory containing explicit negation can be transformed into a an essentially equivalent theory that does not contain explicit negation. What this shows, in fact, is that explicit negation does not give us more expressive power, but 'only' a greater naturalness of expression, a greater modularity, etc.

(171) closed(text1,lexcat/2).

Then, ¬ *lexcat(2-3,n)*, ¬ *lexcat(2-3,adv)*, ¬ *lexcat(2-3,conj)*, etc. are derivable if *lexcat(2-3,n)*, *lexcat(2-3,adv)*, *lexcat(2-3,conj)*, etc. are not.

This is a special case of what in the literature has sometimes been called *selective CWA* – a restricted form of the Closed World Assumption.

10.4 Meta-Logic

A meta-logic is a logic by means of which we may refer to sentences of another language (the object-language). In this section, we discuss a meta-logical extension of the Horn clause language given in Chapter 3 that allows us to 'talk about' and manipulate Horn clause language sentences.

What I have in mind here is what has been called a *mono-lingual* meta-logic (van Harmelen, 1989). The object-language and the meta-language are one and the same language. Object-level predicates are regarded as meta-level function symbols. No distinction between object-level variables (ranging over object-level terms) and meta-level variables (ranging over object-level formulas) is made.[3]

It will be demonstrated in this section, and in the next, that such a meta-logical extension plays an important role in the TagLog approach to integrated syntax, semantics and pragmatics, and that it yields a simple implementation of non-monotonic reasoning.

The idea is to make the provability and unprovability relations of the object language available at the meta-language level. These are the basic relations:

- *provable(A)* – is true if *A* is provable in the current theory
- *unprovable(A)* – is true if *A* is not provable in the current theory

3. There are certain theoretical problems connected with this kind of meta-logic that does not make a clear distinction between object level and meta-level (Hill & Lloyd, 1989). (For one thing, we want *?- provable(p(X))* to mean *?- provable(∃(X)p(X))*, but it means *?- ∃(X)provable(p(X))*!). In practice, however, this kind of meta-programming is widely used, and does not lead to problems if one is careful.

Then, in terms of these relations, or as modifications of them, other useful predicates are defined, such as:

- *predicted(A)* – is true iff *A* is predicted by the current theory
- *observed(A)* – is true iff *A* is a description of an observation
- *consistent(A)* – is true iff *A* is consistent with the current theory
- *unknown(A)* – is true iff *A* is unknown, or undecidable, in the current theory
- *provable(A,Proof)* – is true iff *A* is provable in the current theory and *Proof* is a proof of *A*
- *card({ X |A }, N)* – is true iff *N* is the cardinality of the set of *X* such that *A* (and note that *X* occurs in *A*) is provable in the current theory

10.4.1 The *provable/1* predicate

Below, a TagLog theorem-prover written in Prolog is given. Readers familiar with Prolog programming techniques will recognize this as a so called 'vanilla' meta-interpreter.[4]

(172)
```
provable(true).
provable(A) :-
        builtin_predicate(A), !,
        call(A).
provable(A) :-
        clause(A,B),
        provable(B).
provable((A,B)) :-
        provable(A),
        provable(B).
```

Here's the declarative meaning of the four clauses comprising the interpreter: The formula *true* is provable, unconditionally. A built-in predicate is provable if it can be executed. A literal *A* is provable if a clause *A :- B* can be found and *B* is provable. A conjunction *(A,B)* is provable if *A* is provable, and *B* is provable.

From a procedural point of view, it can be understood as follows: In the call of the *clause/2* predicate, *B* gets bound to the body of a non-unit

4. A major reason for presenting this meta-interpreter here is that in sections to come, it will be extended in various ways in order to handle the generation of proofs needed for explanations, and in order to handle abductive reasoning.

clause the head of which matches (unifies with) with *A*, or *B* gets bound to *true* if there is a unit clause that matches with *A*. Unification is handled implicitly. The *clause/2* predicate also takes care of the systematic top-to-bottom search through the set of clauses constituting the current theory. Note that the last clause of the *provable/1* predicate implements the Prolog leftmost literal computation rule.

Given the theory in (173), an example of the use of the prover is given in (174).

(173)
```
human(X) :- featherless(X), biped(X).
featherless(socrates).
biped(socrates).
```

(174)
```
!- provable(human(socrates)).
     yes
```

Note that although the program in (172) is able to handle *all* the deductive reasoning required in TagLog, it will not be used for this purpose. As pointed out before, one may regard the TagLog system as an extension of a Prolog system, and most of the required reasoning takes place on the level where TagLog as such is implemented: the Prolog level. For the basic reasoning tasks, TagLog cannot afford the overhead of a meta-interpreter. Hence, for most tasks, the definition in (175) is used instead, transforming a meta-level query into an object-level query.

(175)
```
provable(A) :- A.
```

This is what is usually referred to as a *reflection rule*.

10.4.2 The *unprovable/1* predicate

TagLog, just like most implementations of Prolog, provides a meta-logical operator *unprovable/1* called (somewhat misleading, as we shall see) *negation by failure*. The semantics of this operator is as follows:

(176) unprovable(A) succeeds iff A fails (finitely)

This rule can be implemented very efficiently – to prove a goal *unprovable(A)*, the prover merely tries to prove *A* and responds to the outcome as described in (176). The Prolog code in (177) implements this behaviour.[5]

---

5. In order to guarantee soundness and completeness of negation by failure, *A* must be variable free when selected for execution (cf. Lloyd, 1987).

```
(177)   unprovable(A) :- provable(A) -> fail ; true.
```

The *unprovable* can be regarded as a form of negation only under the *closed world assumption*.

Again, the difference between *unprovable(A*) and *provable(¬ A)* is essential whenever we cannot assume that the available positive information about *A* is complete. This happens all the time in TagLog. For example, *A* can fail to be complete when the description of a particular text is still under development.

10.4.3 The predicates *observed/1* and *predicted/1*

The *provable/1* predicate can distinguish what is provable from what is not. But should *A* be an explicit description of a fact, or *A* follow from the universal theory, *provable(A)* succeeds just the same. Hence, *provable/1* does not distinguish data descriptions from the theorems of a universal theory.

But a separation between theory and data – stricter than in Prolog – *can* be maintained, so that predictions generated by a theory on the one hand and descriptions of data on the other can be kept apart when the need arises.

We distinguish between two ways in which a sentence *A* can be provable: *A* can be *predicted*, or *A* can be *observed*. These are mutually exclusive cases.

Consider the following clauses:

```
(178)   p(X) :- q(X).
        q(a).
```

Two sentences are provable from (178): *q(a)* and *p(a)*. But these sentences differ in that *q(a)* is an explicit representation of data, describing an observed fact, whereas *p(a)* is a prediction generated by means of a universally quantified implication.

Predicates that check these properties of sentences can be defined as follows:

```
(179)   observed(A) :-
                clause(A,true).
```

```
(180)   unobserved(A) :-
            not clause(A,true),
            not clause(¬ A,true).

(181)   predicted(A) :-
            provable(A),
            not observed(A).

(182)   unpredicted(A) :-
            clause(A,B),
            unprovable(B).
```

Thus, with respect to the theory in (178) we have:

```
(183)   observed(q(a)).
        yes

(184)   observed(p(a)).
        no

(185)   predicted(q(a)).
        no

(186)   predicted(p(a)).
        yes
```

Note that every sentence that is provable is either predicted or observed.

10.4.4 The *consistent/1* predicate

A theory *T* is *consistent* iff, for no formula *A*, both *A* and ¬ *A* are derivable from it. A Horn clause theory cannot be inconsistent, but as soon as we introduce explicit negation, we invite inconsistencies to arise as well.

To deal with consistency, the current implementation of TagLog has two built-in meta predicates: *consistent/1* and *consistent/*0.

Often, a *modification* of the current theory introduces an inconsistency that was not there before. The idea behind *consistent/1* is to try to detect an inconsistency introduced by a modification *before* actually performing the modification.

The semantics of the *consistent/1* predicate is as follows: *consistent(A)* is true iff the explicit negation of *A* cannot be proved. This immediately suggests the implementation in (187).

```
(187)   consistent(A) :- unprovable( ¬ A ).
```

A couple of comments are in order at this point. First, note that *A* must be variable free, since the soundness of *unprovable/1* cannot be guaranteed otherwise (cf. footnote on page 178).

Secondly, note that the specification above (and thus the implementation of it) is not complete. A better specification might be the following: *consistent(A)* is true iff the explicit negation of none of the *consequences* of *A* can be proved. The latter specification suggests a forward chaining implementation, more complicated (and less efficient) than the implementation in (187). It would be beyond the scope of this thesis to present such an implementation here, and in any case, the implementation in (187) is good enough for our purposes.

Note that *consistent/1* does not check consistency of the whole current theory. Rather, the idea is that if the current theory is consistent *before* adding *A* to it, then, if *consistent(A)* is true, it is guaranteed to be consistent *afterwards* as well.

TagLog also contains a predicate for the global checking of consistency: *consistent/0* is true iff the current theory does not allow a statement and its negation to be proved.

10.4.5 The known and the unknown

The *known/1* and the *unknown/1* predicates are not particularly useful, as far as the present work is concerned. Nevertheless, for completeness, the definitions are presented below.

(188)
```
known(A) :-
        (       provable(A)
        ;       provable(¬ A)
        ).
```

(189)
```
unknown(A) :-
        unprovable(A),
        unprovable(¬ A).
```

10.4.6 Meta-logic summary

Finally, Table 2 summarizes the semantics of the meta-predicates explained in this section. For each meta-predicate *P* discussed above, the

calls *!- P(p(a))* and *!- P(q(a))* are executed in three different theories, and the outcome – *yes* or *no* – is registered.

Meta-logical test	p(X) :- q(X). q(a).	p(X) :- q(X). q(a). p(a).	p(X) :- q(X). q(a). ¬ p(a).
?- provable(p(a)).	yes	yes	yes
?- provable(q(a)).	yes	yes	yes
?- unprovable(p(a)).	no	no	no
?- unprovable(q(a)).	no	no	no
?- observed(p(a)).	no	yes	no
?- observed(q(a)).	yes	yes	yes
?- predicted(p(a)).	yes	no	yes
?- predicted(q(a)).	no	no	no
?- falsified(p(a)).	no	no	yes
?- falsified(q(a)).	no	no	no
?- consistent(p(a)).	yes	yes	no
?- consistent(q(a)).	yes	yes	yes
?- known(p(a)).	yes	yes	yes
?- known(q(a)).	yes	yes	yes
?- unknown(p(a)).	no	no	no
?- unknown(q(a)).	no	no	no

TABLE 2. An overview of the semantics of TagLog meta-predicates

10.5 Non-Monotonic Reasoning

In a *non-monotonic* inference scheme, the validity of certain conclusions may depend on certain sentences *not* being theorems of the current theory. Thus, in a non-monotonic inference scheme, the acquisition of new knowledge can cause old theorems to be discarded. On the other hand, in a *monotonic* inference scheme, new axioms never invalidate previous theorems.

In this section, a technique for performing non-monotonic reasoning in TagLog is presented, the applications of which will be presented in sections to come.

By means of the meta-logical extensions of the TagLog formalism, it is possible to refer to the state of the current theory and thus to what is currently known. In particular, by means of the *consistent/1* predicate, we

can check the consistency of adding a new (variable free, atomic) formula to the current theory. Furthermore, by inserting a call to *consistent(A)* in the antecedent of a conditional, it is possible to make the applicability of the conditional depend on whether the current theory, updated with *A*, would be consistent or not.

Typically then, clauses with positive consequents, and with an occurrence of *consistent/1* in the antecedent, represent general rules, whereas clauses with negative conclusions represent exceptions. For example, in the following theory:

(190)
```
can_fly(X) :- bird(X), consistent(can_fly(X)).
¬ can_fly(X) :- penguin(X).
bird(tweety).
bird(pablo).
penguin(pablo).
```

it is possible to conclude

(191)
```
can_fly(tweety).
¬ can_fly(pablo).
```

but not

(192)
```
can_fly(pablo).
```

10.5.1 Describing communicative functions in TagLog

In this section, as an example of the use of the meta-logical approach to non-monotonic reasoning explained in Section 10.5, a set of simple rules, similar in spirit to the rules discussed by Robbert-Jan Beun in (Beun, 1989) for recognizing the communicative function of an utterance, are given. Beun's rules are formulated in Default Logic (Reiter, 1980) but such rules can easily be expressed using the technique given above. A rule saying that a declarative sentence is a statement by default can be expressed as follows:

(193)
```
function(S,statement) :-
        sentence_type(S,declarative),
        consistent(function(S,statement)).
```

If we want to say that, by default, a declarative sentence which is not a statement, is a question, we can simply write:

(194) function(S,question) :-
sentence_type(S,declarative),
¬ function(S,statement),
consistent(function(S,question)).

With the following rule, we are saying that an interrogative sentence is, by necessity, a question (i.e. this is not a default rule):

(195) function(S,question) :-
sentence_type(S,interrogative).

A segment *S* with content *C* uttered by *X* to *Y* is not a statement if *Y* is believed by *X* to be an expert on *C*:

(196) ¬ function(S,statement) :-
utterance(S,X),
partner(X,Y),
content(S,C),
believes(X,expert_on(Y,C)).

We also want to stipulate that the system believes that everyone is an expert on his own desires:

(197) believes(sys,expert_on(X,want(X,_))).

Here is a simple made-up two-turn dialogue – an idealized version of a piece of a PLUS dialogue – between an information provider and an information seeker – a system and a user. Thus, the system and the user are ‘communicative partners’:

(198) partner(sys,user).
partner(user,sys).

Here is the dialogue:

(199) S : you want a car
U: yes

Here is a some more information about the dialogue:

(200) utterance(1-5,sys).
utterance(5-6,user).
sentence_type(1-5,declarative).
sentence_type(5-6,declarative).
content(1-5,want(user,car)).
content(5-6,want(user,car)).

Now, even though it is declarative in its form, we may use the rule in (194) to infer that this occurrence of *you want a car* is a question:

```
(201)    !- provable(function(1-5,question)).
                yes
```

Thus, what we have proved is that, since it is a declarative sentence, since it is not a statement, and since it is consistent with the rest of what we know, this instance of *you want a car* is a question. We could prove that it is not a statement since it is uttered by someone (the system) who believes that his communicative partner (the user) is an expert on the content of the utterance (the user's own desires).

10.6 Extensions Applied

Meta-logical extensions, explicit negation, as well as the ability to explicitly represent negative information in combination with the meta-logical extensions, have many uses. Although I did not explicitly identify them as such then, we have in fact seen some of these uses already. And more is to come. Let me list them all here, with pointers to the relevant sections.

- Explicit negation enables the user to formulate some negative information about a text, even *before* any positive information is described (see the present chapter).
- Explicit negation enables the user to *explicitly* contradict a prediction made by the system (see Section 8.3).
- Explicit negation is used to represent exceptions to general rules (see Section 10.5).
- The *provable/1* meta-predicate can be used in support of an approach to integrated grammar in the spirit of Hobbs et al. 1993 (See Section 4.8.2).
- The *consistent/1* meta-predicate is used to avoid unintended updates during hand-coding of a text (see Section 6.5.2).
- The *consistent/1* meta-predicate is used in order to restrict abduction (see Section 13.5).
- The meta-predicates are used to allow the distinction between predictive and falsifying instances. This is a special case of allowing the distinction between 'don't know' and 'no' (see Chapter 12).

- The meta-predicate *unprovable/1* is used to implement a restricted version of the Closed World Assumption, called *selective* CWA (see the present chapter).
- The meta-logical approach to non-monotonic reasoning is used to infer communicative function (speech-act) from sentence type (mood) (see the present chapter).
- The non-monotonic reasoning capability is used to make grammars sensitive to context, for parsing, searching and automatic tagging (see Section 11.6 and Section 14.4.2).

This list emphasizes the claim made by Kakas, Kowalski and Toni (1993), that the combination of explicit negation and negation as failure is "very useful for knowledge representation in general".

10.7 Summary and Conclusion

Certain things that we would like to be able to do in TagLog require extensions of the available means of expressing theories about texts, especially with regard to *negative*, *incomplete* and *inconsistent* information. This chapter has discussed two such extensions:

- explicit negation
- meta-level reasoning

Thus, the expressive power of the logic presented in Chapter 3 is significantly raised for a very low cost in terms of loss of efficiency.

Kowalski (1990) argues that Horn clause logic, augmented with such extensions as abduction, integrity constraints, and meta-level reasoning is sufficient for knowledge representation and knowledge assimilation purposes, and that the extension to full first order logic may be unnecessary. It is in that spirit the TagLog approach has been developed.

CHAPTER 11

Automatic Tagging

11.1 Introduction

Hand coding of text is a laborious, error prone, and time-consuming task, and there are a number of advantages, especially regarding speed and consistency, with developing ways to perform the tagging, as far as possible, automatically.[1]

While we are far from being able to do automatic tagging on the level of semantics or pragmatics – fully automatic speech-act tagging is out of the question, for example – the current state of the art of automatic tagging allows part-of-speech tagging with some success. Consequently, this section will concentrate on automatic tagging methods applied to part-of-speech tagging. Keep in mind, however, that they can, at least in principle, be used for other things as well.

1. As regards consistency, it should be noted however, that there is nothing in principle that stops us from checking this property automatically even for a hand coded text.

A part-of-speech tagged text is a text in which each word occurrence is classified in accordance with some system of lexical categories. Usually, the idea is to have one and only one tag per word occurrence. Now, the large number of homographs in most natural languages poses great problems for an automatic part-of-speech tagging approach. To see these problems, consider the statement in (84), repeated here as (202). This can be seen as a rule for 'inheritance' of lexical category from a string to the segments of the current text that are instances of this string.

(202) lexcat(S,C) :- bstring(S,W), lexicon(W,C).

Suppose that we have a lexicon encoded as a predicate *lexicon/2*, possibly assigning to each word form more than one lexical category. Some of the clauses of this predicate is shown in (203).

(203) lexicon("to",inf).
lexicon("to",prep),
lexicon("the",det),
lexicon("walks",v).

Given this, and given the statement in (204), we can derive either (or both) of the statements in (205).

(204) bstring(34-35,"to").

(205) lexcat(34-35,inf).
lexcat(34-35,prep).

Needless to say, because of the remaining ambiguity, this is not satisfactory. And clearly, the problem with the statement in (202) is that it does not look at the context. (As we shall see in Chapter 14, another way to put it is to say that the statement in (202) is *false*!).

To handle disambiguation, we need to look at the context. Now, it is in fact possible to look at only the *local co-text* of the segments to be disambiguated, and to consider only shallow surface properties of this co-text. This observation is the basis for many techniques for automatic tagging, and in particular for the ones that will be discussed here.

11.2 The Elements of Automatic Tagging

Automatic tagging can be described as the process of mapping an untagged corpus into a tagged version of that corpus. In order to do this, a tagger needs to draw on some kind of knowledge base. The process is depicted in Figure 46.

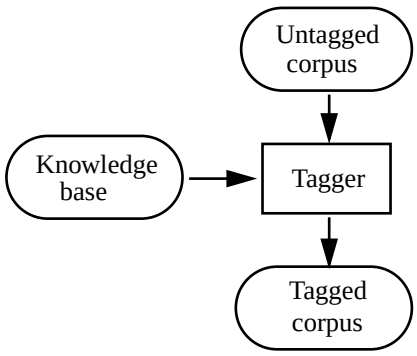


FIGURE 46. Automatic tagging

Thus, automatic tagging is a matter of representation, acquisition, and application of knowledge. That is, at least the following are needed:

- some strategy for representing the knowledge needed for automatic tagging, typically in the form of a lexicon, and a system of rules, constraints, or transitional probabilities;
- some method for acquiring the knowledge, either by manual work, or by learning from a corpus;
- some method of applying the knowledge to untagged corpora.

11.2.1 Requirements

With respect to the above diagram, the following requirements of methods for automatic tagging might be formulated. It is required that:

- the output of the tagger (i.e. the tagged corpus) is *complete* (in the sense that each word token has received a tag), *correct* (in the sense that every tag is correct), and *non-ambiguous* (in the sense that no word token has received more than one tag);
- the tagger does not use up too much computational resources, i.e. that it is reasonably efficient;

- it is reasonably non-problematic to acquire the knowledge base that the tagger needs.

From a TagLog point of view, we will also require of a method for automatic tagging that it is reasonably in line with the rest of the TagLog philosophy, and that it uses, as far as possible, TagLog logic in order to represent the knowledge needed for tagging.

In the following, I will consider four different approaches to automatic part-of-speech tagging: a statistics-based approach, a transformation-based approach, a constraint-based approach, and a novel logic-based approach.

11.3 Statistical Part-of-Speech Tagging

Typically, a statistics-based method for automatic part-of-speech tagging represents the required knowledge as a probabilistic model. Most popular among such models is the *hidden Markov model* (HMM). In a HMM there are *states*, *transitions* between states, and *symbols* emitted by the states. There are two kinds of probabilities associated with a HMM: *transition probabilities*; i.e. the probability of a transition from one state to another, and *output probabilities*, i.e. the probability of a certain state emitting a certain symbol. The model is called *hidden* since from watching the string of symbols that it outputs, we cannot in general determine which states it passes through.

I will illustrate here with a simple HMM in which the states represent parts-of-speech, and the symbols emitted by the states are words. The assumption is that a word depends probabilistically on just its own part-of-speech (i.e. its tag) which in turn depends on the part-of-speech of the preceding word (or on the *start* state in case there is no preceding word).[2]

Our sample HMM can be displayed as in Figure 47.

2. This is the assumption of a so called *biclass* model. It is more common in fact to use a *triclass* model, i.e. to assume that the part-of-speech of a word depends on the parts-of-speech of the preceding *two* words.

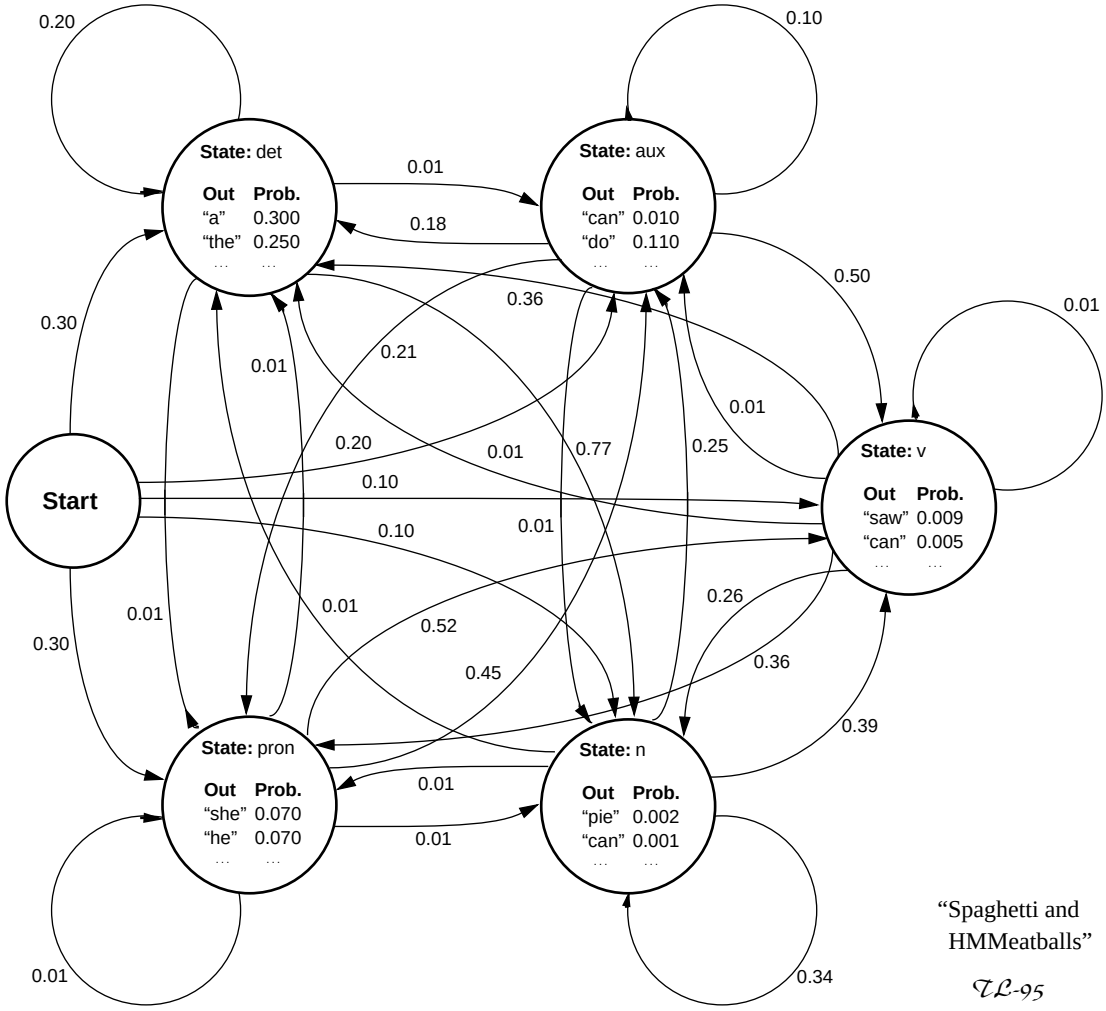


FIGURE 47. A Hidden Markov Model

Needless to say, in a realistic application, the HMM will have to be very much bigger. Still, this particular HMM is capable of generating, among other strings, the sentence "he can can a can". It passes from the *start* state to the state *pron*, from there to *aux*, then to *v*, and from *v* to *det* and finally to *n*, and outputs the words as it moves from state to state. The probability of this happening is easy to compute: we simply multiply all the transition probabilities and output probabilities along the path. This particular HMM could generate the same string of words by going through other states (e.g. start-pron-aux-aux-det-aux) but that would be less probable.

The above, however, is a good description of string *generation*, but not of part-of-speech *tagging*. In part-of-speech tagging, the string of symbols is given and the task is to find the sequence of state transitions most likely to have generated this string.

So to tag a given text, we need a program that computes the sequence of state transitions that has the highest probability of being the one that generated the text. In principle, the simple Prolog program in (206) would do just fine.[3]

(206)
```
most_probable_sequence(Words,P-Ss) :-
        findall(PS,sequence(Words,1-[start],PS),PSs),
        max_key(PSs,P-Ss1),
        reverse(Ss1,[start|Ss]).

sequence([],PSs,PSs).
sequence([Word|Words],P1-[S1|Ss],PSs) :-
        outprob(Word,S2,Po),
        transprob(S1,S2,Pt),
        P2 is Po*Pt*P1,
        sequence(Words,P2-[S2,S1|Ss],PSs).
```

It is assumed that the output probabilities are represented as Prolog clauses of the following kind (only two are shown here, the others are listed in Appendix B):

(207)
```
outprob(a,det,0.300).
outprob(can,aux,0.010).
 . . .
```

Moreover, it is assumed that the transitional probabilities are represented as follows (again not all of them are shown):

(208)
```
transprob(det,det,0.20).
transprob(det,aux,0.01).
transprob(det,v,0.05).
 . . .
```

Then the most probable sequence of tags corresponding to “he can can a can” is computed as follows:

(209)
```
?- most_probable_sequence([he,can,can,a,can],P-Seq).
        P = 1.965e-11
        Seq = [pron,aux,v,det,n]
```

This program is horrendously inefficient, however, since it examines *every* sequence of states that could possibly generate the text, which

---

3. See appendix B for some of the ‘implementation tricks’ used here.

indeed can be very many. In our HMM example there are 'only' 27 different sequences of states that generate "he can can a can" but that sentence is very short and with longer sentences and bigger HMMs we are looking at thousands or even hundreds of thousands of possible sequences. Hence, a naive algorithm will not do. Fortunately, there are algorithms that will find the most probable sequences very fast, without having to calculate the probabilities for all sequences. In Appendix B, I have given a Prolog implementation of one such algorithm – the *Viterbi algorithm* (Viterbi, 1967).

In our sample HMM, the probabilities have been invented. There are two well-known methods for 'training' the model, i.e. for acquiring the required probabilities automatically. The first method involves collecting statistics from a hand coded training corpus. For example, for each state *t* and each word form *w*, the output probabilities can be estimated by taking the number of occurrences of *w* tagged as *t* in a training corpus and divide it by the number of occurrences of *t*. And for each pair of states t_1 and t_2 the transitional probabilities can be estimated by taking the number of occurrences of t_1 being followed by t_2, and divide it by the number of occurrences of t_1. As I have shown in Section 9.4, such data would be simple to collect in TagLog. The second method for training a HMM is to estimate these probabilities without a hand coded corpus by means of the *forward-backward* algorithm (see e.g. Charniak, 1993).

As for the evaluation of methods for statistics-based tagging, researchers are reporting success rates of 90-95% (Church, 1988; DeRose, 1988; Charniak, 1993). For such a simple approach, this is surprisingly good. Tagging by means of this method is rather efficient as well, provided a good algorithm is used. Learning can be very fast, at least with the first method described above, since it merely involves counting. On the other hand, to work well, both learning methods require a lot of training material.

It is sometimes claimed that it is hard to read and grasp the significance of knowledge of the kind used by this approach. I disagree. If there are problems with this method, they are not of a semantic nature. The information represented in these tables is order-independent, meaningful, although perhaps a bit overwhelming when there is a lot of it (but that should be possible to take care of by means of browsers that can hide information

considered insignificant). Moreover, the method is both simple and elegant.

However, although it is clear that TagLog is *compatible* with a statistics-based approach to automatic tagging, and that it would not be out of the question to develop a tagger module in TagLog based on such principles, we will, in the sections to follow, continue to search for something more in the spirit of TagLog. The fact that we have implemented Viterbi tagging in a logic programming language does not make Viterbi tagging part of the logical approach to computational corpus linguistics.

11.4 Transformation-Based Part-of-Speech Tagging

Part-of-speech tagging had for some years been an area of natural language processing where statistical techniques were more successful than rule-based methods. Then Brill (1992) presented a simple rule-based part-of-speech tagger which performs with an accuracy comparable to statistics-based taggers.[4] In this section, I will demonstrate that a system of TagLog condition-action rules (see Section 3.3.3) is a powerful enough device to implement Brill's method.

Brill's method, too, involves a lexicon. Suppose we have a lexicon, accessed via a predicate *lexicon/2*, that assigns *only the most likely tag* to a word form. For example, this lexicon would list "run" as a verb, but not as a noun, despite the fact that "run" sometimes occur as in "The run lasted thirty minutes". One obvious way to acquire such a lexicon would be to extract it from a large hand coded corpus, perhaps using TagLog for that purpose.

Brill's strategy also involves rules that will assign tags to segments, based on what is in the lexicon, the endings and capitalization of words, and co-text. We will use TagLog condition-action rules for this purpose.

4. By way of comparing transformation-based on the one hand with statistical approaches like the HMM method explained in Section 11.3 on the other, one should note that, in a sense, the former approaches rely on contextual/transitional probabilities too, but only in an indirect way: the statistics is calculated during learning time, and is not part of the resulting description.

For example, the rule in (210) has the effect that all segments that are instances of strings that are in the lexicon receive the most likely tags. Instances of word forms that are not in the lexicon do not receive any tag at all.

(210) bstring(S,W), lexicon(W,C) ==> add(lexcat(S,C)).

With the rule in (211), we take advantage of the fact that words not in the lexicon tend to be proper nouns if they are capitalized.

(211) bstring(S,W), not lexcat(S,_), capitalized(W) ==> add(lexcat(S,pn)).

The rule in (212) has the effect that words that are ending in "ous" and that have not already received a tag for *lexcat*, are tagged as adjectives, because that is the most common tag for words ending in that way.

(212) bstring(S,_+"ous"), not lexcat(S,_) ==> add(lexcat(S,adj)).

It is easy to think of other rules along similar lines.

The meaning of the rule in (213) is that if a segment is tagged *inf* (meaning *infinitive marker*) and the following segment is tagged *det* (meaning *determiner),* then switch the tag from *inf* to *prep* (meaning *preposition*).

(213) lexcat(P0-P1,inf), lexcat(P1-P2,det) ==>
replace(lexcat(P0-P1,inf), lexcat(P0-P1,prep)).

The rationale for this rule is that a noun phrase is much more likely to follow immediately after a preposition than after the infinitive marker *to*.

The rule in (214) replaces all *noun* tags immediately preceded by *inf* tags with *verb* tags.

(214) lexcat(P0-P1,inf), lexcat(P1-P2,n) ==>
replace(lexcat(P1-P2,n), lexcat(P1-P2,v)).

A sequence of condition-actions rule are executed, in order, by calling *ca_run/0* (defined in Section 3.3.3).

Thus, the first three rules just add tags so that, if all goes well, every basic segment has received exactly one tag when these three rules have executed. The fourth and fifth rules 'patches' the result of these executions, by replacing certain tags with other tags, if certain co-textual conditions hold. These rules are called "transformation-rules". Note how important the ordering of the rules are, since each rule may radically change the conditions under which the rules that follow work.

11.5 The Learning of Transformation-Rules

One of the main points in Brill's work is that transformation-rules for automatic tagging can be automatically acquired. Brill calls his technique "error driven learning". The distribution of errors produced by an imperfect tagger (i.e. a tagger making use of all available rules except transformation rules) is examined to learn an ordered list of transformation-rules that can be applied to provide more accurate tagging. The method is depicted in Figure 48.

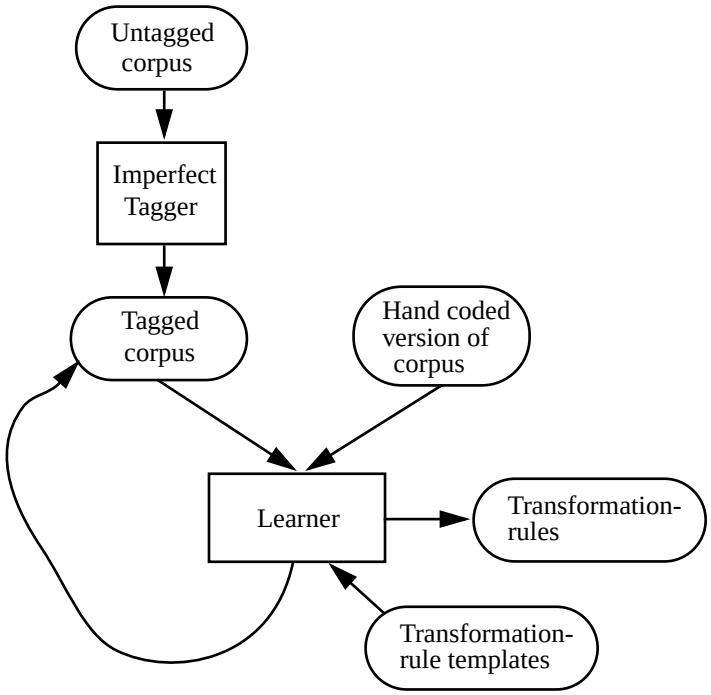


FIGURE 48. Error-driven learning (Adapted from Brill, 1992)

At each stage of learning, the learner finds the transformation whose application to the tagged corpus results in the best scoring tagged text. The transformations considered are instances of templates, defining a set of allowable transformations. The scoring is calculated by comparing the result of transformations with a small hand coded text which is assumed to be tagged correctly. Learning proceeds on the tagged text that results from applying the learned transformation. This continues until no more transformations can be found whose application results in an improvement. Once an ordered list of transformation-rules has been learned,

unseen text is tagged by simply applying the imperfect tagger, followed by each transformation, in order, to the entire text.[5]

This learning method can be implemented as in (215). The procedure *learn/1* generates all possible instantiations of all templates, and then calls *learn/2* which performs an exhaustive search through this space, finds the transformation-rule that maximizes the increase in truthlikeness of the current theory, adds it to the other rules, 'runs' it, and then calls itself recursively in order to proceed with learning giving the new current theory. The procedure terminates when the best rule does not increase performance above a threshold set by the user.

(215)
```
learn(Threshold) :-
        setof(Rule,template(Rule),RuleInstances),
        learn(Threshold,RuleInstances).

learn(Threshold,RuleInstances) :-
        find_best(RuleInstances,WinnerRule,WinnerScore),
        assert(WinnerRule),
        execute(WinnerRule),
        WinnerScore >= Threshold,
        learn(Threshold,RuleInstances).
learn(_Threshold,_RuleInstances).
```

Templates are specified by the user of the system, in the form exemplified in (216). Templates typically involve segments within a few positions on each side of the segment whose description will be affected by the transformations.

(216)
```
template((lexcat(P0-P1,A), lexcat(P1-P2,B) ==>
                replace(lexcat(P0-P1,A), lexcat(P0-P1,C)))) :-
        tag_value(lexcat,A),tag_value(lexcat,B),tag_value(lexcat,C).

template((lexcat(P0-P1,B), lexcat(P1-P2,A) ==>
                replace(lexcat(P1-P2,A), lexcat(P1-P2,C)))):-
        tag_value(lexcat,A),tag_value(lexcat,B),tag_value(lexcat,C).
```

Given a list of rule instances, the procedure in (217) evaluates all of them and finds the one that scores best.

(217)
```
find_best([NewRule|Rules],Winner,Score) :-
        evaluate(NewRule,NewScore),
        find_best(Rules,NewRule,NewScore,Winner,Score).
```

---

5. It might be interesting to regard this method as an instance of Popper's schema:

$$P_1 \Rightarrow TT \Rightarrow EE \Rightarrow P_2.$$

```
find_best([],Winner,Score,Winner,Score).
find_best([NewRule|Rules],Leader,HighScore,Winner,Score) :-
        evaluate(NewRule,NewScore),
        (       NewScore > HighScore ->
                find_best(Rules,NewRule,NewScore,Winner,Score)
        ;       find_best(Rules,Leader,HighScore,Winner,Score)
        ).
```

The predicate *evaluate/2* evaluates a rule with respect to the current text. The number of cases where the current tagging is incorrect and the transformation-rule will correct it, and the number of cases where the current tagging is correct, and therefore the transformation will cause some damage, are calculated. The score of the rule is returned as the difference between the former number and the latter.[6]

```
(218)   evaluate((Condition ==> replace(A, B)), Score) :-
                count((*B, Condition), Positive),
                count((*A, Condition), Negative),
                Score is Positive - Negative.
```

Now, unfortunately, this implementation is far too inefficient to be practically useful. The problem is that we are trying to generate all instances of all templates before doing anything else, and usually the number of instances is very large (for example, given 6 templates and 20 different tags, 48.000 rules will be generated! To find the one that scores best, the procedure would have to evaluate each of them with respect to the whole training corpus!). Instead, as pointed out by Brill, a better strategy is to begin by computing a list of all error types that the current sequence of condition-action rules makes, and then generate and test only those rules that may be used to correct some of these errors. Thus, only a very small percentage of possible transformation-rules really need to be examined. I have implemented a procedure that has been improved in this way (and in other ways as well). It is given in Appendix B. When tested with 6 templates and 20 different tags on a >2000 word corpus, it was able to find 20 rules within a couple of hours on a Sparcstation 10 running Sicstus Prolog. This is not fast, but it is fast enough to be practical.

With 71 patch rules along the lines of (213) and (214), Brill obtained an error rate of 5.1%, when testing the tagger on 5% of the Brown Corpus.

---

6. Note that the clauses in the supposedly correctly hand coded theory are assumed to be clauses of the form **Clause*, where '*' is a prefix functor. This is just to distinguish them from clauses produced by the automatic tagging procedure.

This is on par with the best stochastic taggers. Also, tagging is efficient. Learning, though, is rather slow. On the other hand, learning does not require a very large pre-tagged corpus to learn from, a rather small one will suffice.

It is argued by Brill that rule-based taggers have important advantages over stochastic taggers: they are smaller, more perspicuous, and more portable. I believe, however, that this claim should be balanced against the fact that condition-actions rules can be very hard to understand since what they actually do is dependent on what rules that were executed earlier have already done. Nevertheless, it is probably true that Brill's transformations are more open for manual tuning, and thus more suitable for a balanced cooperative approach, than are the matrices of statistical data.

How well does Brill's approach fit in with TagLog? What about connectivity? Since the conditions of condition-action rules are *logical* conditions, a Brill tagger implemented as above will (in principle) be sensitive to knowledge produced by other TagLog tools. Furthermore, since what is asserted (or retracted, or replaced) by the action parts of condition-action rules are logical sentences, other tools will be aware of the result of using a Brill tagger. But the condition-action rules *as such* will be of no use to other tools. Moreover, TagLog tools such as the Explanation Tool or the Hypothesis Testing Tool – tools that do very useful things with universally quantified conditionals – do not understand condition-action rules, so special purpose tools for debugging and analysis of condition-action rules would probably have to be implemented as well.

11.6 Logic-Based Part-of-Speech Tagging

Brill's strategy is promising: it is simple and powerful, and it can be implemented efficiently in TagLog by means of condition-action rules.[7] But it is anything but declarative, and, as always, a declarative account would be preferable. So in this section, I sketch a rather more declarative and 'logical' solution to the problem of part-of-speech tagging.

7. For an extremely efficient implementation by means of finite-state transducers, see (Roche & Schabes, 1995).

In Section 11.1, we saw that a simple inheritance rule cannot adequately describe the relation between a lexicon and tokens in a text. Fortunately, as we have already seen, there seems to exist rather distinct contextual constraints on when, for example, a segment can be an instance of an infinitive marker. An infinitive marker can never immediately precede a determiner – whatever immediately precedes a determiner is not an infinitive marker.

Nothing can be more natural than to capture such constraints by means of negative statements, such as the statement in (219).

(219) ¬ lexcat(P0-P1,inf) :- lexcat(P1-P2,det).

In order to make our 'lexical inheritance' rule sensitive to constraints, we need to modify it as follows:

(220) lexcat(S,C) :-
bstring(S,W), lexicon(W,C),
consistent(lexcat(S,C)).

Thus, what we are looking at here is a kind of default inheritance rule, that will allow inheritance as long as it does not give rise to an inconsistency. Typically, inconsistencies arise when constraints apply, and thus, inheritance is not allowed in these cases. Ideally, our system of constraints will allow only one part-of-speech to be assigned to each segment – the correct one – although in practice we will probably have to be content with a substantial overall reduction in ambiguity.

Now, given the statements in (221), only (222) can be derived, not (223).

(221) bstring(34-35,"to").
bstring(35-36,"the").

(222) lexcat(34-35,prep).

(223) lexcat(34-35,inf).

Which is exactly what we wanted.

We can see this as a kind of 'ambiguity resolution through consistency recovery'. *Before* we add the constraint in (219) to the current theory, it follows, from the sentence in (220), and from *bstring(34-35,"to")*, that both *lexcat(34-35,prep)* and *lexcat(34-35,inf)* hold. Given a 'single-valued' constraint, these theorems are inconsistent with one another. But each of them holds by default only. So when we add the constraint in (219), one of them, namely *lexcat(34-35,inf)*, is explicitly negated and

thus overridden, so that now only the other, namely *lexcat(34-35,prep)*, still holds. Hence, by the addition of the constraint, the full description of *34-35* was disambiguated in favour of *lexcat(34-35,prep)*.

This is a non-monotonic process: by adding an axiom, we invalidated a previous theorem. Since that theorem happened to contradict another theorem, consistency was recovered, and thus the ambiguity was resolved.

Let us look at a couple of other constraints. An infinitive marker can never precede a proper noun either, so let us reformulate the relevant constraint by means of a disjunction. Moreover, verbs do not succeed determiners. Formally:

(224) ¬ lexcat(P0-P1,inf) :- lexcat(P1-P2,det) ; lexcat(P1-P2,pn).
¬ lexcat(P1-P2,v) :- lexcat(P0-P1,det).

Several hundred such constraints would probably be very useful. The problem is to find them. Here too, of course, the ideal would be a procedure that allows us to acquire them automatically from a hand coded corpus. Perhaps the distribution of errors produced by the 'lexical inheritance rule' could be examined to learn a set of constraints in the form of negative statements that could be added to the current theory to provide more accurate tagging. I have placed this on the agenda for future work.

All things considered, logic-based tagging seems to be a both viable and elegant approach, well worth exploring further. What remains is to assess its practical merits and disadvantages, and to compare it with other approaches when tried on real corpora. This is outside the scope of the present thesis, however. In the mean time, before such an investigation has been performed, we can only assess and compare it from a theoretical point of view. The logical approach to automatic tagging then compares very favourable with Brill's approach.

The logic-based approach to automatic tagging could be characterized as non-modular, implicit, and demand-driven, whereas we might want to characterize Brill's approach (and many other approaches) as modular, explicit, all-at-once, and once-and-for-all.

I think that the non-modular approach of logic-based tagging is more elegant than Brill's two-stage approach. The strategy of adding a tag (say *inf*), only to swap it for another tag (say *prep*) the moment after, is neither elegant nor efficient (why add it in the first place?).

Furthermore, with a large on-line lexicon in the background there should be no need to tag segments explicitly for parts-of-speech, all at once and once and for all. In the logic-based approach, information about parts-of-speech follows implicitly from the lexicon, the 'lexical inheritance' rule, and the negative statements encoding the constraints, and can be derived automatically when there is a *demand* for it. Of course, we could still be interested in 'caching' these statements as lemmas, by deriving them and then storing them explicitly, once and for all, but in the logic-based approach it no longer constitutes a necessary step in a procedure for automatic tagging.

The logic-based approach to automatic tagging is formulated wholly within the TagLog logical framework. It is therefore easy to integrate – or rather, it *is* already integrated – with other TagLog functionalities, such as searching, counting, concordancing, hypothesis testing, and explanation. For example, as will be demonstrated in Chapter 13, the TagLog Explanation Tool can be used for debugging of theories used for tagging, as it stands. As I have mentioned already, since transformation-based tagging can only be implemented outside the logical framework, with condition-action rules, there is a need for special purpose mechanisms for maintenance and debugging of such rules.

It is this last point in particular that shows that it is the logic-based path we want to tread. So even if future practical tests show logic-based tagging to be inefficient and therefore impractical, a standard would have been set, that we ought to try to live up to.

In the mean time, I believe that a transformation-based tagger implemented by condition-action rules is a very useful tool to have in TagLog. In contrast to logic-based tagging, Brill's approach has already been proved to work very well in practice with big corpora and non-trivial tagging tasks.

11.7 Constraint-Based Part-of-Speech Tagging

The logic-based tagging method shows some similarities with the *Constraint Grammar* (CG) approach (Karlsson, 1990; Karlsson et al., 1994).

In CG, a *constraint* is a quadruple. Karlsson (1990) gives the example in (225).

(225) (@w =0 PREP (-1 DET))

This constraint has the 'semantics' that "if a word (@w) has a reading [out of a set of several possible 'readings'] with the feature PREP, this very reading is discarded (=0) iff the preceding word (i.e. the word in position -1) has a reading with the feature DET" (Karlsson ibid., p. 170). Although the constraints in CG are claimed to be declarative, and thus to embody true statements, reading them in the above manner gives them a procedural ring.[8]

In TagLog, a constraint with the same effect as the one in (225) can be given as follows instead:

(226) ¬ syncat(P1-P2,prep) :- syncat(P0-P1,det).

This, indeed, is a declarative constraint. To see what it would mean to use TagLog's condition-action rules for part-of-speech tagging in the CG manner, consider the following rule:

(227) syncat(P0-P1,det) ==> delete(syncat(P1-P2,prep)).

The use of condition-action rules imposes a certain processing strategy on the CG method, however, and an order-dependence between rules that CG people would perhaps claim that CG ought not to have, but it might still be a method that a TagLog user would find useful, in certain circumstances.

Karlsson presents a number of mechanisms that are supposed to make it easier for the rule writer to construct effective rules (indeed, as we will see below, Karlsson's approach presupposes a rule writer). In my view, these mechanisms look unprincipled and *ad hoc*. If they are reconstructed

---

8. Another problem is that it is not sufficiently clear what is meant by 'a set of readings' and how they relate to conditions. Since the CG framework is supposed to be declarative, we should be able to interpret them as statements (the TagLog way). Now if there is more than one reading for a word (which is often the case), we could either interpret them disjunctively, or we could interpret them conjunctively. Suppose we interpret them disjunctively. It would not then be clear what is meant by a condition being satisfied by the local context. A disjunction of statements cannot of course, by themselves, satisfy an atomic condition. Suppose we interpret them conjunctively. This would be the TagLog approach, which works well, but is this the interpretation that Karlsson intends?

in logic we might be able to see how they can be improved, or at least more clearly formulated. For example, the *C* in the constraint in (228) is supposed to mean that the assignment of *TO* to the preceding word is unique:

(228) (@w =0 VFIN (-1C TO))

Needless to say, this is not hard to say in logic either:

```
(229)  ¬ syncat(P1-P2,vfin) :-
            syncat(P0-P1,inf),
            not (syncat(P0-P1,V),
                 V \== inf).
```

In CG "the operator '=!' indicates that the target reading is the correct one iff all context conditions are satisfied, all other readings are discarded" (Karlsson, 1990, p. 170). It seems to me, though, that a constraint such as

(230) (@w =! VINF (-1 TO))

would be better expressed as

```
(231)  ¬ syncat(P1-P2,C) :-
            syncat(P0-P1,to),
            C \= vinf.
```

so that the uniform *negative* format of the constraints is kept.

In CG, a *template mechanism* is available for expressing partial generalizations. Karlsson's example is that a template *&NP* could be declared to contain the alternatives *((N))*, *((A)(N))*, *((DET)(N))*, *((DET) (A) (N))*, etc. This is clumsy and *ad hoc*, and it does not allow recursion. In TagLog, the user can write grammar rules, and define some of the conditions of his constraints in terms of what follows deductively from such grammars. So instead of the template above, we can have the recursive X-bar style grammar in (233).

(232) ¬ syncat(P0-P1,det) :- syncat(P1-P2,np).

```
(233)  np --> det, n1.
       np --> n1.
       n1 --> n.
       n1 --> adj, n1.
```

This is more general and what is more it is pure logic.

Voutilainen et al. (1992) tested the CG system on running English text, with a rule base consisting of 1,100 constraints. They report that while roughly one word out of two is morphologically ambiguous, after disam-

biguation 94-97% were fully unambiguous. Of all word forms in the output, 99.7% have the most appropriate reading left. This is impressive. The system is efficient too, according to tests performed by the developers.

I think I have shown above, however, that certain logical unclarities about the whole approach remain. The Constraint Grammar approach is in need of a logical reconstruction, and perhaps my treatment in this chapter could be regarded as a first attempt at doing this.

Creating a system of constraints in CG is hard manual work since no learning mechanism is available. However, I believe that a comparable result can be achieved with a system of constraints automatically acquired from corpora. Using surface-directed, local constraints but without using automatic learning seems to me to be a less than optimal way of doing things. In this respect, CG as well as the logic-based approach is unbalanced.

11.8 Summary and Conclusion

The TagLog framework is compatible with many methods of automatic tagging. I have concentrated on two techniques that seem to be more suitable than others: Brill's transformation-based technique, and a novel logic-based technique.

Although it cannot be said to be logic-based in itself, Brill's transformation-based technique works very well in a logic-based environment. Implementation is very simple, and the result is a more flexible, high-level system than Brill's own.

This chapter also took a small step towards a logic-based approach to tagging, and showed that this approach is well worth exploring further.

CHAPTER 12

Hypothesis Testing

All grammars leak.
Edward Sapir

12.1 Introduction

How good is my hypothesis? How good is my theory? Should I keep it? Should I throw it away? The testing of hypotheses and theories is a very important aspect of the work of the scientist, and no less so if he happens to be a linguist. According to Popper, it is testing, the possibility of falsification, that distinguishes science from non-science. This chapter is about a tool in TagLog – the TagLog Hypothesis Testing Tool – that to some extent automates the testing of hypotheses and theories, gives a comprehensive overview of the results of the testing, and supports the revision of hypotheses and theories in the light of these results.

12.2 The Logic of Hypothesis Testing

I will assume that four different kinds of instances of a universally quantified conditional are relevant to its status as a hypothesis. Given a statement *S* of the form *A :- B*, and given that *A'* and *B'* are ground instances of *A* and *B*, respectively, *A' :- B'* is a *confirming instance* of *S* iff both *A'* and *B'* are true, a *falsifying instance* of *S* iff *B'* is true but *A'* is false, a *predictive instance* of *S* iff *B'* is true and *A'* is unknown, and a *non-covered instance* of *S* iff *A'* is true but *B'* is not.

If we assume the truth of what is provable, and if we assume that what has been described as 'observed' is also true, then, in terms of the meta-predicates defined in Chapter 10, these notions might be defined as follows:

(234)
```
confirming_instance((A :- B)) :- provable(B), observed(A).
falsifying_instance((A :- B)) :- provable(B), observed(¬ A).
predictive_instance((A :- B)) :- provable(B), unobserved(A).
noncovered_instance((A :- B)) :- observed(A), unprovable(B).
```

Now, as their form may suggest, the definitions in (234) will be used in the TagLog system: for searching and concordancing, of course, but mainly for *counting*.

12.3 Measuring the Status of a Hypothesis

Based on the distribution of confirming, falsifying, predictive, and non-covered instances of a universal statement, I have made an attempt to 'operationalize' the notions of *coverage* and *truthlikeness*. (As we will see, I will also define, and operationalize, a notion of *conclusiveness*.) The way I will do this is certainly subject to change, but it should be clear that what I want to find is a set of measurable notions that the system can use in order to communicate the status of a hypothesis to the user so as to bring about the further development of the current theory, and possibly to give that development a push in the right direction. Such notions must be intuitive, be possible to operationalize, and preferably be easy to visualize. I have designed them in such a way that each measure can be

expressed as a real number between 0 and 1. Thus, as we will see, they can be vizualized as circle diagrams.

These tests certainly do not tell us all there is to know about a universal statement. A statement may score 100% in truthlikeness, 100% in coverage, and 100% in conclusiveness, but still be trivial and uninteresting. Unfortunately, I can think of no way to measure interestingness that would fit here.

12.3.1 Coverage

Intuitively, the *coverage* of a universal statement is the extent to which it predicts or explains the occurrences of the particular phenomenon that it purports to predict or explain.

A measure of coverage of a universal conditional is simply defined by taking the number of its *covered* instances, i.e. the number of confirming instances *plus* the number of falsifying instances *plus* the number of predictive instances, and divide it by the *total* number of all its instances, i.e. by the number of covered instances *plus* the number of non-covered instances.

12.3.2 Truthlikeness

In Chapter 2, some intuitions behind the notion of *nearness to truth*, or truthlikeness, were briefly discussed. In an attempt to make this more precise, we say that the truthlikeness of a universally quantified conditional is obtained by dividing the number of its confirming instances with the number of its confirming instances *plus* the number of its falsifying instances.

12.3.3 Conclusiveness

Note that our definition of truthlikeness did not say anything about predictive instances. This is because as long as a universal conditional has predictive instances, we do not really *know* its truthlikeness, since we do not know whether the predictive instances would turn out to be confirming or falsifying, if we were to investigate them more closely.

Thus it seems that we need some measure of confidence in the measure of truthlikeness that we have obtained in a particular case. I have designed such a measure, and I have called it *conclusiveness*.

This measure is defined as the ratio between the number of confirming instances *plus* the number of falsifying instances, and the number of covered instances (i.e. the number of confirming instances *plus* the number of falsifying instances *plus* the number of predictive instances).

12.4 The TagLog Hypothesis Testing Tool

The TagLog Hypothesis Testing Tool is designed to provide the researcher with a measure of the degree of success with which a general hypothesis describes the relevant facts of a text, based on the proportions between the numbers of confirming, falsifying, predictive and non-covered instances of the universal statement expressing the hypothesis. The proportions are presented in two ways: in the form of a table, and in three circle diagrams. The tool also allows the researcher to search for the segments involved in these different kinds of instances. The dynamic link to the Statement Browser makes it easy to inspect the descriptions of the segments, and modify them if necessary.

Selecting the Hypothesis Testing Tool from the Tool menu brings up the dialog box in Figure 49, containing one scrollable text entry field, one table, four buttons and one diagram drawing area.

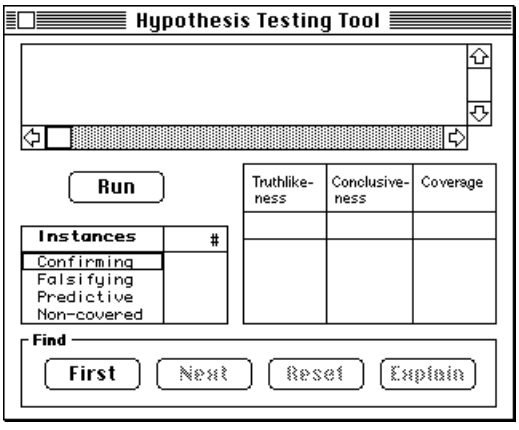


FIGURE 49. The TagLog Hypothesis Testing Tool

The universal statement for which the user wishes to run statistics is entered in the text entry field.

Run: A click on this button runs the statistics for the universal statement in the text entry field, prints the resulting figures in the table, and draws the corresponding circle diagrams in the diagram area.

First: Clicking this button finds the first occurrence of the instance type highlighted in the table (i.e. confirming, falsifying, predictive, or non-covered) and selects the occurrence in the Text window. Besides, clicking this button enables two other buttons:

Next: Finds the next occurrence of the instance type highlighted in the table. Subsequent clicks on this button will find the rest of the instances, one at a time.

Reset: Returns to top level.

Explain: Activates the link to the TagLog Explanation Tool (see Section 13.7). An attempt to explain the consequent of the current hypothesis as it applies to the current segment is then made.

12.5 Interpreting the Results of Hypothesis Testing

The result of a Hypothesis Testing run 'says' something about the universal statement in question, and it suggests actions to be taken in order to improve the current theory, actions such as checking, modifying or extending.

12.5.1 The significance of predictive instances

As defined above, the predictive instances are the instances that claim something about some of the segments of the corpus, but where the system does not know whether what they claim is true or not.

It is up to the researcher to *check* the predictions, i.e. it is the responsibility of the researcher to determine, on the basis of observation, whether a particular prediction is successful or not, i.e. whether the consequent of a particular predictive instance of a universal statement is true or not. If it is found to be true, the instance becomes a confirming instance, but if it is found to be false, it becomes a falsifying instance. (If the prediction is false, the researcher should probably consider the kind of action suggested for falsifying instances, explained below.)

Suppose that we have run the statistics for a particular universal statement and that we have selected 'predictive' in the tool's table. In order to check (and possibly confirm/disconfirm) the predictions made, perform the following steps:

1. Click First. The system will select the segment involved in the first predictive instance found by the system.
2. Determine, on the basis of observation, whether the consequent is true or false of this segment.
3. Regardless of whether the prediction is true or not, it might be a good idea at this stage to *explicitly* confirm or disconfirm it, by introducing, in the current theory, an explicit observation statement describing the selected segment. The consequence is that, if we run the statistics again, the instance will be counted as confirming or falsifying rather than predictive.
4. Click Next in order to check the rest of the predictive instances.

12.5.2 The significance of confirming instances

By definition, a confirming instance of a universal statement is an instance where both the antecedent and the consequent are true. But as Karl Popper (1959), among others, has told us, not even a zillion confirming instances of each of its statements can conclusively verify a theory. There is always a chance that we will find an instance that breaks the rule, and smashes our theory to pieces. Still, a statement that is sufficiently confirmed is usually accepted, at least tentatively, and our *confidence* in a hypothesis is likely to increase as the number of confirming instances increases. Moreover, as the number of confirming instances increases (other things being constant) the degree of *truthlikeness* increases as well.

In contrast to the other types of instances, confirming instances do not really prompt the researcher to any action at all. He may rest in the confidence created by them.

12.5.3 The significance of falsifying instances

Again by definition, the falsifying instances of a universal statement are the instances that are false, i.e. where the antecedent is true but the consequent false.

If a theory is falsified because one of its statements has a falsifying instance, the researcher does not have to throw *all* of it away. But the existence of a falsifying instance suggests that it would be wise to modify *something*. There are at least three possibilities:

1. Change the theory. Remove the universal statement which has a falsifying instance from the theory, and thus save the rest of it. Or find some way of specializing the universal statement in question, so that it no longer has falsifying instances.
2. Change the description of data. The description contradicting the conclusion of the universal statement can be a incorrect. Correct it, and thus make the falsifying instance disappear.
3. Of course, practically speaking, there is also a third possibility open to the researcher, namely to ignore the fact that the theory is falsified. He can even justify this manoeuvre by pointing out the fruitfulness of what he has, in spite of its falsity, or he can promise that ‘soon, very soon now, this apparent contradiction will be resolved, but until then, let’s hang on to this idea...’. As Popper (1992) puts it , “some degree of dogmatism is useful, even in science”.

12.5.4 The significance of non-covered instances

A confirming outcome of the confirming and predictive components of testing is not sufficient in itself, because it may be that the statements confirm or successfully predict only trivial cases and leave the not so trivial cases still to be explained. This is the problem of *insufficient coverage*. A maximally successful theory must generate *only* true statements of the relevant domain, but it must also generate *all* true statements of this domain, at least as long as they are relevant.

The Hypothesis Testing Tool supports a practical method for extending the coverage of a particular universal statement. The first step is to check the cases where it fails to cover.

1. Select ‘Non-covered’ in the table.
2. Click the Find button in order to find the first segment involved in a non-covered instance of the statement.
3. Extend the theory so as to cover the selected case.
4. Repeat until there are no non-covered instances left to be found.

12.5.5 Summing up significance

In practice, we will probably have to learn to live with theories that are known to contain some false statements and known to fail to contain some true statements. Most theories are falsified. Most theories have less than total coverage. The important thing for the researcher is to be *aware* of falsifying and non-covered instances if there are any. It is only if the researcher is aware of them that they can prompt further development of the theory. In this sense, TagLog can free the researcher from ignorance, but I do not think TagLog, or any such system, can ever free the researcher from the responsibility of deciding and acting correctly on the basis of what he knows. Science is not *only* a question of logic.

12.6 Testing a Simple Hypothesis

In order to demonstrate the idea of automatic hypothesis testing, we will work with a simple, almost trivial, example. The example is based on the PLUS corpus text, but I have deliberately introduced some errors and a certain amount of incompleteness in the description, in order to provide us with more interesting results.

Suppose the hypothesis that we want to test can be expressed in natural language as in (235), and thus be formalized as in (236).

(235) Every instance of the string “and” is a conjunction

(236) syncat(S,conj) :- bstring(S,“and”)

We enter the formula in (236) in the Hypothesis Testing Tool’s text entry field, and then click the Run button to run the statistics. The outcome is shown in Figure 50.

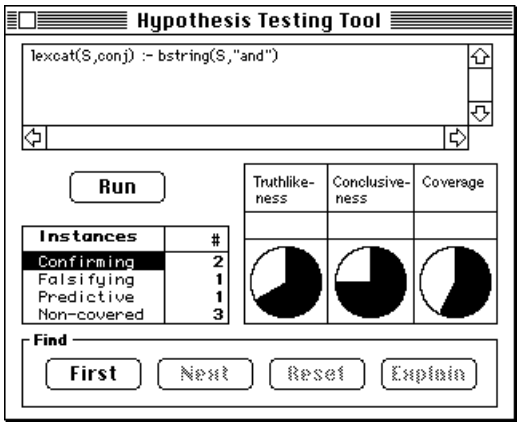


FIGURE 50. Testing a simple hypothesis

The table tells us that there are two segments in the current text that are instances of the string “and” *and* at the same time instances of the lexical category ‘conjunction’. Thus, these are two cases that confirm the hypothesis. But there is also one occurrence of “and” which is provably *not* a conjunction, and which therefore falsifies the hypothesis. Furthermore, there is one predictive instance of the hypothesis, i.e. one case where the hypothesis predicts of an occurrence of “and” that it is a conjunction. In this case, however, the hypothesis can neither be confirmed, nor falsified, on the basis of what the system knows. Upon closer inspection, it may turn out to be either confirming or falsifying. Finally, the high number of non-covered instances, or rather the ratio between the number of *covered* instances and the number of covered instances *plus* the number of non-covered instances, tells us that the coverage is low, which should come as no surprise since there are conjunctions other than “and”.

With only two confirming cases and one falsifying case, one predictive case, and three cases not covered at all, our hypothesis appear to be rather weak.[1] Either the hypothesis does not get the facts of the text right, or these facts have not been completely and truthfully described.

Aided by the table and the diagrams, we are in a position to make well-informed decisions of how to modify either the hypothesis, and hopefully

1. The figures are very small since we are working with only one short text, but it will do for the purpose of demonstration.

improve upon it, or the description of the facts, in the hope of showing that the hypothesis is not as bad as it first seemed.

We begin by looking at the confirming cases. After all, we ought to check that they have not been confirmed for the wrong reasons! We select ‘Confirming’ in the table and click the First button telling the system to select the first confirming instance. The result shows in the screen in Figure 51.

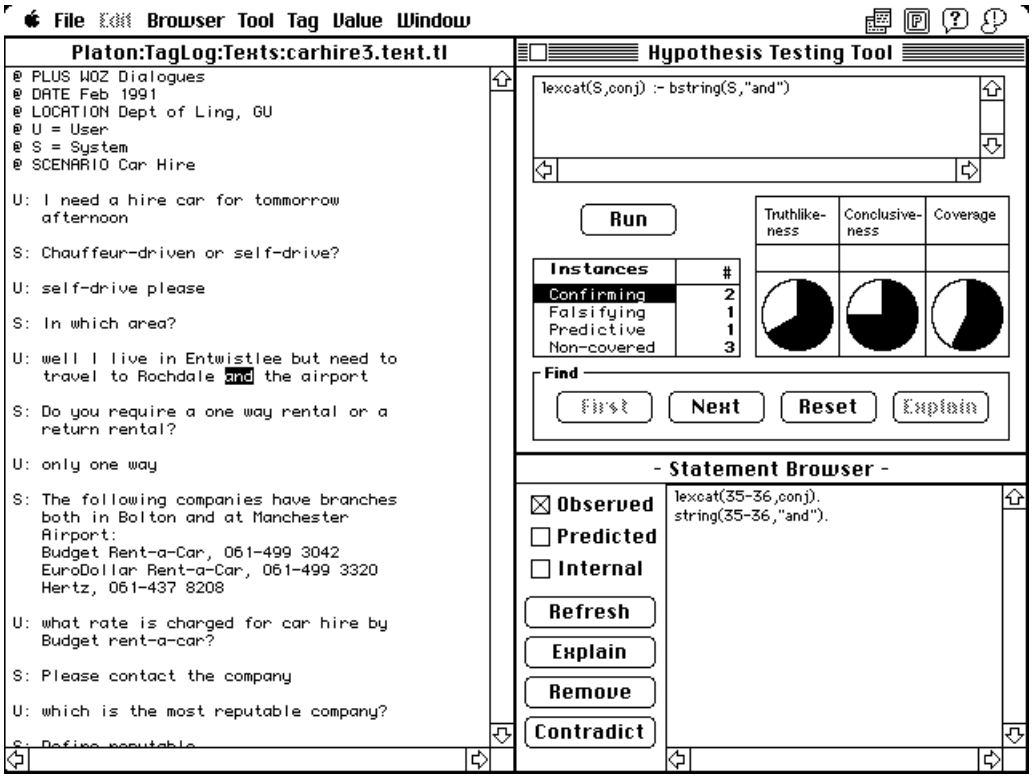


FIGURE 51. Testing a simple hypothesis (cont.)

By observing the selected segment as it appears in the text, and by looking at the Statement Browser, we are able to confirm that the instance in question is a confirming one.

Now, let us inspect the falsifying instances of our hypothesis, to see if we can modify the hypothesis, or perhaps the description of data, so as to make the falsifying instance disappear, or if we can leave it as it is, fully aware that it is false, and remain happy anyway. When we select ‘Falsifying’ in the table, and click the Find button, the screen displays what is shown in Figure 52.

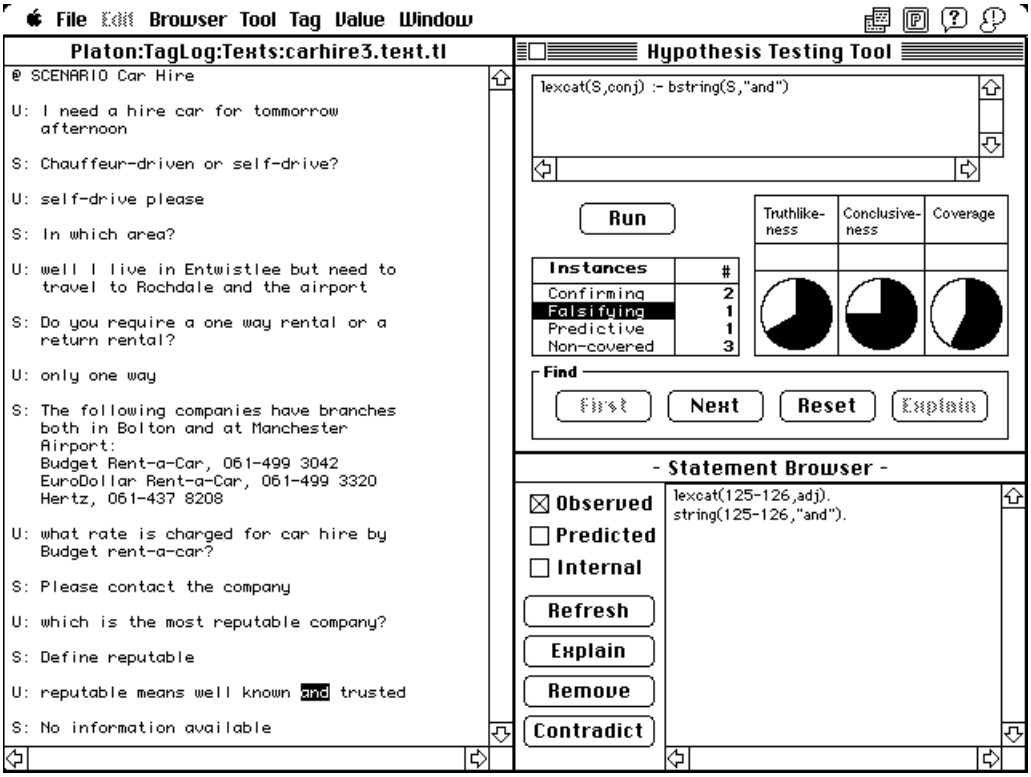


FIGURE 52. Testing a simple hypothesis (cont.)

Note how clearly the Statement Browser reveals the source of the falsifying instance: Due to a mistake during the description phase (a slip of the mouse during selection from a menu, say), the selected instance of “and” was wrongly described as an adjective! Thus, the action called for here is not a modification of the hypothesis, but a modification of the description of data. We delete the false statement (by selecting it in the Statement Browser and pressing the Backspace key) and we add a new statement saying that the selected segment is a conjunction (by selecting from the Value menu – the normal procedure).

If we re-run the statistics (this is not necessary at this stage, but let us do it anyway), the table in Figure 53 is displayed.

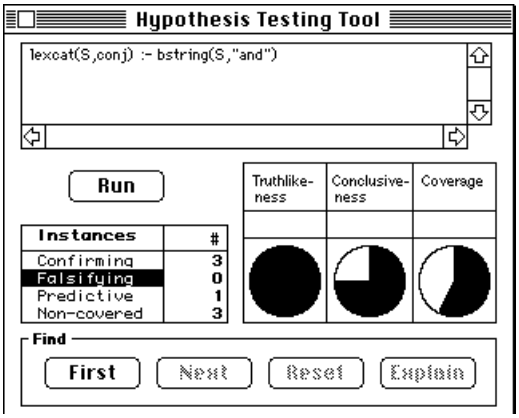


FIGURE 53. Testing a simple hypothesis (cont.)

In order to check the prediction, i.e. in order to inspect the predictive instance and determine the truth value of what it predicts, we perform the same steps as in the case of the falsifying instance, but this time with 'Predictive' selected in the table. Figure 54 shows the resulting screen.

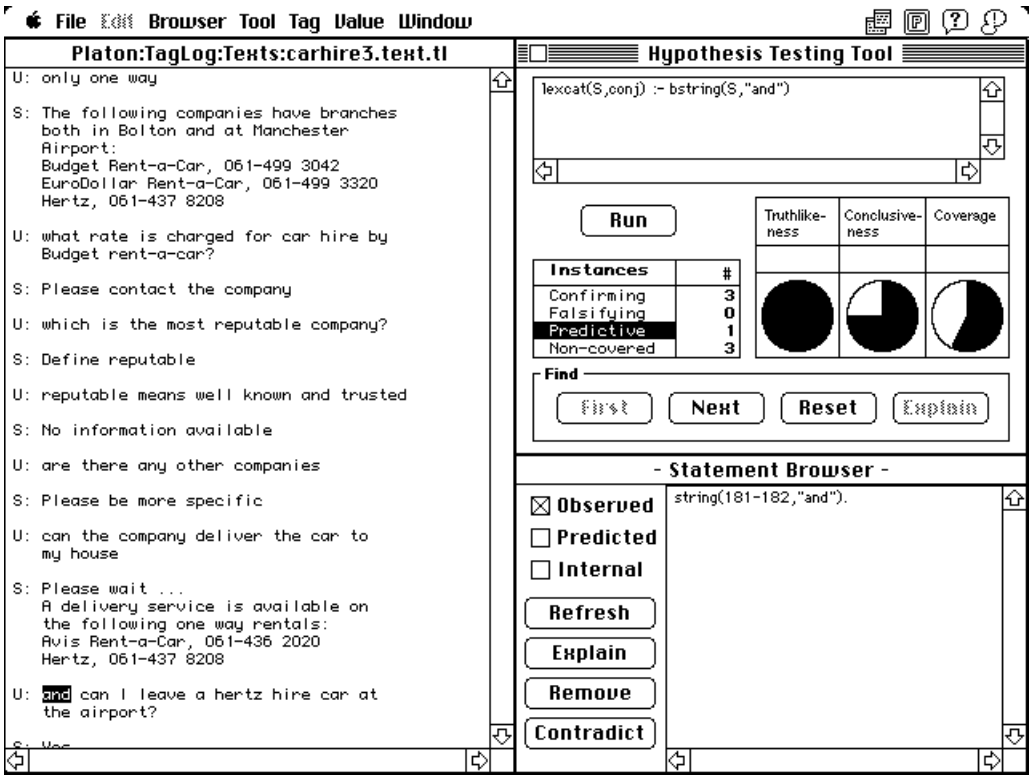


FIGURE 54. Testing a simple hypothesis (cont.)

The Statement Browser gives us a reason to suspect that we have simply forgotten to describe the selected instance of "and"! We repair the omis-

sion, and confirm the prediction, by means of a selection from the Value menu again. Now, if we re-run the test, the table and diagrams looks as in Figure 55.

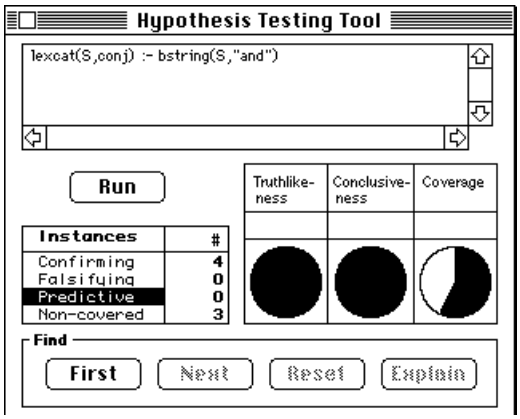


FIGURE 55. Testing a simple hypothesis (cont.)

Finally, suppose that we want to improve upon the coverage of the hypothesis. The first step is to check the cases where it fails to cover. We select 'Non-covered' in the table, and click the Find button in order to find the first segment which is described as a conjunction, but which is not an instance of the string "and". The screen now looks as in Figure 56.

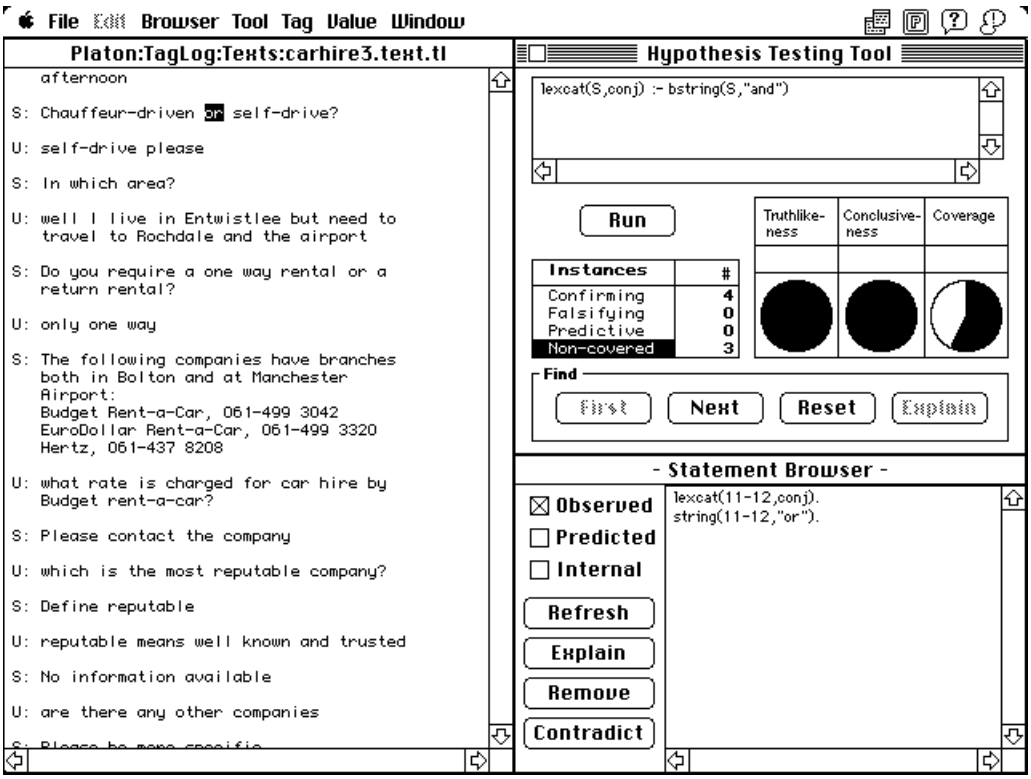


FIGURE 56. Testing a simple hypothesis (cont.)

We have found an instance of the string “or” which is described as a conjunction, and if we continue by clicking the Next button we will find instances of “but” described as conjunctions too. This suggests that we should modify our hypothesis so as to cover these cases:

```
(237)    syncat(S,conj) :- bstring(S,"and")
                         ; bstring(S,"or")
                         ; bstring(S,"but")
```

When we re-run the statistics with (237) in the text entry field, the tables and diagrams display as in Figure 57.

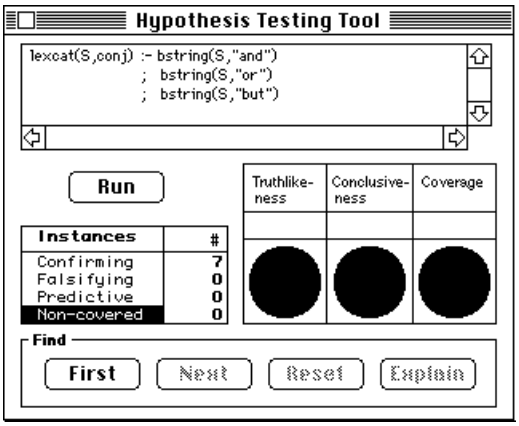


FIGURE 57. Testing a simple hypothesis (cont.)

What we have seen, through this simple example, is how the TagLog Hypothesis Testing Tool is able to support an interactive theory revision process, in which the system does the searching and counting, and presents the results in a way that provides hints for what should be done next, whereas it is the user who is responsible for the checking and confirmation/disconfirmation of predictions, and for performing modifications when the need for modifications arises.

12.7 Testing a Grammar Rule

Figure 58 shows a slightly less trivial example, in which grammar rule notation is used to express the hypothesis to be tested. A rule describing the hypothesis that a segment which is an instance of a pronoun or a proper noun, or which consists of a determiner followed by a noun, is

always a noun phrase, has been entered in the query box and the Run button has been clicked.

Note that the X-bar grammar in the background is consulted during testing. The (relevant part of the) background theory, i.e. the current theory minus the hypothesis, must be assumed to be true, or else there would be no guarantee that the figures and the pie-charts reflect the status of the hypothesis correctly.

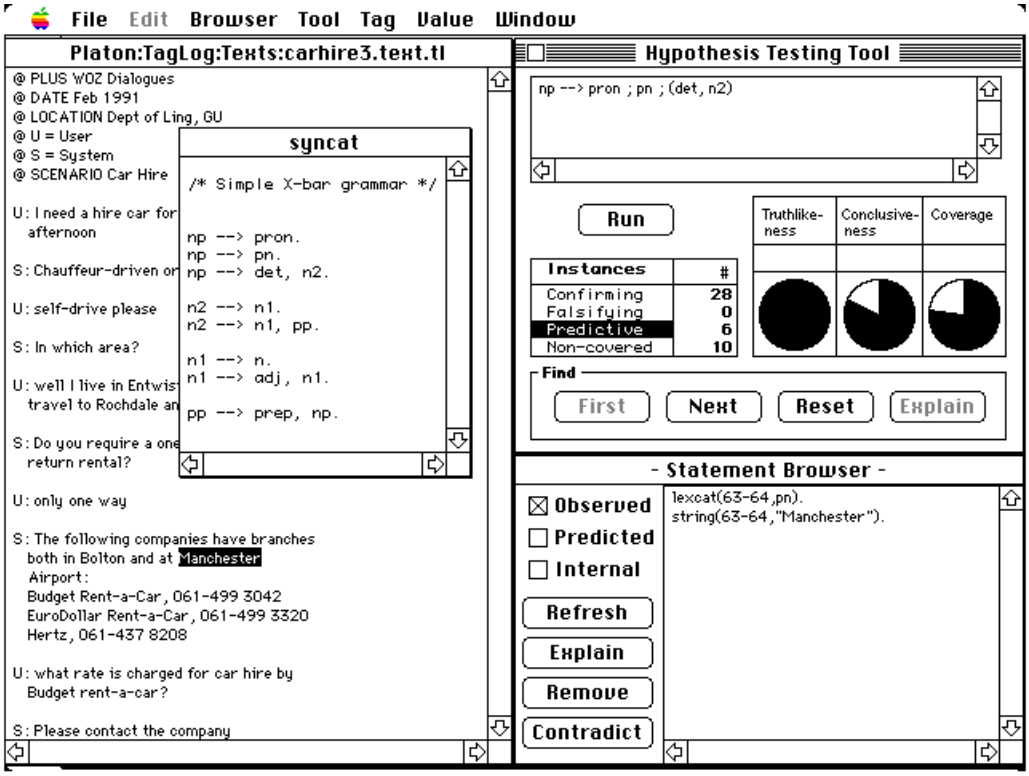


FIGURE 58. Grammar-rule testing in TagLog

As before, the resulting table and circle diagrams show the degree of success of the hypothesis. The user has decided to inspect the predictive instances in order to check the prediction, and has therefore selected the corresponding row in the table and clicked the First button, so that the instance in question has been selected in the Text window. The next logical step is to confirm or disconfirm the prediction by selecting the appropriate tag from the menus in the usual way.

Tracing through the cases not covered by the grammar, the user would find cases of noun phrase coordination, noun groups forming noun

phrases, noun phrases containing adjective-modifying adverbs, and chains of determiners. He would have to add rules to cover these.

12.8 Summary and Conclusion

In this chapter, the significance of *confirming*, *falsifying*, *predictive*, and *non-covered* instances of a hypothesis expressed as a universal statement was explained, and related to the notions of *truthlikeness*, *conclusiveness*, and *coverage*, and to the notions of *checking*, *confirmation*, *disconfirmation*, and *modification* of hypotheses.

I have also presented one of the tools available in the TagLog system – the TagLog Hypothesis Testing Tool – a tool for measuring the empirical support for universal statements. I have shown how such a tool, based on the notions mentioned above, implemented in a framework of computational logic, and equipped with a graphical user interface, can support the linguist in his work.

CHAPTER 13

Explanation

> Men do not think they know a thing till they have grasped the 'why' of it.
>
> *Aristotle*

13.1 Introduction

As was noted in the introductory chapters, the theoretical linguist is not necessarily content with knowing what syntactic properties a particular text segment has, knowing what it means, or knowing its function. The peculiar linguistic attitude towards features of texts amounts to trying to *explain* them, rather than just stopping at simple descriptions.

Explanation is one of the most important notions in the philosophy of science. In this section, I discuss and demonstrate the role played by explanation in the TagLog system, a discussion that to a certain extent will draw on the philosophy of science.

It is a fact of human psychology that we are not capable of knowing all the logical consequences of what we claim. In particular, the use of universal implicational statements in TagLog means that the truth of certain

statements may come as a surprise to the researcher, even when they follow deductively from claims that he has made himself. Another fact of human psychology is that we tend to demand explanations of statements that surprise us. Thus, a system such as TagLog ought to be able to explain its conclusions to the user. Fortunately, not only is a computer good at reasoning from premises to conclusions, it can also be made to produce explanations of how it reached the conclusions. Such explanations can take the form of presentations, in one form or another, of the proofs of the conclusions. Indeed, as we will see in this chapter, the user of TagLog is always free to demand an explanation of a particular statement implied by the current theory, and it is the task of the system to provide it.

Note that the term "explanation" is used in two ways here: the theorist explaining a phenomenon to the rest of the scientific community, and the TagLog system explaining a statement to the user. They have things in common, no doubt, but they should not be confused. From a practical point of view, explanation in the latter sense is clearly something that is useful, for 'theory debugging' purposes (including debugging of grammars), and in general for tracing out the connectedness between observations. Moreover, since a facility for explaining statements to the user seems to be able to promote an awareness, and perhaps also a deeper understanding, of some of the implications of particular theoretical formulations, it may be able to help the user to develop explanatory theories in the former sense as well.

13.2 Some Preliminaries of Explanation

The view I take of explanation is a fairly formal one, and in essence it is the one taken by, for instance, Hempel & Oppenheim (1948). According to this view, explanation is related to deducibility; it involves the deduction of the explanandum from a body of statements (explanans).

More specifically, the TagLog notion of explanation is the following:

ε Given is a set of TagLog statements, forming a theory. Among these statements is a set that describes a particular situation (e.g. a particular object, event, or state, or a particu-

> lar segment of text). One or more of these statements are singled out as demanding explanation. Explanation is provided by showing, by deduction, that a theory just like the one in question, but which does not contain the statements at issue, would entail these statements.

Let us for the moment assume that the description of the particular situation is complete, and that every relevant fact is specified. Later (in Section 13.5), I intend to show that this assumption can be abandoned, if *abduction* rather than deduction is recognized as the major reasoning strategy behind explanation (and that then, in fact, deductive explanation becomes a special case of abductive explanation).

Since the TagLog notion of explanation in some respects covers more, but in other respects less, than some other notions of explanation, implicit in common sense or explicated by philosophers of science, a number of comments are in order.

First, it must be made clear that ε is *not* meant to capture the common sense notion of explanation. How people *actually* explain things is of little concern to us here. Our notion of explanations is a 'rational reconstruction' (cf. Kim, 1967), and should not be expected to conform to our ordinary notion of explanation at every point. Nevertheless, (238) is given as an example of a kind of explanation that ε *is* meant to 'rationally reconstruct'.

(238) U: *What's that?*
S : *That's an NP*
U: *WHY is that an NP?*
S : *It consists of a DET followed by a NOUN*

To come back to ε, the formulation "but which does not contain the statements at issue" is there to rule out trivial proofs, of the 'from A follows A'–kind, that cannot be said to explain anything at all. That is, the idea is to dispense with 'explanations' of the kind given by S in the last utterance in (239).

(239) U: *What's that?*
S : *That's an NP*
U: *WHY is that an NP?*
S : *It is an NP*

An explanation in the sense of ε can of course only be relative to the formal system that provides the framework for description and questioning, in our case, a TagLog theory. Attempts to transcend the framework, such as the attempt made by U in the last utterance of (240), obviously cannot be supported. S provides the only explanation there is, given (say) a simple phrase structure grammar.

(240) U: *What's that?*
S : *That's an NP*
U: *WHY is that an NP?*
S : *It consists of a DET followed by a NOUN*
U: *Yes, but WHY is it an NP?*

Another thing worth noting is that the formulation in ε does not mention the notion of *causality*, or the notion of *law*. Hempel, and many other philosophers of science, assume that explanation is very intimately connected to these notions. But then grammars, and many other kinds of theories proposed by theoretical linguists, would have no explanatory power at all. Others feel that would mean ruling out too much.

> It seems to us that when one is diagnosing physical devices, of course explanations must be in terms of physical causality. But when we are working within an informational system, such as language or mathematics, then the relations are implicational and not necessarily causal.
>
> *Hobbs et al. 1993*

As I have already mentioned, the distinction between descriptive generalizations and definitions is not visible to TagLog. This extends to the explanation mechanism; TagLog is blind as to whether it uses a descriptive generalization or a definition in order to explain a statement in terms of other statements. Thus, a dialogue such as the one in (241) is perfectly possible.

(241) S : *John is a bachelor.*
U: *Why?*
S : *He is a man and he is not married.*

Explanations such as these are sometimes called *pseudo explanations*. Not filtering out such 'explanations' is consistent with the strategy of not making hard-to-apply distinctions visible to TagLog.

Another hard-to-apply distinction that does not matter to the TagLog explanation facility is the distinction between what is observable and

what is not. Indeed, I oppose the idea that explanation always has to relate some observed phenomenon to something which is not observable, some 'hidden' part of nature, a theoretical construct, or whatever. In my view, an explanation of something observable by something *else* observable is still a full-blown explanation. The purpose of an explanation could be described as one of *connecting* a set of seemingly disparate observations – making the set look *coherent* so to speak – and that *can* involve something unobservable, but does not have to.

And indeed, the TagLog notion of explanation does not even preclude an explanation of something unobservable by something unobservable, or, for that matter, something unobservable by something observable.

The formulation in ε imposes other restrictions on what can be explained in TagLog. Whereas in principle almost any statement in a theory may be doubted, and may be explained by being shown to be consequences of the rest of the theory, TagLog cannot handle all cases. In particular, universal statements cannot be explained in TagLog, at least not in a direct way (they can be *tested*, but not explained).

(242) S : *An NP can consist of a DET followed by a N*
U: *Why?*
S : *?*

Moreover, by definition, the *axioms* of a particular TagLog formalization cannot be explained (remember, they cannot explain themselves!). This is not a shortcoming of TagLog; no formal system provides a framework within which its own axioms can be explained. In TagLog, some of these axioms are typically accepted primarily on an ostensive basis – *this* is a noun, *that* is a verb phrase – so one could claim that these axioms are 'explained' by something outside the framework, but not in the standard sense of explanation. Also, note that far from all singular statements in TagLog are axioms: many observations find explanations within the theory.

13.3 The Logic of Deductive Explanations

Naturally, 'showing, by deduction' amounts to the giving of a proof. The TagLog notion of explanation, and the different *modes* of explanation that

TagLog support, are based on the notion of a proof, in the form of a *proof tree*.

It is straightforward to modify the reasoning engine presented in Chapter 10 so that it generates a proof tree if the evaluation of a goal succeeds. Collecting successful branches of the search tree as a proof tree is a well-known technique, described in (Sterling & Shapiro, 1986) for example. Here is one version of such a modified meta-interpreter:

(243)
```
prove(true,true).

prove(A,A/true) :-
        builtin_predicate(A), !,
        call(A).

prove(A,A/ BProof) :-
        clause(A,B),
        prove(B,BProof).

prove((A,B),(ProofA,ProofB)) :-
        prove(A,ProofA),
        prove(B,ProofB).
```

The code in (243) can be read as follows: To prove the formula *true*, do nothing; the resulting proof is *true*. To prove a built-in predicate, execute it; again, the proof is *true*. To prove a literal *A*, find a clause *A :- B* in the current theory, and prove *B*; the resulting proof is *A/BProof*, where *BProof* is the proof of *B*. Finally, to prove a conjunction *(A,B)*, prove *A*, and then prove *B*; the resulting proof is a structure of the form *(AProof, BProof)*, where *AProof* is the proof of *A* and *BProof* is the proof of *B*.

For example, given the theory in (244), the proofs in (245) are produced.

(244)
```
p.
p :- q, r.
q.
r.
```

(245)
```
!- prove(p, Proof).
        Proof = p/true ;
        Proof = p/(q/true,r/true)
```

As another example, grammatical this time, consider the theory in (246).

(246)
```
s --> np, vp.
np --> pn.
vp --> v, np.

syncat(1-2,pn).
syncat(2-3,v).
syncat(3-4,pn).
```

The proof tree in (247) is produced.

```
(247)   !- prove(syncat(1-4,s), Proof).
                Proof =  syncat(1-4,s)/
                                (syncat(1-2,np)/
                                        syncat(1-2,pn)/true,
                                syncat(2-4,vp)/
                                        (syncat(2-3,v)/true,
                                        syncat(3-4,np)/
                                                syncat(3-4,pn)/true))
```

In order to implement a procedure for explanation, we must remember – as was noted in Section 13.2 – that trivial proofs are not considered explanations, and thus what we require from an explanation is that no conjunct in the explanandum is part of the explanans. The first clause in (248) guarantees that no statement in the explanandum is proved in just one (trivial) step from a singular statement in the theory. The second clause is there since we must be able to explain conjunctions of goals. We explain a conjoined goal by explaining each of its conjuncts, resulting in a 'forest' of trees.

```
(248)   explain(A,Proof) :-
                prove(A,Proof),
                Proof \= A/true.

        explain((A,B),(AProof,BProof)) :-
                explain(A,AProof),
                explain(B,BProof).
```

For instance, given the theory in (244), the explanation in (249) produces only one solution, and the attempt at explanation in (250) fails altogether.

```
(249)   !- explain(p, Proof).
                Proof = p/(q/true,r/true)

(250)   !- explain(q, Proof).
                no
```

13.4 Explanation and Presentation

A proof tree is a rather abstract and terse representation of an explanation. To be useful, some thought must be given to the problem of *presenting* an explanation in a clear and persuasive way. In TagLog, explanations can be presented in different *modes*. The choice of mode is made by the user, depending on the situation and on the task at hand. In this section, I

present the different modes that are available in the current implementation of TagLog, and motivate their existence. The modes that will be discussed are the following:

- Graphical proof tree
- Explanation dialogue
- Analysis tree
- General statement
- General rule

13.4.1 Graphical proof tree

A graphical proof tree is perhaps the most direct way of presenting an explanation. It is a picture of the derivation, with the conclusion at the top, and with the branches containing premises stretching downwards. The leaves of a proof tree consist of either the atom ‘true’ or the atom ‘assumed’ (the meaning of branches terminating in ‘assumed’ will be explained in Section 13.6). Readers familiar with logic or logic programming have presumably seen such trees many times before.

For example, the proof trees produced in (245) and (247) display as in Figure 59, to the left and to the right, respectively.

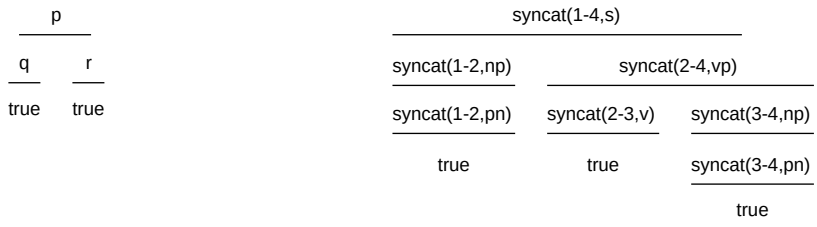


FIGURE 59. Graphical proof trees

One potential disadvantage with proof trees is that they may grow very large. Proof trees are always very detailed and explicit about every step in a derivation. This means that for any but the smallest proofs, huge trees are produced. Such ‘wordy’ and detailed explanations often work against their own purpose, which is to clarify matters, not to blur them.

What can be done about this? The following strategies are well-known in the literature: (1) we can restrict the explanation to one level at a time and allow the user to ask for more if necessary; (2) we can ‘hide’ certain

aspects of the information in each step; (3) very simple and to the user obvious steps in the derivation do not have to be explained; they can simply be bypassed. The modes presented in the following sections implement – or at least contain elements from – these strategies.

13.4.2 Explanation dialogue

Another way for the TagLog system to explain a statement – a way that implements strategy (1) above – is through a dialogue. The user has the initiative in such a dialogue. He asks for, and is presented with, instances of the universal statements that were used to directly derive the explanandum, and is invited to inspect (one at a time) the statements used in order to derive the premises of *these* statements, and so on.

Basically, this is a way to present the user with a ‘folded’ version of a proof tree, and to let him ‘unfold’ it himself, step by step, and in the particular order that he desires.

As an example, the dialogue beginning in Figure 60 has been initiated by the user selecting the statement *syncat(20-23,pp)* in the Statement Browser, and clicking the Explain button. The ‘:-’ symbol after the statement *syncat(21-23,np)* on the third row indicates that it can be further explained.

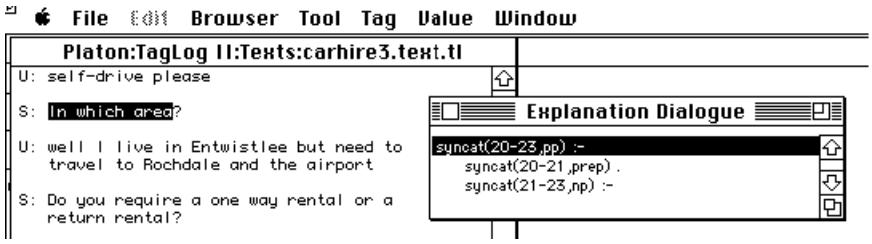


FIGURE 60. Explanation dialogue

The user decides to look closer at the internals of the noun phrase and double-clicked on the third row in order to do so. The screen is changed into Figure 61.

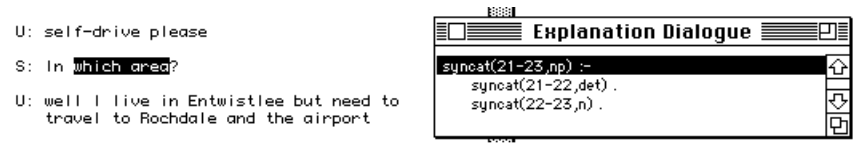


FIGURE 61. Explanation dialogue (cont.)

In Figure 62, the user has selected the statement *syncat(22-23,n)* and the system has selected the corresponding segment in the text. The dot after the statement indicates that it cannot be explained any further.

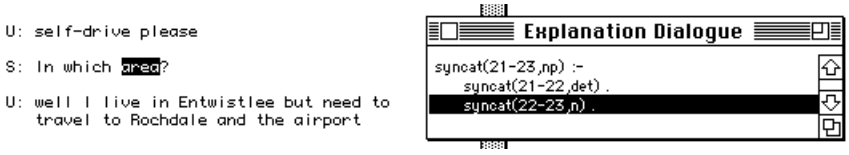


FIGURE 62. Explanation dialogue (cont.)

Note the important role played by *deixis* in this dialogue. The segment involved in the explanation is always automatically selected in the Text window.

13.4.3 Analysis tree

Some proof trees – the ones involving grammar statements only – can be transformed into something that linguists are more used to, namely *analysis trees*.

In a proof tree corresponding to a proof involving grammar statements only, all nodes have the form *P(S,V)*. The nodes in the corresponding analysis tree are formed by *V* only. An analysis tree is required to have leaves that are strings, so if the pre-terminal nodes of the proof tree do not involve strings (i.e. if *P* is not *bstring or string)* the strings corresponding to the segments *S* are used instead.

As an example, the analysis tree in Figure 63 corresponds to the proof tree produced in (247).

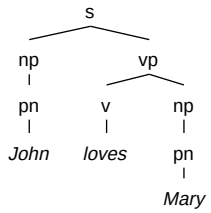


FIGURE 63. Analysis tree

Note that since analysis trees in TagLog are just ‘notational variants’ of proof trees, then, contrary to some of the basic assumptions of some modern theories of syntax (e.g. GB), analysis trees do not have any special ontological status. They do not reflect any property of the linguistic real-

ity; rather, they are reflections of our way of representing and reasoning about this reality.[1]

13.4.4 General statement

A very natural way to give an explanation is to provide it in the form of *one* universal statement that implies the explanandum, i.e. in the form indicated in (252) rather than in the form given in (251).

(251) U: *What's that?*
S : *That's an S*
U: *WHY is that an S?*
S : *It consists of an NP (which consists of a PN) followed by a VP (consisting of a V followed by an NP (consisting of a PN))*

(252) U: *What's that?*
S : *That's an S*
U: *WHY is that an S?*
S : *A PN followed by a V followed by a PN is always an S.*

Such a universal statement can be obtained by means of a machine learning technique called *Explanation-based generalization* (Mitchell, Keller, & Kedar-Cabelli, 1986).

Explanation-based generalization is a technique for automatically analysing and generalizing explanations consisting of proof trees in order to produce universal statements whose antecedents consist only of predications that are deemed *operational*.

In TagLog, predications are operational if they involve built-in predicates, or if they are declared as operational by means of the *operational/1* predicate.

For example, with a declaration of operational predicates as in (253),

(253)
```
operational(syncat(_,pn)).
operational(syncat(_,v)).
```

the statement in (254) corresponds to the proof tree in (247).

---

1. Note that there are modern theories of grammar that are not so much interested in constituent structure as such, notably HPSG and most versions of categorial grammar.

```
(254)   syncat(P0-P3,s) :-
            syncat(P0-P1,pn),
            syncat(P1-P2,v),
            syncat(P2-P3,pn).
```

The statement in (254) is derived from a proof involving grammar statements only, but the procedure works for any kind of TagLog proof.

Note that statements produced by explanation-based generalization follow deductively from the current theory, and therefore do not add any new information to it. However, since the use of specialized statements in derivations give shorter proofs than the use of more general statements, the output from explanation-based generalization may very well be used to replace other more general statements in the current theory, in order to trade a loss of generality for efficiency of use. (See Samuelsson & Rayner (1991) for an application of this technique to NLP systems.)

Moreover, it is certainly also conceivable that a theoretician finds that the specialized statement is sufficient, in the sense that it captures all relevant empirical regularities, and that the general formulation was too general, and can be dropped.

13.4.5 General grammar rule

When it comes to the explanation of *grammatical* facts, it is natural to give an explanation of why a segment must be regarded as an instance of a particular syntactic construction as *one* grammar rule that accounts for the construction, regardless of whether the rule is explicitly given in the grammar or not.

For example, the rule corresponding to the proof tree in (247) is shown in (255).

```
(255)   s --> pn, v, pn.
```

Note that this rule – in analogy to the general statement case – is not necessarily explicit in the grammar; it may only be the case that it follows deductively from the grammar. Such a rule may not be what we want to have in our grammar; often, grammar statements ought to be more general. But as an explanation, it is perfectly adequate, and very natural, to present the user with a specific rule like this.

13.5 Abductive Explanations

We will now consider the kind of explanation where the explanandum is deducible only if certain *assumptions* are made. Let (256) illustrate this kind of explanation.

(256) U: *What's that?*
S : *That's an NP*
U: *WHY is that an NP?*
S : *Assuming the instance of "the" is a DET, it consists of a DET followed by a NOUN*

This is a kind of *abductive* inference. It can be argued that the inference process that underlies explanation is abductive, and that 'explanation as deduction' is just that special case of 'explanation as abduction' where the number of assumptions being made is zero.

ε' modifies ε in order to account for the possibility of assuming statements instead of proving them.

ε' Given is a set of TagLog statements, forming a theory. Among these statements is a set that describes a particular situation (e.g. a particular object, event, or state, or a particular segment of text). One or more of these statements are singled out as demanding explanation. Explanation is provided by showing, by deduction, that a theory just like the one in question, which does not contain the statements at issue, *but which may contain a number of additional assumptions*, would entail these statements.

A brief explanation of the notion of abduction is in order at this point. Abduction can be characterized as a deductive process that simultaneously is able to generate the assumptions that are needed for deduction to succeed. Formally: when trying to prove *A*, hypothesize *B* if *A* :- *B* is known.

Abduction can sometimes be conclusive. This is the case, for instance, when we are dealing with a closed world. But in general, abduction is not conclusive. There may be multiple competing explanations, many candidate hypotheses.

The evaluation of the goodness of candidate hypotheses is an open research problem. Among the criteria for such goodness we find both absolute constraints and mere preferences:

- Sets of abductive assumptions inconsistent with the rest of the theory are disregarded altogether.
- Explanations requiring the fewest abductive assumptions, the shortest possible proofs, etc. are preferred.

Since TagLog is designed as an interactive system, constraints, but not preferences, are automatically enforced. It is up to the user to express *his* preferences by manually selecting one explanation from a set of alternative ones. As we will see in the sections to come, TagLog is designed to present the user with the alternative explanations, and to provide the means of choosing between them.

13.6 The Logic of Abductive Explanations

The TagLog abductive prover is implemented as a predicate *prove/2*. This predicate will be called with the first argument instantiated to a goal, and will bind the last argument to the resulting proof tree. As a side-effect, the current theory will contain those singular statements that the abducer needed to assume in order to make the goal follow from the background theory.

In fact, only a rather small change is needed in order to extend the deductive prover in (243) to allow abductive proofs: an extra clause is added so that instead of failing when a singular statement cannot be proved, the statement is assumed, by being added to the current theory.

The code for such a simple abducer can be given as follows:

(257)
```
prove(true,true).
prove(A,A/true) :-
        builtin_predicate(A), !,
        call(A).
prove(A,A/BProof) :-
        clause(A,B),
        prove(B,BProof).
```

```
prove((A,B),(ProofA,ProofB)) :-
        prove(A,ProofA),
        prove(B,ProofB).
prove(A,A/assumed) :-
        not A,
        assumable(A),
        ensure_grounded(A),
        consistent(A),
        assume(A).
```

The first four clauses are explained in Section 13.3. The fifth clause says that in order to prove *A* abductively and build a branch *A/assumed* in the proof tree, check that *A* is not already in the current theory, make sure that *A* is assumable, that it is ground (or can be grounded), and that it would be consistent to add it to the current theory, then assume it.

The predicate *ensure_grounded/1* must implement an appropriate strategy for instantiating variables before literals containing them are assumed. The *consistent/1* predicate is explained in Chapter 10. The *assume/1* predicate must implement a ‘soft’, backtrackable assert.

For example, given the theory in (258), and the declaration in (259),

(258)
```
p.
p :- q, r.
r.
```

(259)
```
assumable(q).
```

the proof in (260) is produced.

(260)
```
!- explain(p, Proof).
        Proof = p/(q/assumed,r/true)
```

Note that (257) is by no means an optimal implementation of an abductive prover. First, it relies on the database manipulation predicates of Prolog (hidden inside *assume/1*), which are known to be computationally costly. Secondly, the consistency checking procedure is very expensive too. These factors do perhaps contribute to making the abductive prover too inefficient to be of any real practical use. A description of the actual implementation and optimization of an abductive prover suitable for use in the TagLog system is beyond the scope of this thesis, however.

13.7 The TagLog Explanation Tool

13.7.1 The Explanation Tool layout

The TagLog Explanation Tool supports explanation of a ground goal, or a conjunction of ground goals.

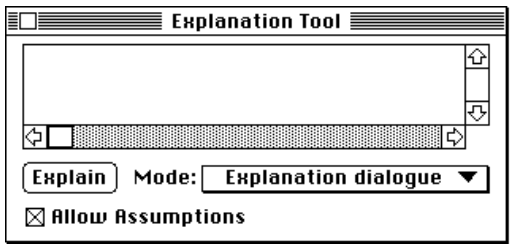


FIGURE 64. The TagLog Explanation Tool

The Explanation Tool contains one text entry field, one button, one pop-up menu, and one checkbox.

The statements to be explained are entered in the text field. They *can* be entered by hand in the text field, but they are usually placed there automatically, via a link from the Statement Browser.

Explain: Clicking this button produces an explanation of the statements in the text entry field, and presents it in the mode currently selected. If the 'Allow Assumptions' checkbox is selected, the explanation can be abductive.

Allow Assumptions: Select this checkbox if you want to allow the explanations to be based on abductive proofs.

Mode: The current mode of explanation is displayed. Select the mode you want from the menu.

Currently, the TagLog system supports five modes of explanation:

- Proof tree: The proof of the conclusion is displayed in its entirety in the form of a graphical tree.
- Explanation dialogue: The user is presented with the instance of the universal statement that was used to directly derive the conclusion, and is invited to inspect (one at a time) the statements used in order to derive the premises of *this* statement, and so on.

- Analysis tree: The proof of the conclusion is displayed as an analysis tree.
- General statement: The user is presented with a 'specialized' universal statement that allows the derivation of the conclusion in just *one* step from the singular statements of the theory.
- General rule: This is the general statement again, but formulated in the grammar notation.

13.7.2 How to use the Explanation Tool

Suppose that a segment in the current text is selected, that the Statement Browser is active and that it displays the statements describing the selected segment.

1. Among the statements in the Statement Browser, select one or more statements that you want the system to explain.
2. Click the Statement Browser's Explain button. A conjunction of the statements selected in the Statement Browser is sent to the Explanation Tool, and is explained in the mode selected there. That is, a dialogue is initiated, or a proof tree, a parse tree, a general statement, or a general rule is displayed.
3. Should more than one explanation be available, only the first one generated is displayed in step 2. The others can be generated on demand.
4. Should an explanation involve *assumptions*, they can be asserted by selecting the Assert Assumptions command from the Tag menu.

13.8 Explaining Syntax

Figure 65 shows the TagLog Explanation Tool in action. The segment corresponding to "In which area" has been selected in the text. The Statement Browser shows this segment to be a prepositional phrase, a fact derived from the grammar in the window on the left. The user has selected the corresponding statement in the Statement Browser, and clicked the Explain button in order to open the Explanation Tool. Three kinds of explanations have been generated, one at a time, according to different settings of the Mode menu: one explanation dialogue, one proof

tree, and one analysis tree. In this case, the ‘General statement’ mode and the ‘General rule’ mode (not shown above) would have presented the user with a grammar statement *syncat(P0-P3,p) :- syncat(P0-P1,prep), syncat(P1-P2,det), syncat(P2-P3,n).*, and a grammar rule *pp --> prep, det, n.*, respectively.

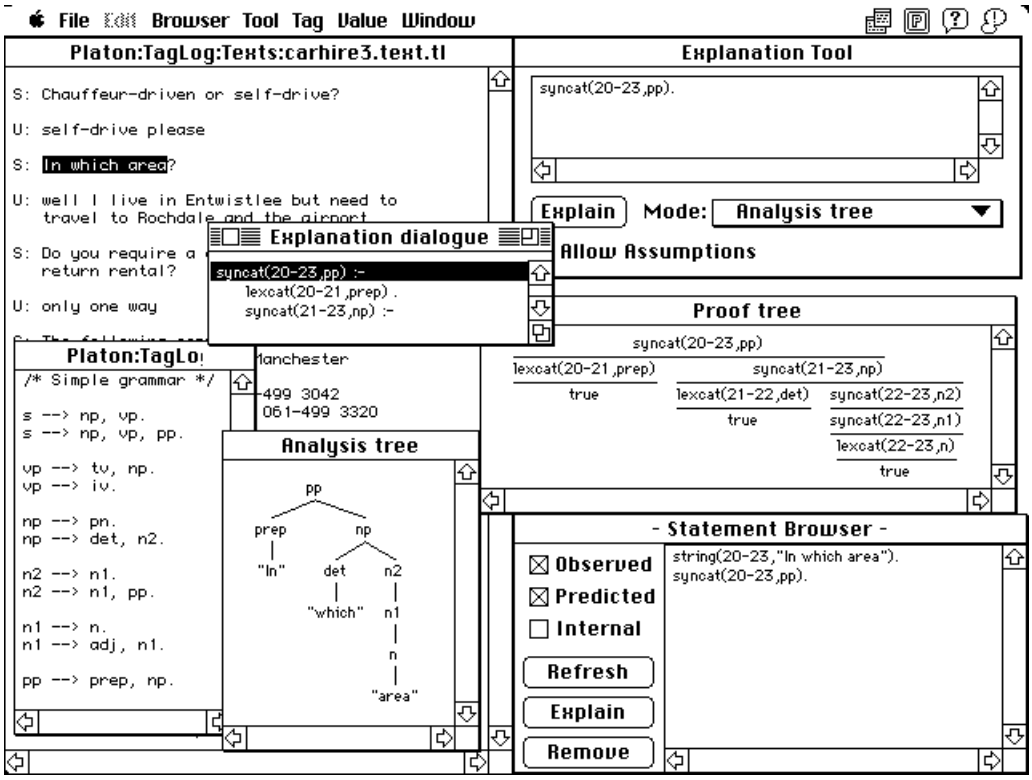


FIGURE 65. Explanation in TagLog

13.9 Explanation and Logic-Based Tagging

One of the advantages with the logic-based approach to automatic tagging sketched in Chapter 11 is that the TagLog explanation facility may be used for debugging theories from which the assignments of tags follow as logical consequences.

Suppose our current theory contains the following clauses:

(261) lexcat(S,C) :- bstring(S,W), lexicon(W,C), consistent(lexcat(S,C)).
¬ lexcat(P0-P1,inf) :- lexcat(P1-P2,det).

```
lexicon(“to”,inf).
lexicon("to",prep).
lexicon("the",det).
```

In Figure 66, a segment in the Brown corpus text has been selected so that a description of it has been displayed in the Statement Browser. One of the statements of this descriptions has been selected and the Explain button has been clicked. The Explanation Tool explains why the selected segment is a described as preposition: The selected segment is an instance of “to”, the lexicon lists “to” as a preposition, and the description of the selected segment as a preposition is consistent with the rest of what is known. Therefore, the selected segment is a preposition.

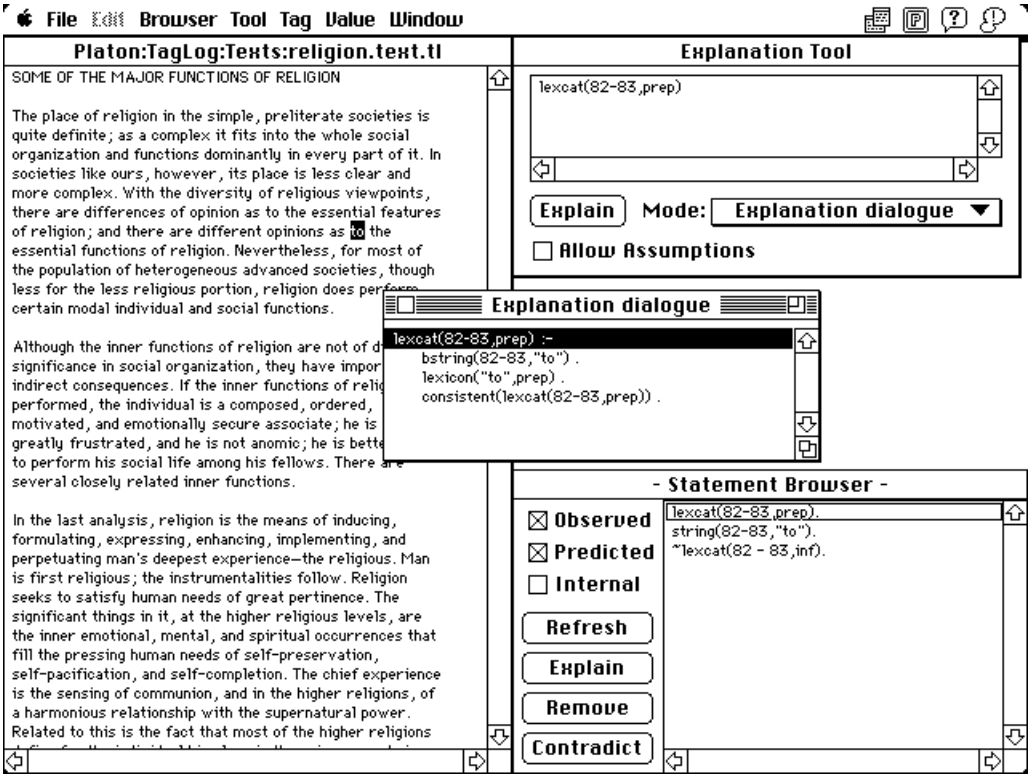


FIGURE 66. Explanation and automatic tagging

But why is it not an infinitive marker? Well, if we select the sentence ¬ *lexcat(82-83,inf)* in the Statement Browser and click Explain, we get the screen in Figure 67, where the Explanation Tool explains to us that the selected segment is not an infinitive marker, since the basic segment that follows immediately after is a determiner.

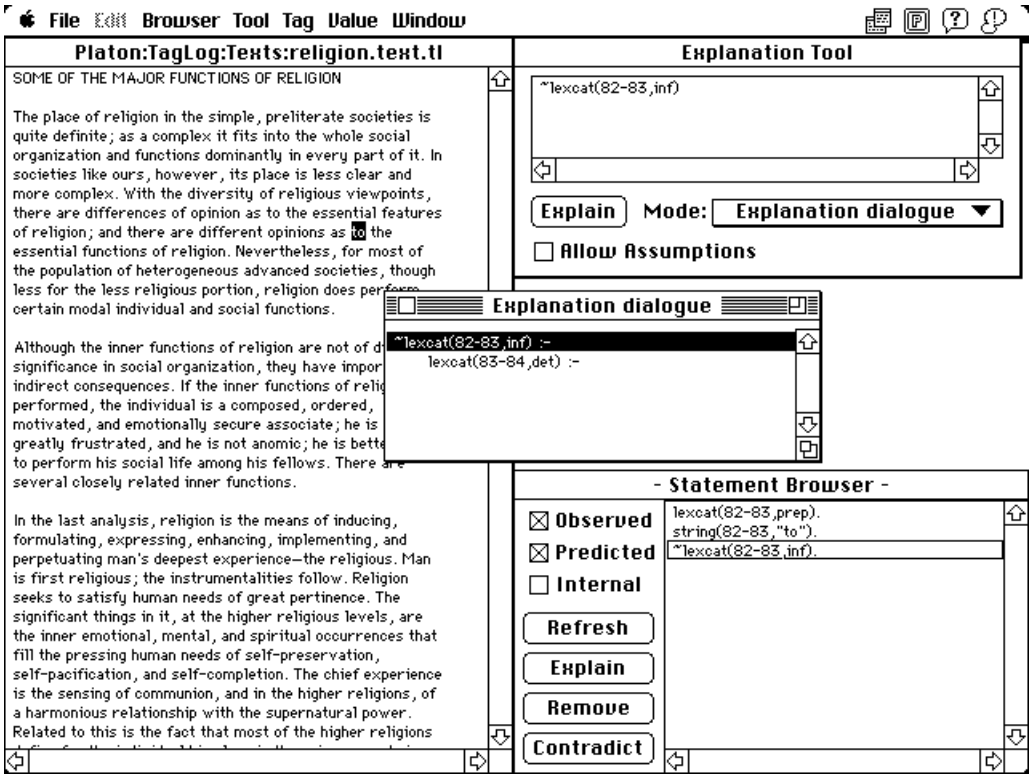


FIGURE 67. Explanation and automatic tagging (cont.)

Now, if the second row in the Explanation dialogue is double-clicked, the instance of “the” immediately to the right of the segment *82-83* will be selected, and the browser will contain an explanation of why *83-84* is a determiner. Thus, the Explanation Tool will always try to ‘put its finger on’ the segment currently being explained.

13.10 Lemmatization as Abduction

In the world of lexicology, a *lemma* is usually thought of as something that is constant across different inflected forms of a word, something closely related to the meaning of the word.[2] By *lemmatizing* we mean the process of categorizing (word-long) segments of texts as belonging to certain lemmas. This is by no means an easy task, at least not for a

2. Sometimes this is called a ‘lexeme’, while the term ‘lemma’ is used for the formal paradigm only.

machine, since there are lots of ambiguous forms like “leaves”, “saw”, and “question”. In this section, I present an abductive solution to the problem of lemmatizing.

In Section 4.7 (see example (85)) I hinted at a formulation of the logic of a lexicon that allows us to deduce strings or syntactical categories, given knowledge about lemmas. But if what we have is (positive) information about word forms and what we want to infer is information about lemmas (and this is the case when lemmatizing), deduction is useless. The implication arrow is pointing in the wrong direction, and we can deduce nothing.

Kowalski (1990) argues that if we are willing to employ *abduction* as a mode of inference, we can represent our knowledge in a natural and ‘true’ way, and still be able to infer what needs to be inferred. It can be argued, I believe, that for this reason the assignment of lemmas to segments ought to be performed by abduction, rather than by deduction. Thus we do not have to take to ‘arrow hacking’, to use an expression coined by Hobbs et al. (1993).

For an example in the abstract, consider the grammar in (262) (written in the style introduced in Section 4.7), the integrity constraint in (263), and the declarations in (264).

(262)
```
syncat(P0-P2,np) :- syncat(P0-P1,det), syncat(P1-P2,n).

bstring(S,"the") :- lemma(S,the).
bstring(S,"saw") :- lemma(S,saw), number(S,sg).
bstring(S,"saw") :- lemma(S,see1), tense(S,past).

syncat(S,det) :- lemma(S,the).
syncat(S,n) :- lemma(S,saw).
syncat(S,v) :- lemma(S,see1).
```

(263)
```
¬ lemma(S,X) :- lemma(S,Y), X \== Y.
```

(264)
```
assumable(lemma(_,_)).
assumable(tense(_,_)).
assumable(number(_,_)).
```

Suppose that we observe the following:

(265)
```
bstring(1-2,"the"), bstring(2-3,"saw"), syncat(1-3,np)
```

We would then be able to abduce the statement in (266).

(266)
```
lemma(2-3,saw).
```

Figure 68 is a proof tree corresponding to an abductive proof of (265). Note that the assumption in (266) is used twice in the proof.

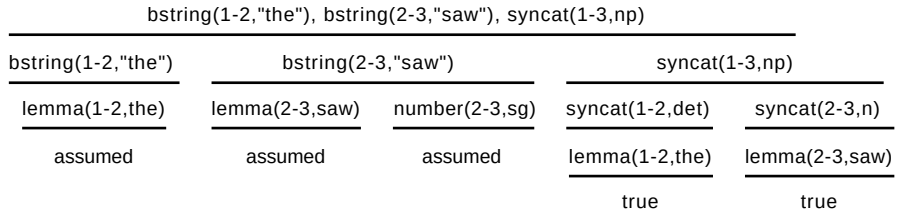


FIGURE 68. Lemmatization as abduction

This is in fact the only solution. Why are there no alternative (incorrect) solutions? Suppose *lemma(2-3,see1)* is assumed as a possible 'explanation' for *bstring(2-3,"saw")*. But according to the lexicon, this is a verb, and the grammar does not allow the last part of a noun phrase to be parsed as a verb. It does allow it to be parsed as a noun, however, but only if *lemma(2-3,saw)* is assumed. But then an inconsistency arises, due to the integrity constraints, which means that this assumption is actually not allowed. In fact, the only set of assumptions consistent with the integrity constraints is the one contained in the tree displayed in Figure 68.

In my opinion, this is a theoretically sound and quite elegant procedure for automatic lemmatization, informed by both a lexicon and a syntactic environment. It is also consistent with the common-sense view of what lexical disambiguation consists in: we *observe* that the segments in question are instances of the strings "the" and "saw", respectively, and we observe that they form a noun phrase. The only set of assumptions consistent with the observations (and various other constraints) is that the instance of "the" is a determiner and that the instance of "saw" is a noun (rather than a verb).

To give a more concrete example, lemmatization as abduction can be performed as follows in the TagLog environment. In Figure 69, the Statement Browser displays the statements that describe the segment selected in the Text window. With Predicted unchecked, and Internal checked, the user has selected the statement *bstring(1-2,"saw")* and has then clicked the Explain button in order to demand an explanation of this statement. Two different explanations have been found, and both of them are currently shown in the proof tree mode by the Explanation Tool. The user has

found that the second one is actually the correct explanation, and he is therefore about to tell the system to accept the assumptions *lemma(2-3,saw)* and *number(2-3,pl)*, and thus adding them to the current theory.

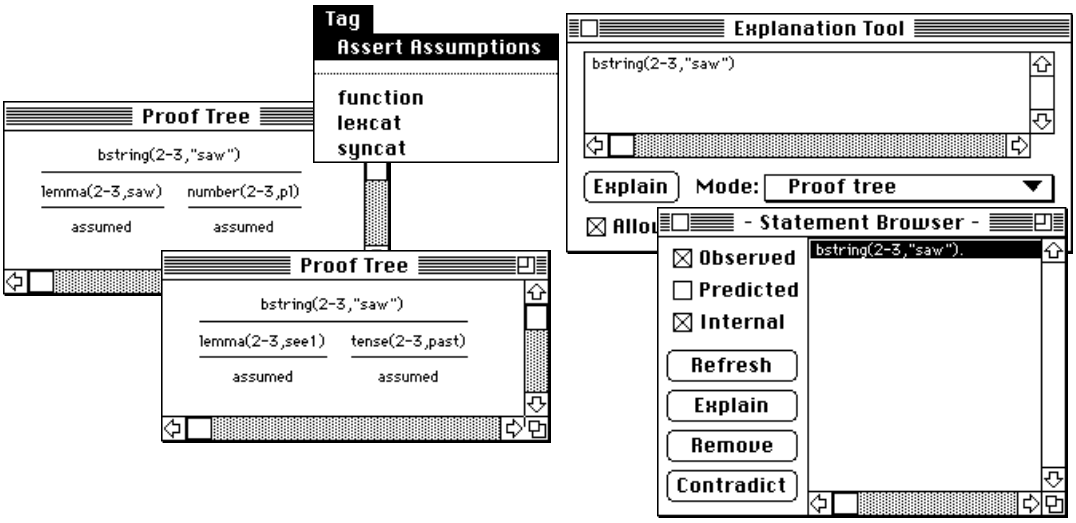


FIGURE 69. Lemmatization as abduction (cont.)

But the point of doing lemmatization as abduction does not become clear until the user selects a slightly bigger segment of text. In Figure 70, the user has done just that and has asked for an explanation of the statements that describe it. Only one explanation has been found, shown in the proof tree mode, and the user is about to tell the system to accept it. As a result, the statements *lemma(1-2,the)*, *lemma(2-3,saw)* and *number(2-3,sg)* are added to the current theory.

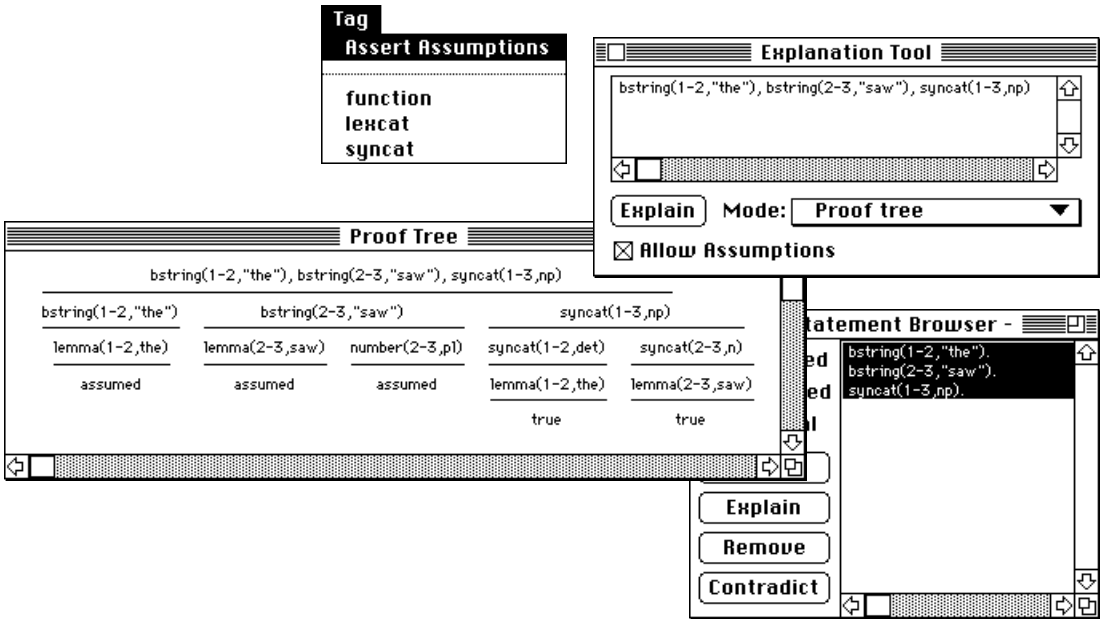


FIGURE 70. Lemmatization as abduction (cont.)

So we see that by looking at a larger segment of text at a time, the user did not have to make a manual selection between two alternative hypotheses; the system knew enough to be able to filter out the inconsistent alternative.

13.11 Interpretation as Abduction Revisited

In Section 4.8.2 an account of grammar integrating syntax, semantics and pragmatics was given. In this section I will demonstrate that the idea of interpretation as abduction by means of such grammars fits well within the TagLog framework, and in Section 13.12, I will show that the strategy can in fact be extended to cover more than just semantic/pragmatic interpretation.

The grammar from Section 4.8.2 is given again in (267).

(267)
```
s:E --> np:X, vp:X^E^A, {provable(A)}.
np:X --> det, n:X^A, {provable(A)}.
vp:A --> [“is”], pp:A.
pp:A --> p:X^A, np:X.
det --> ["the"].
n:X^cat(X) --> ["cat"].
n:X^mat(X) --> ["mat"].
p:Y^X^E^on(E,X,Y) --> ["on"].
```

Suppose the following holds in our situation theory (this is exactly the theory from Section 4.8.2, except that the clause *on(e1,x1,x2)* is absent here):

(268)
```
cat(x1).
mat(x2).
```

Suppose the utterance in (269) is encoded as (270), and that we have manually described *1-7* as in (271).

(269) the cat is on the mat

(270)
```
bstring(1-2,"the").
bstring(2-3,"cat").
bstring(3-4,"is").
bstring(4-5,"on").
bstring(5-6,"the").
bstring(6-7,"mat").
```

(271)
```
syncat(1-7,s:e1)
```

In accordance with the idea of interpretation as abduction we ask the TagLog system for an (abductive) explanation of *syncat(1-7,s:e1)* (e.g. by selecting that formula in the Statement Browser and clicking the Explain button). The formula can 'almost' be proven. The only exception is *on(e1,x1,x2)* which will have to be assumed. The resulting proof tree is depicted in Figure 71. So, in the terminology of Hobbs et al., *on(e1,x1,x2)* is the new information conveyed by (269), or, *on(e1,x1,x2)* is the *interpretation* of (269).

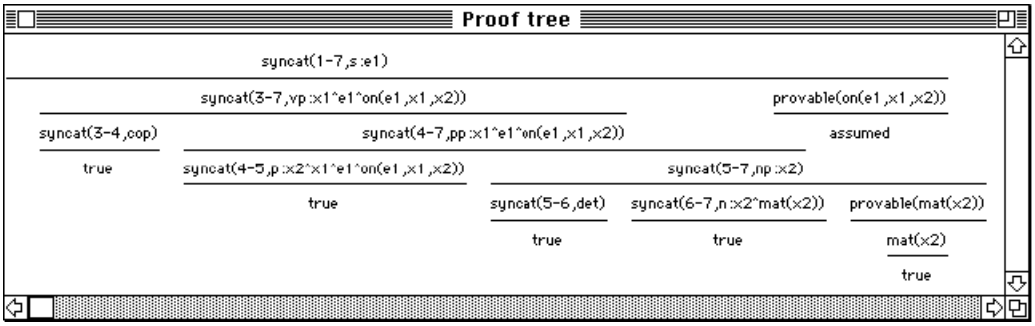


FIGURE 71. Interpretation as abduction

13.12 Knowledge Acquisition as Abduction

The interpretation of a text is *one* kind of knowledge of that text a linguist might be interested in, but the linguist is usually interested in gaining other kinds of knowledge as well. In this section, I try to show that explanation as abduction is more than just a mechanism for getting at semantic/pragmatic interpretations. Explanation as abduction can actually play the role as a general check for the coherence and completeness of a body of knowledge, and as a 'generator' of knowledge which will suggest *any* kind of 'fillers' – not just interpretations – of gaps of knowledge. In the case of the current example, if *mat(x2)* (but not *on(e1,x1,x2)*) was absent from (268), we would get the proof tree in Figure 72 instead.

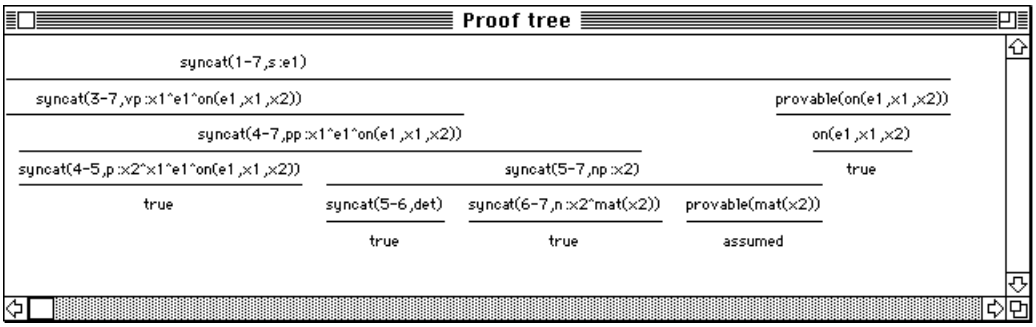


FIGURE 72. Knowledge acquisition as abduction

So we see that abduction is useful, not only for finding an interpretation, but also for finding out what we would need to assume about the background contextual knowledge in order to get a given interpretation. Thus, what we have here is actually a method and a tool for knowledge acquisition on the basis of text corpus input. In particular, of course, if further developed, it could become very useful as a way of acquiring knowledge for applications of interpretation as abduction.

13.13 Summary and Conclusion

To conceive of an explanation as a presentation, in one mode or another, of an abductive proof of the explanandum (or a deductive proof in cases where no assumptions are needed), has been shown to fill the need for explanation that has arisen as a consequence of the high degree of expressiveness of the TagLog formalism. It also provides a sketch for a workable solution to some problems of knowledge acquisition, notably lemmatization and world knowledge acquisition.

One thing worth noting is that explanations of the facts of linguistic reality are very similar to explanations of non-linguistic facts. This is no coincidence, but a consequence of recognizing that the two domains are really one and the same.

CHAPTER 14

Grammar, Truth, and Context

> Truth is a shining goddess, always veiled, always distant, never wholly approachable, but worthy of all the devotion of which the human spirit is capable.
>
> *Bertrand Russell*

14.1 Introduction

In Chapter 4, I argued that grammar rules are statements in disguise, and that grammars are systems of such grammar statements, or, in other words, theories. Thus – I argued – grammar rules (and therefore grammars) can be true or false, just like other kinds of statements. I proposed further to treat grammars as descriptions of segments of texts, or alternatively, as descriptions of the facts of texts. I saw no point in treating a grammar as a device to generate strings, or licensing trees, since neither strings nor trees have much to do with text, and since text is what we are dealing with in this thesis.

In the more tool-oriented chapters, I have isolated typical ‘errors’ that tools make, such as imprecision in the case of searching, and ambiguity in

the case of parsing. To these application-oriented errors can be added tagging errors in the case of automatic tagging.

In this chapter, I will show that, under my preferred interpretation, false grammar rules turn out to be surprisingly common in 'normal' grammars, and I will argue that they can be blamed for some of the errors that tools make. But there is hope, I will argue, since grammars can be made to better fit the facts of texts.

The conclusions drawn in this chapter are very tentative. I believe, however, that if they *are* true, they suggest something of a paradigm shift.

14.2 The Problem of False Grammars

Examples in some of the earlier chapters display a problem that at first sight seems to have rather serious consequences: common and perfectly normal grammar rules are falsified with respect to common and perfectly normal segments of text.

For a small and simple example of the problem, consider the rule in (272) and the utterance in (273).

(272) np --> n(pl).

(273) John loved the red apples

The rule in (272) predicts that the segment underlined in (273) is an instance of a noun phrase (since the underlined segment is a plural noun, and since the rule claims that every plural noun is a noun phrase). Now, either that segment *is* an instance of a noun phrase and the rule is confirmed, or it is *not* a noun phrase and the rule is falsified. Neither of the horns of this dilemma is very easy to accept. On the one hand, it is obvious from the co-text that the segment in question is not an instance of a noun phrase,[1] but on the other hand, (272) is a very traditional kind of

1. You doubt that? Suppose you hired someone to hand code a text on the level of syntax, would you still pay him if he tagged the occurrence of "apples" as a noun phrase in that context? Or do you think the Penn team would bank that as a noun phrase?

rule to have in a grammar, and to assume that it is false would be a radical thing to do. Yet, the latter is actually what I am going to propose.

Nothing is very special about the rule in (272). In fact, the rule in (274), rewriting a verb phrase into a transitive verb followed by a noun phrase, is also false, since the underlined segment in (275) is predicted to be a verb phrase, but is not.[2]

(274) vp --> tv, np.

(275) John <u>likes apples</u> and bananas.

Now, a false prediction, resulting from the application of a false rule, may give rise to further false predictions. For example, the segment underlined in (277) is predicted by the sentence rule in (276) to be a sentence, but is not.

(276) s --> np, vp.

(277) <u>John likes apples</u> and bananas.

Note, however, that (276) is not *falsified* by the segment underlined in (277), since a rule is falsified only in those cases where the consequent is false and the antecedent is true. In this case, as we have seen, the antecedent is not true of the underlined segment, so the rule is not falsified.

The problem with false grammar rules is perhaps even more apparent for the rules constituting a traditional DCG lexicon. Consider the following rule:

(278) syncat(S,v) :- bstring(S,"saw").

---

2. Let me offer an informal *reductio ad absurdum* demonstration of the fact that the underlined segment is not a verb phrase. Suppose it was a verb phrase. Clearly, the segments instantiating “likes apples and bananas” is a verb phrase, and just as clearly, “bananas” is a noun phrase. But then we would have to say that a verb phrase in conjunction with a noun phrase can form a verb phrase. But English does not ‘allow’ this (i.e. *vp --> vp, conj, np* is not a rule that we would like to have in our grammar, since such a rule would generate examples such as **John sleeps and bananas*). English allows the forming of a noun phrase by taking the conjunction of two noun phrases, and the forming of a verb phrase by concatenating a verb and a noun phrase. Thus, the segment instantiating “likes apples” in the example is not a verb phrase, but only a non-constituent *part* of a verb phrase.

No doubt, this statement is false. It claims that every instance of the string "saw" is a verb. But obviously, the instance of "saw" in (279) for example is not a verb, but a noun.

(279) John bought a saw in the hardware store

Adding the statement in (280) does not help.

(280) syncat(S,n) :- bstring(S,"saw").

On the contrary, all we get is a contradiction, since these two rules taken together say that every instance of the word form "saw" is a verb *and* a noun. But a particular instance of a word form can only be *either* a verb *or* a noun. So here again, I think that the most natural conclusion is that many lexical rules are false: there are cases where the antecedent is true whereas the consequent is false.

14.2.1 Grammars revisited

I suspect that not every reader accepting most of my approach so far is willing to accept my view of grammars, now that he has learned about some of the consequences of this view. Let us therefore backtrack and consider once again the relevant choice point. You could either:

1. Accept the view of grammar rules as sentences, accept the interpretation of them as strong statements about text *segments*, and accept that grammar rules which under other interpretations are true might be false under this interpretation.
2. Accept the view of grammar rules as sentences, accept the interpretation of them as statements about text segments, but not as statements of a strength that gives rise to false instances. Default logic interpretations or probabilistic interpretations are possibilities that could be explored here.
3. Accept the view of grammar rules as sentences, but not the interpretation of them as statements about segments. Instead, one could think of them as expressing statements about *strings*. As such, they might stand a better chance of being true.
4. Reject the view of grammar rules as statements. Then they *cannot* even be true or false, and thus, the problems as I have stated them above cannot arise. One could think instead of grammar rules as string generating devices, or as licensers of trees, just to mention two traditional interpretations.

As I have clearly indicated, the first option is my choice here, but I realize that some readers might prefer one of the others. Now, I do agree that to take the first option would be unwise, unless one saw a way to improve upon false grammars, a way to make them more truthlike. Indeed, I see at least one such way, to be explored later in this chapter. Moreover, I think this option is more interesting than the other options because, as I shall argue, it appears to have found a point where the quest for truth and the quest for usefulness seem to coincide.

One disadvantage with the second option as compared to the first, from my perspective, is that whereas the first option can take the DCG formulation of grammar rules as universally quantified sentences seriously (cf. the quote from Pereira and Shieber (1987) on page 72), the second option cannot. I think this formulation of DCG deserves to be taken seriously. Another problem with the second option, from a methodological point of view, is that if we weaken the universally quantified statements they will in general become too weak to be falsifiable. Counter examples will have no bite, and a major driving force behind theory development will be missing. Still, the second option is interesting, and I will experiment with something along this line in Section 14.4.2, but it remains to be worked out in detail, and I think that in general weaker interpretations should not be explored until stronger ones have been tried and found to be lacking. Finally, note that the first and the second option are possible to combine: some grammar rules can be given a strong interpretation, others a weaker.

Considering the fact that we are interested in concrete occurrences of sentences, as they occur amidst other concrete occurrence of sentences in real corpora, it seems that the third option is too tied up with a view of a text as strings. However, for those who *prefer* to treat the elements of texts as context-free objects, this would be the way to go. The *string* “apples”, considered as a decontextualized abstract entity, might be a noun phrase, for all I know, and therefore a grammar that talks about strings might be correct in describing that string as a noun phrase. For a proponent of this choice the problem of false grammar rules may simply not arise. The DCG idea comes in very natural here too, and DCG as a grammar of strings might even be more in the spirit of what is said in the quote from Pereira and Shieber.

The fourth option, finally, is an option for those who would rather not speak of grammar rules as statements at all. Thus it is hardly compatible with the approach advocated in this thesis.

Now then, I hope I have persuaded the reader to give the views of grammar rules as statements about text segments a chance, so let us carry on with our investigation.

14.3 Errors Tools Make

In the more application-oriented chapters, we have isolated typical 'errors' that tools make, such as imprecision in the case of searching, and ambiguity in the case of parsing. To these errors can be added tagging errors in the case of automatic tagging. In fact, the existence of such 'errors' is the main motivation for the present chapter, since I do not want to leave the reader with a system that seems to malfunction. Instead, I blame these errors on the grammars. They can be corrected, although not necessarily by me.

14.3.1 Parsing and the problem of ambiguity

In Chapter 8, a reminder was given of *ambiguity* as one of the 'classic' problems of parsing. In the same chapter, I also distinguished categorial ambiguity from structural ambiguity. It should be added that structural ambiguity is usually seen as *the* problem of ambiguity: with realistically sized grammars and input strings, often hundreds, or even thousands, of parse trees are generated for each input string. But then again, as has been noted (e.g. by Gazdar & Mellish, 1989, p. 170), categorial ambiguity often gives rise to structural ambiguity. I believe a case can be made for the claim that false grammar rules underlie both kinds of ambiguity.

Now, at this point, let me introduce another distinction. I will speak of *X* as *inherently* ambiguous if *X* is ambiguous 'in itself'. If *X* is not inherently ambiguous, but still ambiguous, *X* is merely *apparently* ambiguous.

In the expression 'X is ambiguous', for what can X be substituted? We can speak of *strings*, *segments*, *grammars*, and *languages*, as being ambiguous.

A *string* is categorically and inherently ambiguous if it has at least two instances (i.e. if there are two segments) that are also instances of different and incompatible grammatical categories. *Word forms* are often ambiguous in this sense. For example, the word form "saw" is categorically ambiguous since some of its instances are nouns, and some of them are verbs. For a non-ambiguous word form it can very easily be determined to which grammatical category its instances belong. The word form "Eskimo" is categorically unambiguous since its instances always fall in the noun category. Find an instance of "Eskimo" and you will know immediately that it is a noun.

On the other hand, *segments* (utterances) are seldom inherently ambiguous. When we speak of a segment as being ambiguous we usually mean that it has received (at least) two different and conflicting descriptions of 'the same kind', and that we are unable to decide between them.[3] So the segment *appears* to be ambiguous. But, in most cases, such an ambiguity is an ambiguity *only* in the eyes of the (uninformed) beholder. Given some more knowledge of the context, such an ambiguity can almost always be resolved.

Finally, in formal-language theory at least, the notion of ambiguity is applied to *grammars* and *languages* as well:

> A context-free grammar *G* such that some [string] has two parse trees is said to be *ambiguous*. [...] A [context-free language] for which every [context-free grammar] is ambiguous is said to be an *inherently ambiguous* [context-free language].
>
> *Hopcroft & Ullman (1979, p. 87).*

These notions of ambiguity of grammars and languages will concern us no more in this work, however. Nor will ambiguity of strings matter much. Rather, we will concentrate on apparent ambiguities of segments.

Now, from a logical point of view, there are (at least) two ways of conceiving of this kind of ambiguity. Following Johnson (1994), I will call them the 'disjunctive treatment' and the 'conjunctive treatment', respectively.

On the disjunctive treatment, a segment *S* is (apparently) ambiguous if the formula in (281) is true.

---

3. A *segment* is categorically ambiguous if it is an instance of a *string* that is categorically ambiguous and no other information is brought to bear on the case.

(281) !- provable(((syncat(S,C1) ; syncat(S,C2)), incompatible(C1,C2))),
unprovable(syncat(S,C1)), unprovable(syncat(S,C2)).

Thus, in this approach, ambiguity is reflected by a disjunction of sentences, where none of the disjuncts are provable as such. (We assume here that disjunctions are provable without the disjuncts being provable, something which is not possible in TagLog, nor in Prolog.)

On the conjunctive treatment, a segment *S* is categorically ambiguous if the following is true.[4]

(282) provable((syncat(S,C1), syncat(S,C2), incompatible(C1,C2))).

Thus, ambiguity is reflected by a conjunction of sentences, each describing a different analysis.

I believe Johnson is wrong when he argues that “the disjunctive treatment of ambiguity [...] is in some ways more natural [than the conjunctive treatment], since an ambiguous utterance (with perhaps the exception of jokes involving ambiguity, etc.) probably has only one interpretation, even though it is not clear from the words in the utterance which interpretation that is.” (Johnson, 1994, p. 243).

That is, I believe Johnson is right in claiming that an ambiguous utterance probably has only one interpretation, but I think he is wrong in his disapproval of the conjunctive treatment for that reason. I think it is just as natural – and in some ways more appropriate – to conceive of an ambiguity as a conjunction, *provided we regard it as inconsistent*. A segment is ambiguous when it has received an *inconsistent*, and thus necessarily *false*, description. If *A* and *B* are inconsistent with each other, it means: ‘they can’t be right at the same time, one of them must go, we just don’t know which yet’.

To put it slightly differently, the treatment of ambiguity as an inconsistent conjunction fits well with the fact that ambiguity is something unwanted, something bad, something that should be signalled as an error. In contrast, a disjunction gives us no cause for worrying; why change it, if it is true?

---

4. Note that we couldn’t have \== instead of *incompatible/2* here: *np* \== *n* holds, but *np* and *n* are not incompatible.

In accordance with the view of ambiguity as inconsistency and error, *ambiguity resolution* becomes a process of recovery of consistency, a process of error elimination.

To fight ambiguity, computational linguists have pursued many different approaches. To name just a few, computational linguists have built *modular* systems, like the one depicted in (A), that impose a division of processing labour between a parser and a disambiguation module. Or they have used an *incomplete* parsing strategy that does not compute all the logical consequences of the grammar and (ideally) in particular not the ones that are false (B). Or they have tried to compute representations that *preserve the ambiguity* of the input string (C)[5]. Or, they have tried to combine some such approaches.

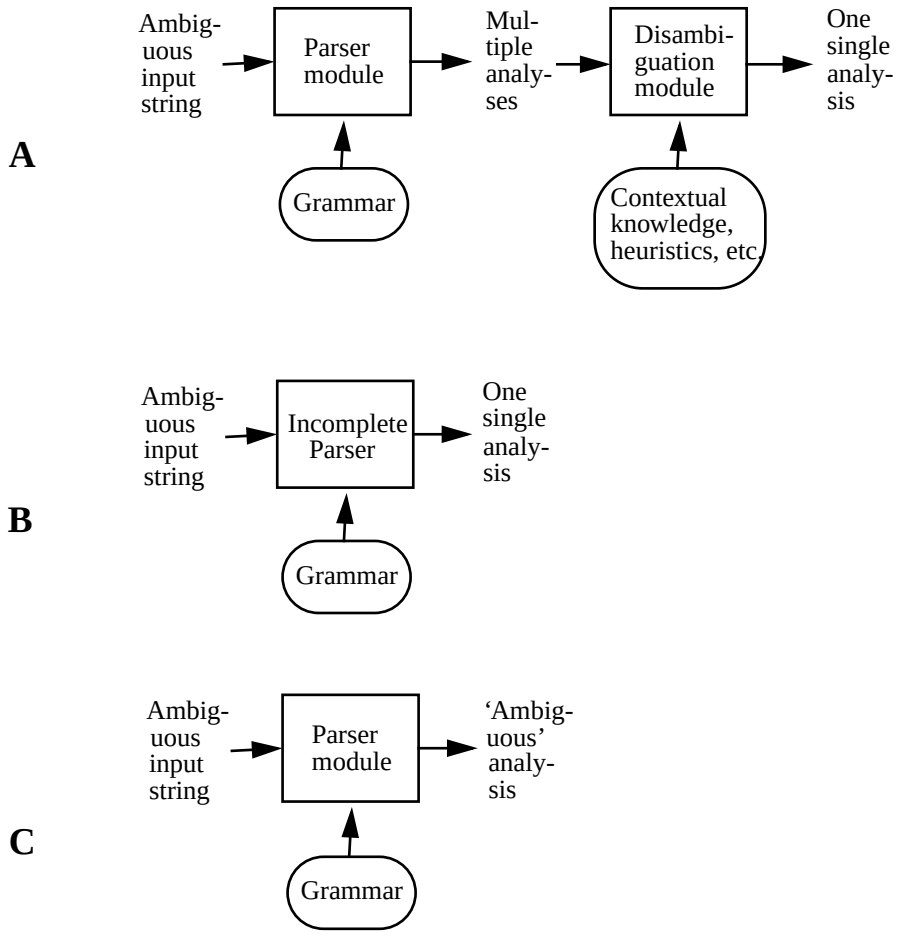


FIGURE 73. Three ways of handling ambiguity

5. See for example (Westerståhl, Haglund, & Lager, 1993) for one such approach.

With the right kind of information in these boxes, and a proper strategy for applying it, these approaches are all practical solutions. But as I see it, they are not *principled* solutions. A principled solution would attack the problem at its root and improve the truthlikeness of the grammar instead, alleviating the need for the disambiguation module completely.

To sum up, I have argued for a view of the problem of ambiguity that makes it intimately connected with the problem of false grammar, and which seems to suggest that solving the problem of false grammar rules would be *one* way of solving the problem of ambiguity.

14.3.2 Searching and the problem of imprecision

Chapter 7 mentioned lack of precision as one of the classic problems of searching (the other classic problem being lack of recall).

A statement meaning “a determiner followed by a noun is always a noun phrase” can be interpreted procedurally as saying “to find a noun phrase segment, find a determiner segment followed by a noun segment”. Now, with a false rule, it will sometimes happen that when we have found the segments specified by the antecedent, we still have not succeeded in finding what is specified by the consequent.

Hence, the problem of lack of precision of searching can also be traced to the problem of lack of truthlikeness of rules. No wonder that, if the map is not correct, we occasionally end up in places where we did not want to go.

Chapter 7, however, also mentioned one approach to searching for instances of grammatical categories that does not suffer from lack of precision: the treebank approach. I like to think of this approach as the ‘treebank trick’. Here’s why: The reason it does not suffer from the imprecision problem is that the errors that have arisen as an effect of using a false grammar have been corrected during a rather elaborate preprocessing phase, consisting of parsing by means of a (false!) grammar and then manually editing all the ambiguous output. The process is depicted in Figure 74.

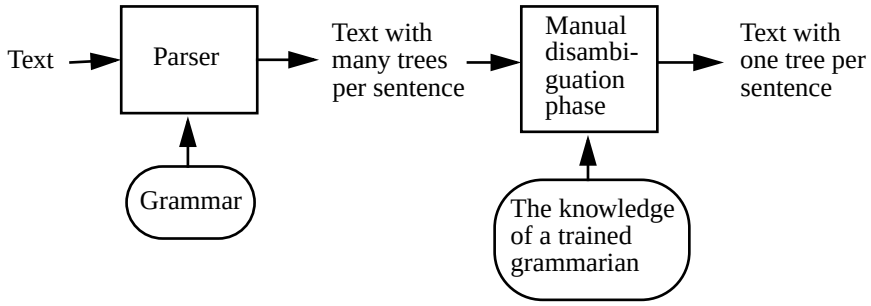


FIGURE 74. A treebank factory

So the grammar is not *used* in the search process itself. It is used only at an earlier stage, to parse the text into that forest of trees in which the search is then performed. This parsing process is of course affected by the problems discussed in Section 14.3.1; that's why the disambiguation phase is needed.

As was shown in Chapter 7, the treebank trick is simple to implement in the TagLog system; besides that it means of course a lot of labour intensive manual pre-processing. But one may suspect that there are other less resource demanding ways of implementing the searching by means of grammars, by using the grammars in a less indirect way.

The treebank approach works well, but it is a solution tailored to just one task out of many. The imprecision problem is solved, but nothing is done about the root of the problem: the falsity of grammars. By definition, a more truthlike grammar means more precision. So to improve upon searching, we ought to try to find ways of making our grammars more truthlike.[6]

---

6. There are other ways of improving the precision of searching by means of a grammar, *without* having to improve the truthlikeness of the grammar. One idea is to search for the *maximal segment* for which a description is true. Applied to the problem in (273), we would find the segment corresponding to "the red apples", but not the segment corresponding to "apples". The problem with this solution is, of course, that it means trading recall for precision: looking for noun phrases, we would find the segment corresponding to "apples and bananas" but not the segment corresponding to "apples" or the segment corresponding to "bananas". Nevertheless, for those willing to pay that price, TagLog provides the *max_provable/2* built-in predicate, defined in such a way that *max_provable(S,Goal)* is true iff *Goal* is provable with *S* instantiated to a segment expression, and there is no proper subsegment of *S* such that *Goal* is provable.

14.3.3 Automatic tagging and the problem of tagging errors

Many methods for automatic tagging of parts-of-speech, or of syntactic categories, have been developed in recent years. In this thesis, a whole chapter has been devoted to such methods. Here, I will only point to the rather self-evident connection between the use of a false (and even inconsistent) lexicon and tagging errors. The idea of using a DCG encoded lexicon for automatic tagging of parts-of-speech – an idea which in itself is quite natural – means that each instance of (say) “saw” will be assigned contradictory syntactic categories, and only one of them is correct. Again, if we did have access to a lexicon with rules that were more truthlike, this would be less of a problem.

14.3.4 Truth and Usefulness

Three seemingly different types of errors are traced to the same cause: false grammars. We have seen that some grammar rules are neither completely true nor maximally useful. I contend that they are not maximally useful *because* they are not true. There are strong reasons for assuming that a grammar that is nearer to truth is a more useful grammar for certain purposes. In particular, as I will show, it is rather obvious that a more truthlike grammar makes possible less ambiguity when parsing, more accurate searching, and fewer errors when tagging.

I am not necessarily advocating a revolution. I am not saying we should discard all our old grammars, simply because they are false. Even a false grammar might be very useful, as a component in one of the traditional parsing systems depicted in Section 14.3.1, for example, or, if we are willing to put up with the imprecision, as a component in a searching system.

It is also important to note that given certain background restrictions imposed by certain applications, often concerning input and output, some false grammar rules will seldom or never get a chance to make false predictions, and hence will seldom or never give rise to the particular kinds of errors in these applications that would have arisen without such restrictions. For example, consider a parser which can always rely on getting the whole of its input string at a time and which is expected to deliver (at least) one well-formed parse tree covering the whole of that string. Suppose “John likes apples and bananas” is fed to this parser. Then the erro-

neous parse of the first three words as a sentence (built by means of the falsifying instance of the rule in (274)) will never show up, simply because that parse has not consumed the whole input. I believe the existence/non-existence of (usually implicit) background restrictions like these can partly explain why parsing can be as successful as it sometimes is,[7] despite false rules, and why another potential application of the same set of rules, such as searching for phrases, will sometimes fail to produce correct results. Hence, careful design of restrictions like these may help to produce successful applications using grammars that are strictly speaking false.

I believe, however, that a conservative approach to grammars will not result in grammars as useful as they could be. Moreover, we have to be very careful when dealing with false grammars, since whereas a true grammar would be useful in any direction, for any purpose, a false grammar must be supplemented by a mechanism for handling the errors that arise as a consequence of using it. The conservative approach necessitates a lot of pre-processing (building treebanks in the case of searching) or input-output restrictions (as above in the case of parsing) and post-processing (disambiguation in the case of parsing) in order to reduce the amount of errors due to false grammars.

14.4 How to Fix a False Grammar

The way to make grammars more truthlike is, of course, to change them. Let me distinguish between two kinds of changes in a grammar: incremental change, and revolutionary change. An incremental change of a false grammar into a grammar somewhat ‘less false’ may simply involve changing one of its grammar rules, by either logically strengthening its antecedent, or logically weakening its consequent. A revolutionary change in a grammar may involve a total restructuring of information, perhaps even assuming the existence of novel ontological categories. Another way to put it is that in the latter case we are changing our *grammar theory*, rather than changing our grammar.

7. If we dare speak of success in the light of thousands of parse trees for a single input segment...

How come some grammars are false? I believe that the following observations reveal at least part of the truth. That grammar statements will make some false predictions is only to be expected, since they lack the information of the *context* that is necessary to guarantee true predictions in all cases.

By their very nature, traditional phrase structure grammar rules are insensitive to *co-text* in a very precise sense: the right-hand side is not able to 'look' outside the segment covered by the category (predication) on the left-hand side. Nor do traditional grammar rules have access to the *situational context*. Predicate symbols and terms in a grammar rule statement always refer to entities in the text, or to something on an abstract type level.

A 'performance grammar' that treats utterances the way sentences are treated by a 'competence grammar', i.e. as context-free objects, will be false, almost by necessity.

14.4.1 An integrated approach to context sensitivity

I will show, in this section, that integration of syntax, semantics, and pragmatics, in the manner proposed by Hobbs et al. 1993, may lead to more truthlike grammars.[8] By adding to the antecedents of grammar statements, conditions that refer to (situational) contextual knowledge, grammar rule statements are made more sensitive to that context, and as a consequence, they may become more truthlike, and thus, ambiguity will become less of a problem. Ideally, we would always have the situation depicted in Figure 75.

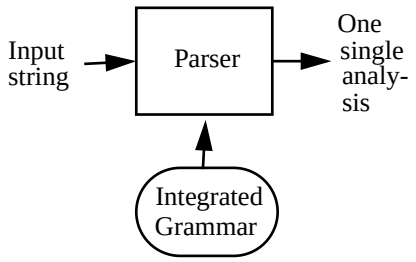


FIGURE 75. An integrated system with a truthlike grammar

8. This, indeed, is something that we might regard as new evidence for the soundness of the integrated approach of Hobbs et al.

I will illustrate the effect of the integrated approach on the following pp-attachment ambiguity:

(283) Mary saw a man with a dog

A grammar containing the rules in (284) would assign two analysis trees to this utterance, corresponding to whether the *pp* attaches below the *np* node or below the *vp* node in the tree.

```
(284)  s --> np, vp.
       np --> det, n.
       np --> det, n, pp.

       vp --> v, np.
       vp --> vp, pp.

       pp --> p, np.
```

When due consideration is given to content and context of the utterance in (283), it is easy to see that only the first of these analyses is correct. (That is, Mary saw a man walking a dog. She did not see a man *by means* of a dog.)

Thus, suppose that the situation theory describing the content/context of the particular utterance in (283) contains the following clauses:

```
(285)  named(x6,mary).
       saw(e2,x6,x4).
       man(x4).
       with(x4,x5).
       dog(x5).
```

The integration of semantics and pragmatics with the theory in (284) proceeds as follows. First, separated from the syntax by a colon, compositional semantics is added to the categories of the rules. This, as such, does not make rules more sensitive to context, however. But then, by adding, to the antecedent of the rules, (meta-logical) conditions that check whether (some of) the results of compositional semantics processing is really part of the semantic content of the text or allowed by the current context, we obtain the rules in (286).

```
(286)  s:E --> np:X, vp:X^E^A, {provable(A)}.

       np:X --> [mary], {provable(named(X,mary))}.
       np:X --> det, n:X^A, {provable(A)}.
       np:X --> det, n:X^A, pp:X, {provable(A)}.

       vp:A --> v:X^A, np:X.
       vp:X^E^A --> vp:X^E^A, pp:E.

       pp:X --> p:X^Y^A, np:Y, {provable(A)}.
```

```
det --> [a].
n:X^man(X) --> [man].
n:X^dog(X) --> [dog].
v:Y^X^E^saw(E,X,Y) --> [saw].
p:X^Y^with(X,Y) --> [with].
```

This grammar, together with the given situation theory, allows the correct analysis and this analysis *only*. The rule expanding the *vp* into a *vp* adjoined with a *pp* is not applicable since the rule expanding the *pp:e2* is not applicable. The *pp* rule is not applicable since *with(e2,x5)* is not provable.

Compared to the statements in (284), the antecedents of the statements in (286) are logically strengthened, filtering away some false instances of rules. There is no need for ambiguity resolution, since ambiguity never arises.[9]

So with a large set of true (or truthlike) rules like these, would we not then be able to do semantic interpretation with a 100% accuracy? How could we fail, with all that knowledge available? But this is not the correct conclusion to draw. Interpretation would still be subject to failure, since we are not using the grammar deductively, but *abductively*. In an abductive framework, no matter how accurate our information is, we cannot escape the problem of finding the *best* explanation among several possible ones. This is what Hobbs et al. use weights for in their approach to semantic interpretation based on *weighted abduction* (see Stickel, 1990; Hobbs et al., 1993).

---

9. But given a grammar written in this style, and given the task of searching a text for (say) instances of noun phrases, would we then not be *forced* to model all of the text's content, as well as the contextual knowledge needed to interpret the text, before we could start searching it? – a formidable undertaking if the corpus is large! No, not necessarily. If searching is implemented, not as *deduction*, but rather as *abduction*, and if the contextual conditions of grammar rules have been declared as assumable, they can be *assumed*, rather than proved. In that case, the *result* of searching would always be a pair consisting of a segment and a (possibly empty) set of assumptions: "*this* is one of the segments you are looking for, assuming that you can accept *the following*". If the user is willing to make the assumptions, he must also agree that the segment found is right on target, or else, his conclusion must be that the rule is false.

14.4.2 A non-monotonic approach to co-text sensitivity

It is important to realize that the *addition* of a rule or two to a false grammar will not make it more truthlike, since the logic we use is monotonic: new axioms can never invalidate previous theorems. If the rule is false now, it will continue to be false, no matter how many other rules are added. This leads directly to the idea of using a *non-monotonic* grammar, where the addition of an axiom *may* block some previous theorems.

In this section, I sketch an approach that involves making grammar rule statements more sensitive to co-text, by adding, to the antecedents of them, one very general and 'non-monotonic' condition that makes the applicability of the rule depend on local co-text. In Chapter 11, it was proposed that something along these lines could be useful for automatic tagging purposes.

For example, instead of the grammar rule in (272), we add the following statement to the current grammar theory:

(287) syncat(S,np) :- syncat(S,n(pl)), consistent(syncat(S,np)).

This statement can be read as follows: 'an instance of a plural noun is an instance of a noun phrase if this is consistent with the rest of the current theory', or 'every instance of a plural noun is an instance of a noun phrase if no exceptions apply'.

Exceptions can be formulated as sentences expressing negative information. For example, an exception excluding noun phrases immediately following adjectives can be written as in (288) and added to the current grammar.

(288) ¬ syncat(P1-P2,np) :- syncat(P0-P1,adj).

The important thing is that exceptions override general rules. *Before* we added the exception in (288) to the current theory, it followed, from the sentence in (287), and (referring to example (273) again) from *syncat(5-6,n(pl))*, that *syncat(5-6,np)* held. Now, when (288) is added, it so happens that *syncat(5-6,np)* is explicitly negated, and thus that (287) is temporarily overridden.

This is a non-monotonic process; by adding an axiom, we invalidated a previous theorem which is exactly what we wanted in this case, since the blocked statement happened to be false.

Now, let's give a final twist to this proposal. We could, and I think we should, regard the addition of that extra "very general and 'non-monotonic' condition" as a change in the *meaning* of grammar rules, rather than as a proposal for a way to change particular grammars by piecemeal modification of their rules. In other words, this is a proposal in the spirit of the second of the interpretation options discussed in Section 14.2.1. It thus becomes the task of the translator from grammar rules to universal statements (i.e. the 'rule compiler') to add this condition to the antecedent of each universal statement corresponding to a grammar rule, as follows:

Rules of the form given in (289) are translated into clauses of the form in (290).[10]

(289) $C \text{ --> } C_1,...,C_n.$

(290) $\text{syncat}(P_0\text{-}P_n,C) \text{ :- } \text{syncat}(P_0\text{-}P_1,C_1),..., \text{syncat}(P_{n-1}\text{-}P_n,C_n),$
$\text{consistent}(\text{syncat}(P_0\text{-}P_n,C)).$

Lexical rules, i.e. rules of the form given in (291), are translated into clauses of the form in (292).

(291) C --> [W].

(292) syncat(S,C) :- bstring(S,W), consistent(syncat(S,C)).

Exceptions can be given the form in (293), meaning that instances of the syntactical category C_1 is never followed by instances of C_2. An exception written on this form is expanded to the two clauses in (294).

(293) $\neg (C_1,C_2)$

(294) $\neg \text{ syncat}(P1\text{-}P2,C_2) \text{ :- } \text{syncat}(P0\text{-}P1,C_1).$
$\neg \text{ syncat}(P0\text{-}P1,C_1) \text{ :- } \text{syncat}(P1\text{-}P2,C_2).$

Perhaps "Definite Clause Default Grammar" (DCDG) would be a suitable name for this new grammar formalism; or maybe "Definite Clause Constraint Grammar" (DCCG), to suggest a link with the Constraint Grammar of Karlsson et al. Either way, this could be a way to increase the truthlikeness of grammars: to fiddle with the interpretation of the rules,

---

10. Kowalski and Sadri (1990) – who have inspired the approach here although they do not deal specifically with grammars – give a method by which the result of performing the transformation proposed above can be further transformed into an efficient conventional Prolog program with negation as failure, but without explicit negation. Their transformation involves the renaming operation presented in (Gelfond & Lifschitz, 1990) (cf. Footnote 2 on page 175).

rather than to modify the rules as such. My use of the term “fiddle” here is meant to reflect something of my own ambivalence: perhaps we should, and then again, perhaps not.

14.5 Summary and Conclusion

In traditional grammars, under the interpretation of grammar rules favoured in this thesis, false rules are surprisingly common. They make tools that depend on them work less than optimally. Ambiguity, imprecision, and other errors result. An ‘incremental’ change of a false grammar into a grammar somewhat ‘less false’ may simply involve either logically strengthening the antecedent of one of its grammar rules, or logically weakening its consequent. A ‘revolutionary’ change in a grammar may involve a radical restructuring of information, perhaps even assuming the existence of novel ontological categories. One way to ‘fix’ an erroneous grammar rule is to make it more sensitive to context (e.g. by adding conditions in the antecedent that refer to items in the context). An interesting step towards a general solution is total integration of syntax, semantics, and pragmatics, in the sense of Hobbs et al. (1993). Another approach involves non-monotonic reasoning about local co-text. I see no reason why these approaches should not be possible to combine.

CHAPTER 15

Conclusion

> My conclusion is that the two kinds of linguists need each other. Or better, that the two kinds of linguists, wherever possible, should exist in the same body.
>
> *Charles J. Fillmore*

15.1 Meet This Linguist!

Imagine a linguist who is both a corpus linguist and a theoretical linguist – two kinds of linguists existing in one body. He thinks hard about problems of syntax, semantics, and pragmatics, and he is well versed in traditional theoretical linguistics, but at same time, he is very concerned about corpus data.

This linguist has *intuitions*, and he is happy to acknowledge their role in the process of discovering new things about text, and in the process of generating hypotheses. But he would not dream of seeking justification for his theories in his intuitions. After all, there is *so* much text out there, so why not use it? And use it he does – this linguist spends a lot of time buried in corpus data.

His other focus is on *theories*. This linguist thinks that science is the business of making theories, and that the science of corpus linguistics ought to be making theories about text. Moreover, since he believes in truth, his theories will have to be *true* theories about text. He prefers *explicitly* formulated theories, because he finds it hard otherwise to determine what is true therein and what is false, or to determine their logical consequences. In fact, he is most happy with *formal* theories.

Approaching a new text, he brings along with him a theory – based on what he has read, on what he has been told, on his intuitions, and perhaps on knowledge of his previous studies – and his goal is to *revise* that theory to fit the facts of the text. Thus to the next text, he might be able to bring a better theory. Descriptive general theories that cover as much as possible of the data – and truthfully describe, predict, and explain as much as possible – that's what he is after.

This linguist is *strongly* aware of the fact that context plays an extremely important role in text, and that it would be a grave mistake to try to treat the elements of texts as context-free objects. To him, text is not merely text. Texts relates to other situations – to parts of a real or fictional world described or discussed or presupposed by the texts. This linguist is open for some radical suggestions for how that should be described.

This linguist loves tools. Or rather, he loves what tools can do for him. But he wants to move beyond the use of word processors and simple tools for searching and counting, since he has understood that present-day computers and software technology are capable of much more. He is a highly motivated user of computerized research instruments, but he is very demanding, and he can become very frustrated when the functionality offered does not get him where he wants to go. His needs are unpredictable; he does not always know what he wants.

This linguist thinks he is important. He believes that successful corpus analysis depends on a division of labour between the corpus system and the human mind. He's not someone to click a button, walk away, and come back to collect the result. He wants to be *part* of the process of developing corpus theories; he wants to be able to bring his capacity for observation, understanding, and creativity into the process.

Through dialogue with his tools, but with his mind on the texts and the theories, he wants to forget about his tools; he wants them to vanish as he becomes completely absorbed in his work.

15.2 What Does He Need?

I have argued, in this thesis, that a corpus linguist would benefit most from a corpus theory development environment in which corpus theories, as well as texts, can be represented explicitly – an environment in which corpus theories can, as it were, be *shared* by the user and the system – understood by the system, and the user, improved by the system, or the user.

Such an environment should allow the linguist to formulate and test descriptive generalizations which are (at least broadly) compatible with his current favourites among theoretical linguistics ideas and principles. It should allow him to develop descriptive generalizations on the basis of specific descriptions of data, and to apply descriptive generalizations in order to produce such descriptions of other similar data.

To support this linguist's work style, and to satisfy his varying needs, this environment should be multi-functional, general, and very flexible. Simple things should be simple to do, and complex tasks should be possible.

Buried in corpus data, this linguist needs support for navigation. He has maps – because theories are like maps – but he needs a co-driver: a map reader and a guide to the texts that he is working with. However, a theory – if it is general – is not *only* a map. Not only does it describe, it has explanatory power as well.[1] This then happens to coincide with another need of the linguist: the need to follow the arguments presented by the

1. As Popper (1992, p. 77) puts it: "there exist languages, such as maps, which are descriptive but not argumentative. (This, incidentally, makes the familiar analogy between maps and scientific theories a particularly unfortunate one. Theories are essentially argumentative systems of statements: their main point is that they explain deductively. Maps are non-argumentative. Of course every theory is also descriptive, like a map – just as it is, like all descriptive language, communicative, since it may make people act; and also expressive, since it is a symptom of the 'state' of the communicator – which may happen to be a computer.)".

system. He wants to know why the system said what it said, or did what it did. This linguist does not like to be left in the dark concerning actions and conclusions of his tools. He needs therefore a tool that is able to exploit the explanatory power of corpus theories in order to explain its actions, to present the rationale behind its conclusions, and so on.

It is for this linguist, and for other linguists like him, that I have designed the TagLog system.

15.3 How Do We Give It to Him?

I have argued, in this thesis, that if we want to give this linguist the corpus theory development environment that he needs and wants, then formal logic, and in particular *computational* logic, is a key technology: the use of sentences of logic to express statements about texts and the use of inference to manipulate these sentences in order to analyse the texts.

By using a logical language instead of a mark-up language – ground, atomic sentences instead of labels – we are able to make greater sense of the practice of coding and tagging; we are able to give this practice a ‘semantics’.

The logic I propose to use is of the first order – we could think of this as the addition of variables, quantifiers, and logical operators to a mark-up language. The result is something that is much more powerful than what is actually needed for specific descriptions of particular facts of texts. I have proposed to use this extra power to allow the linguist to formulate general theories – definitions and descriptive generalizations about text.

The expressive power of the TagLog formalism has also been harnessed as a ‘corpus query language’. Partial descriptions can be formulated, and reasoning engines – deducers, abducers, inducers – can be used to fill them in, and thus retrieve information about corpus texts. Thus, the ‘description language’ – for describing particular facts, and for formulating descriptive generalizations – has been supplemented with a matching ‘query language’ – for querying the corpus described.

By way of numerous examples I have shown how various ‘logical reconstructions’ of functionalities needed by the corpus linguist can be imple-

mented: searching as deduction, parsing as deduction, interpretation as abduction, counting as deduction, lemmatization as abduction, automatic part-of-speech tagging as non-monotonic reasoning, and so on.

15.4 What Does He Get?

The basis of computational logic together with principles of human-computer interaction have been shown to lead to a system which scores high in terms of:

- *Expressive power.* The availability of logical variables and universal quantifiers guarantees a level of expressiveness sufficient for a natural formulation of systems of definitions and descriptive generalizations, and the availability of logical connectives and 'higher-order' predicates involving sets and their properties, allows the user to formulate complex queries and specifications.
- *Explicitness of underlying reasoning*. An explicit proof theory allows the system to deduce, in a transparent way, the answers to queries posed by the user, in the form of sentences that follow as logical consequences from the theory describing the corpus.
- *Tractability.* The efficient search space traversal of the SLD resolution strategy allows these logical consequences to be computed in reasonable time and space. Where things tend to take long time, short-circuiting is probably possible.
- *Multi-functionality.* The chosen formalism – powerful but still tractable – allows a 'Swiss-Army knife' approach to corpus tools, placing at the disposal of corpus linguists a battery of functionality: hand coding, searching, concordancing, parsing, counting, tabling, collocating, automatic part-of-speech tagging, lemmatizing, excerpting, interpreting, tree banking, hypothesis testing, explanation, and various kinds of learning.
- *Generality.* The system generalizes the classical tools: segments of any length can be handled, descriptions on different levels of analysis are allowed, overlap can be handled, not only properties of segments, but also relations between segments may be described. Everything we can describe, we can also query.

- *Declarativity.* The system enforces the view of corpus theories as declarative bodies of knowledge, 'detached' from the tools that use them. Everything is explicit, nothing is hidden. In the course of analysis, the user only has to state *what* (information) he wants to get as the result of a computation, not *how* he thinks the system should compute it. The user does not have to be a programmer.
- *Relevancy.* Corpus theories relevant to current theories of syntax, semantics and pragmatics can be developed in TagLog. Since the formalism is rule-based and uses unification as one of its basic operations, the kind of grammars advocated by modern theories of syntax can be written – or at least emulated to a high degree. Compositional logical semantics can be integrated with the syntax – indeed another current trend in modern theories of grammar. Finally, since many would say that the key to the solution of many problems in the area of pragmatics lies in the notion of interpretation in context, and in the integration of knowledge of syntax and semantics with knowledge of the world, it should be noted that such an integrated picture is possible to build in TagLog, in a straightforward and principled way.
- *Flexibility.* The corpus linguist is able to do very simple things in TagLog – the mark-up-and-count game that corpus linguists have played for decades, for example, is very easy to play in TagLog too. The corpus linguist is able to do very complex things in TagLog: explanation, automatic hypothesis testing, text-context modelling, to mention some. Moreover, TagLog allows the simple approaches, and the very sophisticated ones, to be implemented in *the very same framework*, and allows the former to be extended into the latter, in a smooth and natural way.
- *Incrementality, interactiveness*, and *balanced cooperativeness*. Sentences of theories are represented such that the user, as well as the system, can easily input, modify, and inspect them. Thus corpus theories can be built in an incremental fashion, in interaction and balanced cooperation with the user.
- *Connectedness*. The tools all speak the same language. This common representational basis of TagLog has made it possible to dynamically link its many functionalities, so that the corresponding tools work, not as a bunch of tools thrown into a bag, but as an integrated connected whole.

- *Consistency.* The use of one single formalism based on logic and set theory, for descriptions as well as for queries, independent of what subdomain and what tool we are working with, lends consistency to the whole system.
- *Simplicity.* The TagLog system is getting rather big. Are we running the risk that TagLog will eventually begin to suffer from too many features? I do not think so, and I agree with Nelson (1990, p. 236) when he claims that:

  > Featuritis doesn't mean that a program aspires to do too much; there need be no limit on how many distinguishable things a program can do. The disease of featuritis is the unclarity and confusion that results from *having too many separate, unrelated things to know and understand.* The relationship between *power and flexibility for the user* and *simplicity at the interface* need not be inverse. Ultimately, the best software design assimilates all its functions to a few clear and simple principles.

- *Usability.* The usability of the TagLog system is something that emerges from the combination of the above points, together with the conceptual model based on notions from linguistics and philosophy of science, and the conversational view on human-computer interaction.

15.5 What More Could He Possibly Ask For?

I think this thesis only scratches the surface of the possibilities inherent in the logical approach to computational corpus linguistics. There is room for improvements and variations, and there is room for extensions.

15.5.1 More testing

The development of corpus tools and methods, just like the development of theories, is a trial-and-error process. While I think I have created something which appears feasible on paper, and something which seems to be a promising prototype, I have yet to evaluate the TagLog system on a larger scale, in a major corpus project. In other words, if 'the proof of the pudding is in the eating', I have still to prove my pudding.

Another task high on the agenda is a round of thorough and systematic *user testing*. The development of interactive software always needs to

pass through a couple of phases of user testing. Now, TagLog has users; they have given valuable feedback; I have tried to respond to their criticism, and that has undoubtedly improved the system a lot. But I have still to perform user testing in a thorough and systematic way.

15.5.2 Better formalism

One possible variation of the present system is to replace the current TagLog formalism – a subset of the Prolog programming language plus some special purpose built-in predicates – with a formalism designed specifically for the purpose of describing texts and querying the resulting descriptions. We could imagine using many-sorted logic, higher-order logic, description logic, or whatever, and indeed, we could imagine a formalism allowing the use of logical disjunction in descriptions of facts. However, we would have to balance the advantages of using such a language against the advantages of using an existing language which is fairly well-known and which is slowly becoming the *lingua franca* of computational linguistics.

I hope I have not left the reader with the impression that I oppose the use of mark-up languages. I do indeed oppose the use of ‘formal’ languages for which important linguistic properties are unclear – nothing good can come of that. But I have shown, I hope, that a marked-up text may be ‘logically reconstructed’ and that what we get then is, in fact, a ‘semantics’ for the mark-up language.

We might want to view a marked-up text as a notational variant of a set of ground atomic clauses, similar to how a DCG is just another way of writing a set of Horn clauses. That is, we might want to regard the *whole* of

(295)

bstring:	John	loves	Mary
lexcat:	pn	v	pn

as a notational variant of

(296)

```
bstring(1-2,”John”).   bstring(2-3,”loves”).   bstring(3-4,”Mary”).
lexcat(1-2,pn).        lexcat(2-3,vp).         lexcat(3-4,pn).
```

which is to be regarded as a description of

(297) John loves Mary

It seems to me that this might be a viable approach to the semantics of mark-up languages.

15.5.3 More speed

Speed is likely to become less of an issue as computers are getting more powerful, memory is getting cheaper, and parallelism makes it into mainstream computing. But we still might want to try to apply some of the optimization techniques mentioned in Chapter 5. In certain cases, to be automatically detected by the system, we might also want the system to short-circuit general reasoning, and revert to more traditional kinds of processing. For example, in cases where the system can detect that a grammar (of a certain kind) is going to be used for parsing, it could invoke a conventional and highly optimized parser, instead of the general theorem prover. As long as the user does not notice the difference, except that things are getting faster, this is perfectly in line with the TagLog approach. That is, parsing *as if* by deduction is quite acceptable.

15.5.4 More functionality

Machine learning techniques have already been exploited in the current version of the TagLog system: explanation-based learning in the Explanation Tool, error-driven learning of condition-action rules, and abduction for automatic lemmatization, grammar development, and world knowledge acquisition. We could imagine extending TagLog further in that direction. Machine learning is a vast area of research but I think that the nature of TagLog is such that we could restrict our attention to approaches that are formulated within a logical framework,[2] and to interactive theory revision approaches.

A tool supporting restoring of consistency and correction of errors could be designed. Such a tool could be based on the work of Shapiro (1983). Given a prediction known to be erroneous, a proof tree is generated that explains why the error is implied by the current theory. The tool then interacts with the user (the user takes the role of an ‘oracle’) and asks, about each node in the proof tree, if it is true about the segment that it purports to describe, or not (the segment could be pointed to by the system). The idea is to try to find a false node whose daughter nodes are all true, since that determines a false instance indicating that the general form of the clause is false. The incorrect clause is then removed or modi-

---

2. In particular to the area of inductive logic programming (see e.g. Muggleton, 1992).

fied (or a decision is taken to keep it anyway). Shapiro (ibid.) also develops a technique (similar to abduction) for diagnosing the coverage of a set of clauses, for finding ‘missing’ facts that should be covered, but are not.

It seems to me that a template-based approach to learning, similar to Brill’s method, would be an interesting direction in which to move. The user supplies a set of templates, and perhaps some further restrictions, and the system considers all possible instances of the templates that conform to the restrictions. As an example of such a system, let me mention RDT (Kietz & Wrobel, 1992), a part of Mobal (Morik et al., ibid.).

I have already indicated some possible applications of machine learning techniques: the learning of negative constraints needed for the logic-based approach to automatic part-of-speech tagging is one such application. It would also be extremely interesting to see how far one could go towards learning integrated grammars à la Hobbs et al. on the basis of manually coded corpora and context/content models constructed by hand.

15.5.5 More clarity

There are different kinds of clarity: logical clarity, philosophical clarity, methodological clarity, clarity with respect to aims and expectations. Clarity is always worth striving for. I hope this thesis has succeeded in achieving some clarity, and I hope I have managed to bring some of this clarity into the design of my system, and into its user interface. I am sure, however, that a lot remains to be done.

Logical clarity has been of top priority, and if I have failed here, I have failed to reach what I was aiming for. There is more to the logic of corpus linguistics than mere formalisms – there is action – and there is more to a formalism than mere notation – there is meaning. Notation, meaning, action – these are the aspects of corpus linguistics where I think I have reached some clarity.

Philosophical clarity does not mean knowing how things ‘really’ are – that may not be possible – but rather making assumptions explicit that would otherwise remain hidden. In this work, I have taken stands where philosophers would have been more cautious – the fact that I am working on an application has forced me to. Philosophical stands, however, need to be considered again and again. I have opted for a fact-based ontology, but there may be alternatives that are also worth considering. As regards

truthlikeness, there is probably more to such a notion than I have been able to reveal. Furthermore, for those who think one has to be a saint to believe that truth is important, the instrumentalist alternative should perhaps be explored as well. (Although personally, I think that a theory is useful *because* it is true, and that to think of it as true because it is *useful* is to put the cart before the horse. "Useful for what?" is a question we can always ask.)

Moreover, the quest for truth is of a methodological importance that cannot be overestimated. It is this quest that sends us looking for ways to modify a theory to better fit the facts. If the theory says *A*, but we see that clearly *not A*, then the quest for truth will send us tracing backwards, via *A :- B*, and further backwards via *B*, revealing hopefully an error, perhaps suggesting a fix. Logic shows us where to look, but it is the quest for truth that makes us want to search in the first place.

Finally, as regards goals and expectations, I think the linguist has to be very clear about the balance between his role as a scientist and his role as a technologist. Unclarity with respect to this balance could be dangerous for *computational* linguists in particular, since, as Kay (1992, p. 5) has noted, "just about every computational linguist lives a double life as a linguistic technologist, that is, as at least an aspiring architect of well crafted, elegant, and efficient devices that will do useful things with language." Yet, to think too much about tools and usefulness, and too little about theories and truth, may not be what pays off in the end. As someone has said, "there's nothing as practical as a good theory."

References

Ahrenberg, L. & Jönsson, A. (1989) An Interactive System for Tagging Dialogues. *Literary and Linguistic Computing* 3(2), 66-70.

Allwood, J. (1976) *Linguistic Communication as Action and Cooperation*. Doctoral Dissertation. University of Göteborg: Department of Linguistics.

Allwood, J., Andersson, L.-G. & Dahl, Ö. (1977) *Logic in Linguistics*. Cambridge University Press.

Beun, R.-J. (1989) *The Recognition of Declarative Questions in Information Dialogues*. Doctoral Dissertation. University of Tilburg.

Botha, R. P. (1989) *The Generative Garden Game: Challenging Chomsky at Conceptual Combat*. Basil Blackwell, USA.

Bowen, K. A. & Kowalski, R. A. (1982) Amalgamating Language and Metalanguage in Logic Programming. In K. L. Clark & S.-Å. Tärnlund (Eds.). *Logic Programming*. Academic Press.

Brennan, S. E. (1990) Conversation as Direct Manipulation: An Iconoclastic View. In B. Laurel (Ed.). *The Art of Human-Computer Interface Design*. Addison-Wesley.

Brill, E. (1992) A Simple Rule-Based Part of Speech Tagger. *Papers presented at the 3rd Conference of Applied NLP*, Trento, Italy.

Brown, P. F., Pietra, V. D., DeSouza, P. & Lai, J. C. (1990) Class-Based n-gram Models of Natural Language. *Computational Linguistics* 18(4), 467-479.

Bunge, M. (1995) Quality, Quantity, Pseudoquantity and Measurement in Social Science. *Journal of Quantitative Linguistics* 2(1), 1-10.

Calder, J. & Humphreys, K. (1994) *Pleuk Overview.* (Manual) University of Edinburgh: Human Communication Research Centre.

Carr, P. (1990) *Linguistic Realities: An Autonomist Metatheory for the Generative Enterprise*. Cambridge University Press.

Chafe, W. (1992) The Importance of Corpus Linguistics to Understanding the Nature of Language. In J. Svartvik (Ed.). *Directions in Corpus Linguistics: Proceedings of Nobel Symbosium 82, Stockholm, 4-8 August 1991*. Berlin: Mouton De Gruyter.

Charniak, E. (1993) *Statistical Language Learning*. Cambridge, Mass.: MIT Press.

Chomsky, N. (1957) *Syntactic Structures*. The Hague: Mouton.

Christ, O. (1994) A Modular and Flexible Architecture for an Integrated Corpus Query System. *Paper presented at the 3rd Conference on Computational Lexicography and Text Research*, Budapest, Hungary.

Church, K. W. (1988) A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. *Paper presented at the 2nd Conference on Applied Natural Language Processing*, ACL.

Clocksin, W. F. & Mellish, C. S. (1987) *Programming in Prolog*. Springer-Verlag.

Coulthard, M. (1977) *An Introduction to Discourse Analysis*. London: Longman.

Dahl, V., Huang, Y. N., & Tarau, P. (1994) Datalog Grammars. In *Proceedings of GULP-PRODE 1994*, Valencia.

Davies, J. (1991) Automated Tools for Qualitative Research. In N. G. Fielding & M. G. Lee (Eds.). *Using Computers in Qualitative Research*. SAGE Publications.

DeRaedt, L. (1992) *Interactive Theory Revision: An Inductive Logic Programming Approach*. Academic Press.

DeRose, S. J. (1988) Grammatical Category Disambiguation by Statistical Optimization. *Computational Linguistics* 14(1), 31-39.

Dyvik, H. (1992) *To forelesninger om lingvistikkens vitenskapsteori*. (41.) Institutt for fonetikk og lingvistikk, Universitetet i Bergen.

Fillmore, C. J. (1992) "Corpus linguistics" or "Computer-aided armchair linguistics". In J. Svartvik (Ed.). *Directions in Corpus Linguistics: Proceedings of Nobel Symposium 82, Stockholm, 4-8 August 1991*. Berlin, New York: Mouton De Gryer.

Francis, W. & Kučera, H. (1982) *Frequency Analysis of English Usage*. Houghton Mifflin.

Gazdar, G. & Mellish, C. (1989) *Natural Language Processing in Prolog: An Introduction to Computational Linguistics*. Addison-Wesley.

Gelfond, M. & Lifschitz, V. (1990) Logic Programming with Classical Negation. *Paper presented at the Seventh International Conference on Logic Programming*, Jerusalem.

Gilbert, N., Wooffitt, R. & Fraser, N. (1990) Organising Computer Talk. In P. Luff, N. Gilbert, & D. Frohlich (Eds.). *Computers and Conversation*. Academic Press.

Grice, H. P. (1975) Logic and Conversation (From the William James lectures, Harvard University, 1967). In P. Cole & J. Morgan (Eds.). *Syntax and Semantics 3: Speech Acts*. New York: Academic Press.

Hempel, C. G. & Oppenheim, P. (1948) Studies in the Logic of Explanation. *Philosophy of Science* 15, 135-175.

Hill, P. M. & Lloyd, J. W. (1989) Analysis of Meta-Programs. In H. Abramson & M. H. Rogers (Eds.). *Meta-Programming in Logic Programming*. University of Bristol: MIT Press.

Hill, P. M. & Lloyd., J.W. (1991) *The Gödel Report*. (TR-91-02.) Department of Computer Science, University of Bristol.

Hindle, D. (1983) *User Manual for Fidditch, a Deterministic Parser.* (Technical Memorandum 7590-142.) Naval Resarch Laboratory.

Hobbs, J. R., Stickel, M. E., Appelt, D. E. & Martin, P. (1993) Interpretation as Abduction. *Artificial Intelligence* 63, 69-142.

Hogger, C. J. (1990) *Essentials of Logic Programming*. Oxford University Press.

Hopcroft, J. E. & Ullman, J. D. (1979) *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

ISO/IEC. (1995) ISO/IEC 13211-1:1995 Information technology -- Programming languages -- Prolog -- Part 1: General core. .

Itkonen, E. (1978) *Grammatical Theory and Metascience*. Amsterdam: Benjamin.

Johansson, S. (1992) Comments [on a paper by Staffan Hellberg]. In J. Svartvik (Ed.). *Directions in Corpus Linguistics: Proceedings of Nobel Symposium 82, Stockholm, 4-8 August 1991*. Berlin: Mouton de Gruyter.

Johnson, M. (1994) Two Ways of Formalizing Grammars. *Linguistics and Philosophy* 17(3).

Kakas, A. C., Kowalski, R. & Toni, F. (1993) Abductive Logic Programming. *Journal of Logic and Computation* 2(6), 719-770.

Kamp, H. (1985) Context, Thought and Communication. *Proceedings of the Aristotelian Society,* 239-261.

Karlsson, F. (1990) Constraint Grammar for Parsing Running Text. *Paper presented at the 13th International Conference on Computational Linguistics* (COLING), Helsingfors.

Karlsson, F., Voutilainen, A., Heikkilä, J. & Antilla, A. (1994) *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*. Mouton De Gruyter.

Kay, M. (1979) Functional Grammar. *Paper presented at the Fifth Annual Meeting of the Berkeley Linguistics Society*, Berkeley.

Kay, M. (1992) Unification. In M. Rosner & R. Johnson (Eds.). *Computational Linguistics and Formal Semantics*. Cambridge University Press.

Kietz, J.-U. & Wrobel, S. (1992) Controlling the Complexity of Learning in Logic through Syntactic and Task-Oriented Models. In S. Muggleton (Ed.). *Inductive Logic Programming*. Academic Press.

Kim, J. (1967) Explanation in Science. In P. Edwards (Ed.). *Encyclopedia of Philosophy* 3. New York: Macmillan Free Press.

Kós-Dienes, D. (forthcoming) *Communication between Doctor and Patient*, Doctoral Dissertation. University of Göteborg.

Kowalski, R. A. (1979) *Logic for Problem Solving*. North-Holland.

Kowalski, R. A. (1990) *Problems and Promises of Computational Logic*. In J. W. Lloyd (Ed.). *Computational Logic*. Symposium Proceedings, Brussels, November 1990. Springer-Verlag.

Kowalski, R. A. & Sadri, F. (1990) Logic Programs with Exceptions. In D.H.D Warren & P. Szeredi (Eds.). *Proceedings of the Seventh International Conference of Logic Programming*. MIT Press.

Lambek, J. (1958) The Mathematics of Sentence Structure. *American Matemathical Monthly* (65), 154-170.

Leech, G. (1991) The State of the Art of Corpus Linguistics. In K. Aijmer & B. Altenberg (Eds.). *English Corpus Linguistics: Studies in Honour of Jan Svartvik*. London and New York: Longman.

Leech, G. (1992) Corpora and Theories of Linguistic Performance. In J. Svartvik (Ed.). *Directions in Corpus Linguistics: Proceedings of*

Nobel Symposium 82, Stockholm, 4-8 August 1991. Berlin: Mouton de Gruyter.

Levesque, H. & Brachman, R. J. (1985) A Fundamental Tradeoff in Knowledge Representation and Reasoning. In R. J. Brachman & H. Levesque (Eds.). *Readings in Knowledge Representation*. Los Altos California: Morgan Kaufman Publishers, Inc.

Levinson, S. C. (1983) *Pragmatics*. Cambridge: Cambridge University Press.

Lloyd, J. (1987) *Foundations of Logic Programming*. Springer-Verlag.

Lyons, J. (1968) *Introduction to Theoretical Linguistics*. London and New York: Cambridge University Press.

Lyons, J. (1977) *Semantics*. London and New York: Cambridge University Press.

MacWhinney, B. (1990) *Computational Tools for Language Analysis: The CHILDES System*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K. & Schasberger, B. (1994) The Penn Treebank: Annotating Predicate Argument Structure. In *Proceedings of the Human Language Technology Workshop*. San Francisco, CA.: Morgan Kaufmann Publishers Inc.

Marcus, M. P., Santorini, B. & Marcinkiewicz, M. A. (1993) Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics* 19(2), 313-330.

McGee Wood, M. (1993) *Categorial Grammars*. London: Routledge.

McNaught, J. (1993) User Needs for Textual Corpora in Natural Language Processing. *Literary and Linguistic Computing* 8(4), 227-234.

Mellish, C. S. (1988) Implementing Systemic Classification by Unification. *Computational Linguistics* 14(1), 40-51.

Meyer, C. F. & Tenney, R. L. (1993) Tagger: An Interactive Tagging Program. In C. Souter (Ed.). *Corpus-Based Computational Linguistics*. Amsterdam: Rodopi.

Mitchell, T. M., Keller, R. M. & Kedar-Cabelli, S. T. (1986) Explanation-Based Generalization: A Unifying View, *Machine Learning* 1.

Morik, K., Wrobel, S., Kietz, J. U. & Emde W. (1993) *Knowledge Acquisition and Machine Learning: Theory, Methods and Applications*. London: Academic Press.

Morik, K. (1994) Balanced Cooperative Modeling. In R. Michalski & G. Tecuci (Eds.). *Machine Learning: A Multistrategy Approach*. San Francisco, California: Morgan Kaufmann Publishers.

Muggleton, S. (Ed.). (1992) *Inductive Logic Programming*, Academic Press.

Nelson, T. H. (1990) The Right Way to Think About Software Design. In B. Laurel (Ed.). *The Art of Human-Computer Interface Design*. Addison-Wesley.

Nivre, J. e. (1992) Corpus Collection and Analysis. (Deliverable D2.1.) ESPRIT Project P5254 PLUS.

Partee, B., ter Meulen, A. & Wall, R. E. (1990) *Mathematical Methods in Linguistics*. Dordrecht: Kluwer Academic.

Pereira, F. C. N. & Shieber, S. M. (1987) *Prolog and Natural-Language Analysis*. Stanford: Chicago University Press.

Pereira, F. C. N. & Warren, D. H. D. (1980) Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks. *ArtificiaI Intelligence* 13(3), 231-278.

Pereira, F. C. N. & Warren, D. H. D. (1983) Parsing as Deduction. *Paper presented at the 21st Annual Meeting of ACL*.

Popper, K. R. (1972) *Objective Knowledge: An Evolutionary Approach.* Oxford: Clarendon Press.

Popper, K. R. (1980) *The Logic of Scientific Discovery*, Tenth impression. London: Hutchinson.

Popper, K. R. (1992) *Unended Quest: An Intellectual Autobiography.* London: Routledge.

Rayner, M. (1993) *Abductive Equivalential Translation and its application to Natural Language Database Interfacing*. Doctoral Dissertation. Stockholm University and Royal Institute of Technology: Department of Computer and Systems Sciences.

Reiter, R. (1978) On Closed World Databases. In H. Gallaire & J. Minker (Eds.). *Logic and Databases*. NY: Plenum Press.

Reiter, R. (1980) A Logic for Default Reasoning. *Artificial Intelligence* 13(1), 81-132.

Robinson, J. A. (1965) A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the Association for Computing Machinery* 12(1), 23-41.

Roche, E. & Schabes, Y. (1995) Deterministic Part-of-Speech Tagging with Finite-State Transducers. *Computational Linguistics* 21(2), 227-253.

Salton, J. (1989) *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley.

Samuelsson, C. & Rayner, M. (1991) Quantitative Evaluation of Explanation-Based Learning as an Optimization Tool for a Large-Scale Natural Language System. *Paper presented at the 12th Int. Joint Conference on Artificial Intelligence*.

Santorini, B. (undated) *GNU-Emacs Tools for the Penn Treebank Project*. University of Pennsylvania.

Shapiro, E. (1983) *Algorithmic Program Debugging*. MIT Press.

Shieber, S. M. (1986) *An Introduction to Unification-based Approaches to Grammar.* Chicago, Illinois: University of Chicago Press.

Shneiderman, B. (1992) *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley.

Sinclair, J. M. (1992) The Automatic Analysis of Corpora. In J. Svartvik (Ed.). *Directions in Corpus Linguistics: Proceedings of Nobel Symposium 82, Stockholm, 4-8 August 1991*. Berlin: Mouton de Gruyter.

Smadja, F. (1993) Retrieving Collocations from Text: Xtract. *Computational Linguistics* 19(1), 143-177.

Sterling, L. & Shapiro, E. (1986) *The Art of Prolog: Advanced Programming Techniques*. Cambridge, Mass.: MIT Press.

Stickel, M. E. (1990) Rationale and Methods for Abductive Reasoning in Natural Language Interpretation. In R. Studer (Ed.). *Natural Language and Logic*. Hamburg: Springer-Verlag.

van Harmelen, F. (1989) A Classification of Meta-level Architectures. In H. Abramson & M. H. Rogers (Eds.). *Meta-Programming in Logic Programming*. University of Bristol: MIT Press.

Viterbi, A. J. (1967) Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Transactions on Information Theory* IT-13(2) , 260-269.

Voutilainen, A. (1993) NPtool, a Detector of English Noun Phrases. *Paper presented at the Workshop on very large corpora: academic and industrial perspectives*, Ohio State Univ.

Voutilainen, A., Heikkilä, J. & Antilla, A. (1992) *Constraint Grammar of English: A Performance-Oriented Introduction*. (Publication No. 21.) Department of Linguistics, University of Helsinki.

Westerståhl, D., Haglund, B. & Lager, T. (1993) A Situation-Theoretic Representation of Text Meaning: Anaphora, Quantification, and Negation. In P. Aczel, D. Israel, Y. Katagiri, & S. Peters (Eds.). *Situation Theory and its Application*, Volume 3. Oiso, Japan: Chicago University Press.

Yngve, V. (1986) *Linguistics as a Science*. Bloomington and Indianapolis: Indiana University Press.

Zomogyi, Z., Henderson, F. J. & Conway, T. C. (1994) The Implementation of Mercury: an Efficient Purely Declarative Logic Programming Language. *Paper presented at the ILPS '94 Postconference Workshop on Implementation Techniques for Logic Programming Languages*, Syracuse, New York.

APPENDIX A

TagLog Built-In Predicates

A.1 Introduction

This appendix describes the built-in predicates available in TagLog. These predicates are provided in advance by the system and they cannot be redefined by the user. The following categories of built-in predicates are available:

1. Point and segment predicates
2. String predicates
3. Tree predicates
4. Metalevel predicates
5. Arithmetic
6. Schema predicates
7. Presentation procedures

8. Corpus and corpus theory management procedures

The sections to follow will list the built-in predicates accordingly. Each entry in these lists specificies four kinds of information:

- The *name* and *arity* of the predicate.
- The *type* and *mode* of the arguments.

  The type of the argument is defined by one of the following symbols:

  atom
  boolean
  goal
  clause
  integer
  point
  segment
  set-expression
  statement
  string
  term
  expression
  list

  The mode of each argument defines whether or not an argument shall be instantiated when the built-in predicate is executed. The mode is one of the following:

  \+ the argument shall be instantiated

  ? the argument shall be instantiated or a variable

  \- the argument shall be a variable that will be instantiated if the predicate succeeds.
- A description of the built-in predicate which gives the conditions under which the predicate is true, and/or describes the procedural behaviour of the predicate.
- Examples where the built-in predicate is used as a goal, together with a statement saying whether the goal succeeds, fails or produces an error.

A.2 Point and Segment Predicates

point/1

point(?point)

point(Point) is true iff Point is a point in the current text.

Ex.1: point(5) succeeds if 5 is a point in the current text.
Ex.2: point(P) succeeds with P bound to a point in the current text.

segment/1

segment(?segment)

segment(Segment) is true iff Segment is a segment in the current text.

Ex.1: segment(3-5) succeeds if 3-5 is a segment in the current text.
Ex.2: segment(S) succeeds with S bound to segment in the current text.
Ex.3: segment(3-P) succeeds with P bound to a point >= 3 in the current text.
Ex.4: segment(P-5) succeeds with P bound to a point =< 5 in the current text.

bsegment/1

bsegment(?segment)

bsegment(Segment) is true iff Segment is a basic segment in the current text, i.e. a segment of length 1.

Ex.1: bsegment(3-4) succeeds.
Ex.2: bsegment(S) succeeds with S bound to a segment of length 1.
Ex.3: bsegment(3-5) fails, since 3-5 is not a basic segment.

segment_length/2

segment_length(?segment,?integer)

segment_length(Segment,Length) is true iff Length is the length of Segment.

Ex.1: segment_length(3-5,N) succeeds with N bound to 2.
Ex.2: segment_length(S,2) succeeds with S bound to a segment of length 2.

segment_maxlength/2

segment_maxlength(?segment,+integer)

segment_maxlength(Segment,MaxLength) is true iff the length of Segment is less or equivalent to MaxLength.

Ex.1: segment_maxlength(2-4,5) succeeds, since the length ot the segment 2-4 is less than 5.

Ex.2: segment_maxlength(S,5) succeeds with S bound to a segment the length of which is shorter than or equal to 5.

segment_prefix/2

segment_prefix(+segment,?segment)

segment_prefix(Segment1,Segment2) is true iff Segment2 is a prefix of Segment1.

Ex.1: segment_prefix(13-32,13-17) succeeds.

Ex.2: segment_prefix(13-32,13-13) succeeds.

Ex.3: segment_prefix(13-32,13-32) succeeds.

Ex.4: segment_prefix(13-32,S) succeeds with S bound to a prefix of 13-32.

Ex.5: segment_prefix(13-32,12-14) fails.

segment_pprefix/2

segment_pprefix(+segment,?segment)

segment_pprefix(Segment1,Segment2) is true iff Segment2 is a proper prefix of Segment1.

Ex.1: segment_pprefix(13-32,13-17) succeeds.

Ex.2: segment_pprefix(13-32,13-13) succeeds.

Ex.3: segment_pprefix(13-32,13-32) fails, since 13-32 is not a proper prefix of 13-32.

Ex.4: segment_pprefix(13-32,S) succeeds with S bound to a proper prefix of 13-32.

Ex.5: segment_pprefix(13-32,12-14) fails.

segment_suffix/2

segment_suffix(+segment,?segment)

segment_suffix(Segment1,Segment2) is true iff Segment2 is a suffix of Segment1.

Ex.1: segment_suffix(5-17,13-17) succeeds.

Ex.2: segment_suffix(5-17,17-17) succeeds.

Ex.3: segment_suffix(5-17,5-17) succeeds.

Ex.4: segment_suffix(5-17,S) succeeds with S bound to a suffix of 5-17.

Ex.5: segment_suffix(5-17,12-14) fails.

segment_psuffix/2

segment_psuffix(+segment,?segment)

segment_psuffix(Segment1,Segment2) is true iff Segment2 is a proper suffix of Segment1.

Ex.1: segment_psuffix(5-17,13-17) succeeds.

Ex.2: segment_psuffix(5-17,17-17) succeeds.

Ex.3: segment_psuffix(5-17,5-17) fails.

Ex.4: segment_psuffix(5-17,S) succeeds with S bound to a proper suffix of 5-17.

Ex.5: segment_psuffix(5-17,12-14) fails.

subsegment/2

subsegment(+segment,?segment)

subsegment(Segment1,Segment2) is true iff Segment2 is a subsegment of Segment1.

Ex.1: subsegment(15-171,34-65) succeeds.

Ex.2: subsegment(15-171,18-18) succeeds.

Ex.3: subsegment(15-171,15-171) succeeds.

Ex.4: subsegment(15-171,S) succeeds with S bound to a subsegment of 15-171.

Ex.5: subsegment(15-171,12-160) fails.

psubsegment/2

psubsegment(+segment,?segment)

psubsegment(Segment1,Segment2) is true iff Segment2 is a proper subsegment of Segment1.

Ex.1: psubsegment(15-171,34-65) succeeds.

Ex.2: psubsegment(15-171,18-18) succeeds.

Ex.3: psubsegment(15-171,15-171) fails.

Ex.4: psubsegment(15-171,S) succeeds with S bound to a subsegment of 15-171.

Ex.5: psubsegment(15-171,12-160) fails.

segment_concat/2

segment_concat(+segment,+segment,?segment)
segment_concat(?segment,?segment,+segment)

segment_concat(Segment1,Segment2,Segment3) is true iff Segment3 is the concatenation of Segment1 and Segment2.

Ex.1: segment_concat(23-34,34-67,23-67) succeeds.

Ex.2: segment_concat(23-34,34-67,S) succeeds with S bound to the segment 23-67.

Ex.3: segment_concat(23-34,S,23-67) succeeds with S bound to the segment 34-67.

Ex.4: segment_concat(S,34-67,23-67) succeeds with S bound to the segment 23-34.

segment_overlap/2

segment_overlap(+segment,+segment)

segment_overlap(Segment1,Segment2) is true iff Segment1 and Segment2 in the current text overlap.

Ex.1: segment_overlap(12-45,34-78) succeeds.

adjacent/2

adjacent(+segment,+segment)

adjacent(Segment1,Segment2) is true iff Segment2 immediately succeeds Segment1 in the current text.

Ex.1: adjacent(56-78,78-81) succeeds.

Ex.2: adjacent(56-78,S) succeeds with S bound to 78-P where P is a variable.

segment_precedes/2

segment_precedes(+segment,+segment)

segment_precedes(Segment1,Segment2) is true iff Segment1 precedes Segment2 in the current text.

Ex.1: segment_precedes(12-45,56-78) succeeds.

within_span/2

within_span(?segment,?segment,?integer)

within_span(Segment1,Segment2,Span) is true iff the length of the segment determined by the first point in Segment1 and the second point in Segment2 is less or equal to Span, and the first point of Segment1 is not preceded by the first point of Segment2.

Ex.1: within_span(54-56,57-58,5) succeeds.

Ex.2: within_span(54-55,S,5) succeeds with S bound to a subsegment of 54-59.

A.3 String Predicates

string/2

string(+segment,?string)
string(?segment,+string)

string(Segment,String) is true iff Segment is a segment in the current text and String is the string of which Segment is an instance.

Ex.1: string(45-48,"truth and beauty") succeeds if segment 45-48 of the current text is an instance of the string "truth and beauty".

bstring/2

bstring(?segment,?string)

bstring(Segment,String) is true iff Segment is a basic segment in the current text and String is the string of which Segment is an instance.

Ex.1: bstring(4-5,"truth") succeeds.

Ex.2: bstring(4-5,W) succeeds with W bound to the string "truth".

Ex.3: bstring(S,"truth") succeeds with S bound to the segment 4-5.

Ex.4: bstring(S,W) succeeds with S bound to the segment 4-5 and W bound to the string "truth".

string_length/2

string_length(+string,?integer)

string_length(String,Length) is true iff Length is the length (in number of characters) of String.

Ex.1: string_length("truth",5) succeeds
Ex.2: string_length("beauty",N) succeeds with N bound to 6.

string_prefix/2

string_prefix(+string,?string)

string_prefix(String1,String2) is true iff String2 is a prefix of String1.

Ex.1: string_prefix("truth","tr") succeeds.
Ex.2: string_prefix("truth","") succeeds.
Ex.3: string_prefix("truth","truth") succeeds.

string_pprefix/2

string_pprefix(+string,?string)

string_pprefix(String1,String2) is true iff String2 is a proper prefix of String1.

Ex.1: string_pprefix("truth","tr") succeeds.
Ex.2: string_pprefix("truth","") succeeds.
Ex.3: string_pprefix("truth","truth") fails.

string_suffix/2

string_suffix(+string,?string)

string_suffix(String1,String2) is true iff String2 is a suffix of String1.

Ex.1: string_suffix("truth","th") succeeds.
Ex.2: string_suffix("truth","") succeeds.
Ex.3: string_suffix("truth","truth") succeeds.

string_psuffix/2

string_psuffix(+string,?string)

string_psuffix(String1,String2) is true iff String2 is a proper suffix of String1.

Ex.1: string_psuffix("truth","th") succeeds.
Ex.2: string_psuffix("truth","") succeeds.
Ex.3: string_psuffix("truth","truth") fails.

substring/2

substring(+string,?string)

substring(String1,String2) is true iff String2 is a substring of String1.

Ex.1: substring("truth","rut") succeeds.
Ex.2: substring("truth","") succeeds.
Ex.3: substring("truth","truth") succeeds.

psubstring/2

psubstring(+string,?string)

psubstring(String1,String2) is true iff String2 is a proper substring of String1.

Ex.1: psubstring("truth","rut") succeeds.
Ex.2: psubstring("truth","") succeeds.
Ex.3: psubstring("truth","truth") fails.

string_concat/3

string_concat(+string,+string,?string)
string_concat(?string,?string,+string)

string_concat(String1,String2,String3) is true iff String3 is the concatenation of String1 and String2.

Ex.1: string_concat("truth","and beauty","truth and beauty") succeeds.

lowercase/2

lowercase(+string,?string)

lowercase(String1,String2) is true iff String2 is a lower case version of String1.

Ex.1: lowercase("Truth and Beauty","truth and beauty") succeeds.
Ex.2: lowercase("Truth and Beauty",W) succeeds with W bound to the string "truth and beauty".

uppercase/2

uppercase(+string,?string)

uppercase(String1,String2) is true iff String2 is an upper case version of String1.

Ex.1: uppercase("Truth and Beauty","TRUTH AND BEAUTY") succeeds.
Ex.2: uppercase("Truth and Beauty",W) succeeds with W bound to the string "TRUTH AND BEAUTY".

capitalized/1

capitalized(+string)

capitalized(String) is true iff String is capitalized.

Ex.1: capitalized("John") succeeds.

A.4 Tree Predicates

proof_tree/2

proof_tree(+goal,?tree)

proof_tree(Goal,Tree) is true iff Tree is the proof tree corresponding to the derivation of Goal.

Ex.1: Given a theory consisting of the following statements:

```
syncat(P0-Pn,np) :- syncat(P0-Pi,det), syncat(Pi-Pn,n).
syncat(1-2,det).
syncat(2-3,n).
```

?- proof_tree(syncat(S,np),T) succeeds and T is bound to the tree
syncat(1-3,np)/(syncat(1-2,det)/true,syncat(2-3,n)/true)

parse_tree/2

parse_tree(+goal,?tree)

parse_tree(Goal,Tree) is true iff Tree is the parse tree corresponding to the derivation of Goal.

Ex.1: Given a theory consisting of the following statements:

```
syncat(P0-Pn,np) :- syncat(P0-Pi,det), syncat(Pi-Pn,n).
syncat(1-2,det).
syncat(2-3,n).
bstring(1-2,"a").
bstring(2-3,"box").
```

?- parse_tree(syncat(S,np),T) succeeds with T bound to the tree
np/(det/"a",n/"box")

node/2

node(+tree,?term)

node(Tree,Node) is true iff Node is a node in the tree Tree.

Ex.1: node(pp/(p/"in",np/(det/"a"),n/"box")), np) is true.

Ex.2: node((syncat(1-3,np)/(syncat(1-2,det)/true,syncat(2-3,n)/true)),T) succeeds with T bound to syncat(1-3,np), syncat(1-2,det), etc.

subtree/2

subtree(+tree,?tree)

subtree(Tree,SubTree) is true iff SubTree is a sub-tree of Tree.

Ex.1: subtree(pp/(p/"in",np/(det/"a"),n/"box")), n/"box")) is true.

Ex.2: subtree((syncat(1-3,np)/(syncat(1-2,det)/true,syncat(2-3,n)/true)),T) succeeds with T bound to the tree syncat(1-3,np)/(syncat(1-2,det)/true,syncat(2-3,n)/true), the tree syncat(1-2,det)/true, etc.

sister/3

sister(+Tree,?Node,?Node)

sister(Tree,Node1,Node2) is true iff Node1 is a sister node of Node2 in the tree Tree.

parent/3

parent(+Tree,?Node,?Node)

parent(Tree,Parent,Child) is true iff Parent is the parent of the node Child in the tree Tree.

ancestor/3

ancestor(+Tree,?Node,?Node)

ancestor(Tree,Node1,Node2) is true iff Node1 is an ancestor of Node2 in the tree Tree.

A.5 Set and Bag Predicates

ext/2

ext(+set-expression,?list)

ext(SetExpression,List) is true iff List is the list of elements in the set denoted by SetExpression.

Ex.1: ext({W|bstring(S,W)},L) succeeds with L bound to the list [and,beauty,truth].

card/2

card(+set-expression,?integer)

card(Set,Card) is true iff Card is the cardinality of Set. The Count Tool can be regarded as a graphical interface to this built-in predicate.

Ex.1: card({W|bstring(S,W)},N) succeeds with N bound to 3.

Ex.2: card({S|bstring(S,W)},N) succeeds with N bound to 3.

A.6 Other Metalevel Predicates

not/1
\+/1

not(+goal)

not(Goal) is true iff Goal cannot be proved. This is negation as failure, as in Prolog.

max_provable/2

max_segment(?segment,+goal)

max_segment(Segment,Goal) is true iff Goal is true and there is no segment of which Segment is a proper subsegment for which Goal is true.

ebg/2

ebg(+goal,?statement)

ebg(Goal,Statement) is true iff Statement is bound to a universal statement that can be used to derive Goal in exactly one step from the singular statements of the current theory. This predicate implements *explanation-based generalization*.

consistent/1

consistent(+goal)

consistent(P) is true iff the (explicit) negation of P cannot be proved.

consistent/0

consistent is true iff the current theory does not allow a statement and its (explicit) negation to be proved.

inconsistent/0

inconsistent is true iff for some P in the current theory, P and ¬ P can be proved.

inconsistent/1

inconsistent(?segment)

inconsistent(S) is true iff the segment S has received (at least) two different and conflicting descriptions. (The segment might be ambiguous.) It is the task of the user to define inconsistent/1.

add/1

add(+clause)

add(Clause) asserts Clause in the current theory if it is not there already, and if it does not violate any integrity constraints in doing so (i.e. consistent(Clause) must be true). This procedure is used by the tagging commands.

delete/1

delete(+clause)

delete(Clause) deletes all clauses that match Clause from the current theory.

replace/2

replace(+clause,+clause)

replace(Pattern,NewPattern) replaces each instance of Pattern in the current theory with an instance of NewPattern.

Ex.1: replace(function(S,question), function(S,request_for_info)).

prove/1
provable/1

prove(+goal)

prove(Goal) is true if Goal can be proved in the current theory. Variables in Goal might be bound in the process.

A.7 Arithmetic

is/2

is(?expression,+expression)

is(Value,Expression) is true iff Value is the arithmetic value of Expression.

Ex.1: 5 is 3+2 succeeds.

Ex.2: X is 3+2 succeeds with X bound to 5.

>/2

>(+expression,+expression)

>(Expression1,Expression2) is true iff the value of Expression1 is greater than the value of Expression2.

Ex.1: 2+4 > 2+2 succeeds.

Ex.2: 2+4 > 6 fails.

>=/2

>=(+expression,+expression)

>=(Expression1,Expression2) is true iff the value of Expression1 is greater than, or equal to, the value of Expression2.

Ex.1: 2+4 >= 2+2 succeeds.

Ex.2: 2+4 >= 6 succeeds.

</2

<(+expression,+expression)

<(Expression1,Expression2) is true iff the value of Expression1 is less than the value of Expression2.

Ex.1: 2+4 < 2+5 succeeds.

Ex.2: 2+4 < 6 fails.

=</2

=<(+expression,+expression)

=<(Expression1,Expression2) is true iff the value of Expression1 is less than, or equal to, the value of Expression2.

Ex.1: 2+4 =< 2+5 succeeds.

Ex.2: 2+4 =< 6 succeeds.

A.8 Schema Predicates

text/1

text(?string)

text(Text) is true iff Text is the name of a text in the current corpus, as declared in the schema.

theory/1

theory(?string)

theory(Theory) is true if Theory is available for the current corpus.

tag/1

tag(?atom)

tag(Tag) is true iff Tag is declared as a tag in the current schema.

tag_value/2

tag_value(?atom,?atom)

tag_value(Tag,Value) is true iff Value is a possible value of the tag Tag.

property/1

property(?atom)

property(P) is true iff P is declared as a property in the current schema.

property_value/2

propert_value(?atom,?atom)

property_value(P,V) is true iff V is a possible value of the property P.

single_valued/1

single_valued(?atom)

single_valued(Property) is true iff Property is constrained to have only one value. The effect of declaring single_valued(P) true for a property P is that if P(S,V1) holds for a segment S and a property value V1, and V2 is different from V1, then ~P(S,V2) holds.

closed/2

closed(?atom,?atom)

closed(Text,Property) is true if Property is closed in the theory describing Text. If a property P involved in a statement A is declared as being closed, and A is not derivable from the current theory, the negation of A (i.e. ¬ A) can be derived.

A.9 Presentation Procedures

select/1

select(+segment)

select(Segment) selects (highlights) the segment Segment in the current text.

selection/1

selection(?segment)

selection(Segment) is true iff the segment Segment is selected in the current text.

excerpt/2

excerpt(+segment,+integer)

excerpt(Segment,N) will write the string corresponding to Segment, surrounded by a context of plus/minus N lines of text, and terminated by a newline, to the current output channel.

show_statement/2

show_statement(+segment,+boolean)

show_statement(Segment,DerivationStatus) will write the statements describing Segment to the current output channel. If DerivationStatus is 'true', derived statements will be included, if it is 'false' they will not be included. This procedure is used by the Statement Browser.

show_concordance/4

show_concordance(+set-expression,+integer,+integer,+atom)

show_concordance(SetExpression,PreContext,PostContext,SortOrder) prints the concordance specified by SetExpression to the current output channel. Every string corresponding to a segment in the set specifed by SetExpression is written to the current output channel, in the order specifed by SortOrder, prefixed by a line-number plus the textual context consisting of the PreContext number of characters, and suffixed by the PostContext number of characters following the string. SortOrder is one of the atoms 'appearance' or 'alphabetical'. This procedure is used by the Concordance Tool.

show_table/3

show_table(+set-expression,+atom,+atom)

show_table(SetExpression,SortOrder,Dimension) prints the table specified by SetExpression to the current output channel. SortOrder is one of the atoms 'ascending' or 'descending'. This procedure is used by the Table Tool.

show_statistics/1

show_statistics(+set-expression)

show_statistics(SetExpression) prints a table of descriptive statistics to the current output channel. This procedure is used by the Statistics Tool.

show_clauses/1

show_clauses(+set-expression)

show_clauses(SetExpression) lists the set of clauses denoted by SetExpression to the current output channel, in such a form that they can be consulted by the TagLog system, or by an Edinburgh family Prolog system.

A.10 Corpus Management Procedures

load_text/1

load_text(+atom)

load_text(TextId) loads the text denoted by TextId.

close_text/0

close_text

close_text closes the current text.

ca_run/0

ca_run executes all condition-action rules, in the order that they are found.

APPENDIX B

Selected Code

B.1 Viterbi Tagging in Prolog

In this section an implementation of Viterbi tagging, in less than twenty lines of Prolog code, is given.

The goal is to map a sequence of words $<W_1,..,W_n>$ into the most probable sequence of tags $<T_1,..,T_n>$.

The basic idea is to maintain a set *PSs* of possible sequences and to discard the impossible ones as early as possible. Suppose we read a word W_i from the input list of words. Then for each state T_i in the HMM such that T_i outputs W_i, and for each sequence P_{i-1}-$<T_1,..,T_{i-1}>$ in PSs_{i-1}, PSs_i is formed by all probability-sequence pairs P_i-$<T_1,..,T_{i-1},T_i>$ such that $P_i = max\ p(W_i|T_i)p(T_i|T_{i-1})$. The most probable sequence in PSs_n is the one we are looking for.

Here are some implementation details: The set of probability-sequence pairs are represented as lists of terms of the form *P-Sequence*, where *Sequence* is a list of terms representing tags,[1] and *P* is a number representing the probability for that sequence. During processing, it is conven-

ient to keep the sequences in reverse order (thus P_i-<T_1,..,T_{i-1},T_i> is represented by the term P_i-[T_i,T_{i-1},..,T_1]) so that adding a tag to a sequence becomes a matter of putting it first in the list. Note also the use of a dummy 'start' element. This is so that we do not need to complicate pattern matching further down in the program. The last thing this program does is to reverse the most probable sequence and remove the dummy.

The processing of a word is taken care of by nesting two calls to *findall/3*. The predicate *max_key/2* is a simple utility predicate that finds the element with the largest key (i.e. the maximum probability sequence in our case) in a list of key-value pairs.

(298)
```
most_probable_sequence(Words,P-Sequence) :-
        sequences(Words,[1-[start]],PSs),
        max_key(PSs,P-Sequence1),
        reverse(Sequence1,[start|Sequence]).

sequences([],PSs,PSs).
sequences([Word|Words],PSs0,PSs) :-
        findall(PS2,
                (outprob(Word,T2,PL),
                findall(P2-[T2,T1|Ts],
                        (member(P1-[T1|Ts],PSs0),
                        transprob(T1,T2,PT),
                        P2 is PL*PT*P1),
                    PSs),
                max_key(PSs,PS2)),
            PSs1),
        sequences(Words,PSs1,PSs).
```

The lexicon is represented as clauses of the form *outprob(Word, Tag, Probability)*, e.g. as:

(299)
```
outprob(a,det,0.300).
outprob(can,aux,0.010).
outprob(can,v,0.005).
outprob(can,n,0.001).
outprob(he,pron,0.070).
```

---

1. One useful aspect of doing this in Prolog is that the tags are allowed to be general terms, not just atoms.

The transitional probabilities are represented as clauses of the form *transprob(Tag1,Tag2, Probability)*, e.g. as:

(300)
```
transprob(start,det,0.30).
transprob(start,aux,0.20).
transprob(start,v,0.10).
transprob(start,n,0.10).
transprob(start,pron,0.30).

transprob(det,det,0.20).
transprob(det,aux,0.01).
transprob(det,v,0.01).
transprob(det,n,0.77).
transprob(det,pron,0.01).

transprob(aux,det,0.18).
transprob(aux,aux,0.10).
transprob(aux,v,0.50).
transprob(aux,n,0.01).
transprob(aux,pron,0.21)

transprob(v,det,0.36).
transprob(v,aux,0.01).
transprob(v,v,0.01).
transprob(v,n,0.26).
transprob(v,pron,0.36).

transprob(n,det,0.01).
transprob(n,aux,0.25).
transprob(n,v,0.39).
transprob(n,n,0.34).
transprob(n,pron,0.01).

transprob(pron,det,0.01).
transprob(pron,aux,0.45).
transprob(pron,v,0.52).
transprob(pron,n,0.01).
transprob(pron,pron,0.01).
```

Given this data, *most_probable_sequence/2* might be called as follows:

(301)
```
?- most_probable_sequence([he,can,can,a,can],P-Sequence).
P = 1.965e-11
Sequence = [pron,aux,v,det,n]
```

This code has not been integrated with the TagLog system yet. But by adding a tokenizer, procedures for opening and closing files, and procedures for writing to a file, I have built a small but useful standalone automatic part-of-speech tagger system (less than sixty lines of Prolog all together). Initial testing shows that it is able to tag 10,000 words in 15 seconds (i.e. around 670 words per second) on a SPARCstation 10, running SICStus Prolog 3.0 (compiled code).

B.2 An Improved Error-Driven Learner

Here is a greatly improved version of the implementation of the error-driven learner, discussed in Section 11.5. When tested with 6 templates and 20 different tags on a >2000 word corpus, it was able to find 20 rules within a couple of hours on a Sparcstation 10 running Sicstus Prolog. This is not fast, but it might be fast enough to be of some use.

```
/*****************************************************************************************
Declarations
*****************************************************************************************/

        :- op(1100, xfy,==>).
        :- op(400, fy,*).

        :- dynamic lexcat/2.

        :- use_module(library(lists)).

/*****************************************************************************************
learn/1 --- learn(+Threshold) starts by computing a list of all error types present in the
current data description on the level of lexical categories. Clauses of the form *lex-
cat(S,C) (where * is a prefix functor) are assumed to be clauses in the supposedly
true hand coded theory. Clauses of the form lexcat(S,C) are assumed to be clauses
produced by lexicon lookup and an (imperfect) set of transformation-rules. The
nested setof-bagof call generates a list of terms of the form (N,C1,C2) where C1 is
the tag value which in N cases is present instead of the correct tag value C2. The call
to the procedure find_best/5 seeks the rule that fixes more error occurrences than
any other rule. learn/1 terminates if the Score for that rule is less than Threshold.
Else it executes the rule, and goes on to learn from there.
*****************************************************************************************/

        learn(Threshold) :-
                setof((N,C1,C2),
                        Ss^(bagof(S, ( *lexcat(S,C2), lexcat(S,C1), C1 \== C2),Ss),
                                length(Ss,N)),
                        ErrorList),
                reverse(ErrorList,RevErrorList),
                find_best(RevErrorList,no_rule,Threshold,Winner,Score),
                Score >= Threshold,
                write(Winner),nl,
                call(Winner),
                learn(Threshold).
        learn(_Threshold).
```

```
/*************************************************************************************************
find_best/5 --- find_best(+ErrorList,+LeaderRule,+LeaderScore,-WinnerRule,-
WinnerScore) tries to find the rule that eliminates more errors than any other rule.
Beginning with the most frequent error it goes through the list of errors keeping track
of a leading rule and its score. When the error list is empty or the most frequent error
type in what remains of the error list is less than the leading rule's score, the leader is
declared winner. Note how the call to findall/3 is used to generate the list of rules that
are instances of the templates that are already partially instantiated by the error list,
and which now becomes more instantiated (in different ways) by the binding of C3 to
possible values of lexcat as given in the TagLog schema.
*************************************************************************************************/

        find_best([],Winner,Score,Winner,Score).
        find_best([(N,C1,C2)|Ls],Winner,Score,Winner,Score) :-
                Score >= N, !.
        find_best([(_N,C1,C2)|Ls],Leader,Score,Winner,WScore) :-
                findall(Rule,(template([C1,C2,C3],Rule),tag_value(lexcat,C3)),Rules),
                (       find_best1(Rules,Leader,Score,NewLeader,NewScore) ->
                        find_best(Ls,NewLeader,NewScore,Winner,WScore)
                ;       find_best(Ls,Leader,Score,Winner,WScore)
                ).
        find_best1([],Winner,Score,Winner,Score).
        find_best1([NewRule|Rules],BestRule,BestScore,Winner,Score) :-
                (       evaluate(BestScore,NewRule,NewScore) ->
                        find_best1(Rules,NewRule,NewScore,Winner,Score)
                ;       find_best1(Rules,BestRule,BestScore,Winner,Score)
                ).


/*************************************************************************************************
evaluate/3 --- evaluate(+BestScore,+Rule,-NewScore) succeeds if the score of Rule
is higher than BestScore, and binds NewScore to the score of Rule. The idea is to
determine the difference between the number of cases where the current tagging is
correct, and therefore the transformation rule will do some damage, and the number
of cases where the current tagging is incorrect and the transformation will correct it.
This difference is the score of the rule.
*************************************************************************************************/

        evaluate( BestScore, (Condition ==> replace(A, B)), Score) :-
                count((*B, Condition), Positive),
                Positive > BestScore,    % else there is no hope for improvement
                count((*A, Condition), Negative),
                Score is Positive - Negative,
                Score > BestScore.


/*************************************************************************************************
Various utility predicates
*************************************************************************************************/

        count(Goal,N) :-
                findall(.,Goal,Dots),
                length(Dots,N).
```

```
/*****************************************************************************************
Sample templates
*****************************************************************************************/
        template([A,B,C],(   lexcat(P0-P1,A), lexcat(P1-P2,C) ==>
                                    replace(lexcat(P0-P1,A), lexcat(P0-P1,B)))).
        template([A,B,C],(   lexcat(P0-P1,C), lexcat(P1-P2,A) ==>
                                    replace(lexcat(P1-P2,A), lexcat(P1-P2,B)))).

/*****************************************************************************************
Condition-Action rule interpreter
*****************************************************************************************/
        ==>(Condition,Action) :-
              call(Condition),
              once(Action),
              fail.
        ==>(_Condition,_Action).
        once(Goal) :- call(Goal), !.
        replace(A,B) :-
              retract(A),
              assert(B).
```

APPENDIX C

An Overview of Corpus Texts Used

C.1 Introduction

To save space, and to simplify the exposition, I have, to some extent, in this thesis, allowed myself to use laboratory sentences, made-up texts and toy theories. On the other hand, some of the example texts *are* taken from real corpora, and in this appendix I intend to briefly describe these texts.

C.2 The Brown Corpus Text

This is one of the 500 texts of the well-known Brown corpus of written American English (Francis & Kučera, 1982). It is numbered *J23*. Francis and Kučera gives the following publication details:

> Joyce O. Hertzler, American Social Institutions; A Sociological Analysis, Allyn and Bacon, Inc., pp. 478-482.

The text consists of around 2300 words, and is classified as a piece of 'learned (mainly scientific and technical) writing'.

SOME OF THE MAJOR FUNCTIONS OF RELIGION

The place of religion in the simple, preliterate societies is quite definite; as a complex it fits into the whole social organization and functions dominantly in every part of it. In societies like ours, however, its place is less clear and more complex. With the diversity of religious viewpoints, there are differences of opinion as to the essential features of religion; and there are different opinions as to the essential functions of religion. Nevertheless, for most of the population of heterogeneous advanced societies, though less for the less religious portion, religion does perform certain modal individual and social functions.

Although the inner functions of religion are not of direct significance in social organization, they have important indirect consequences. If the inner functions of religion are performed, the individual is a composed, ordered, motivated, and emotionally secure associate; he is not greatly frustrated, and he is not anomic; he is better fitted to perform his social life among his fellows. There are several closely related inner functions.

In the last analysis, religion is the means of inducing, formulating, expressing, enhancing, implementing, and perpetuating man's deepest experience – the religious. Man is first religious; the instrumentalities follow. Religion seeks to satisfy human needs of great pertinence. The significant things in it, at the higher religious levels, are the inner emotional, mental, and spiritual occurrences that fill the pressing human needs of self-preservation, self-pacification, and self-completion. The chief experience is the sensing of communion, and in the higher religions, of a harmonious relationship with the supernatural power. Related to this is the fact that most of the higher religions define for the individual his place in the universe and give him a feeling that he is relatively secure in an ordered, dependable universe. Man has the experience of being helpfully allied with what he can not fully understand; he is a coordinate part of all of the mysterious energy and being and movement. The universe is a safe and permanent home.

A number of religions also satisfy for many the need of being linked with the ultimate and eternal. Death is not permanent defeat and disappearance; man has a second chance. He is not lost in the abyss of endless time; he has endless being. Religion at its best also offers the experience of spiritual fulfillment by inviting man into the highest realm of the spirit. Religion can summate, epitomize, relate, and conserve all the highest ideals and values – ethical, aesthetic, and religious – of man formed in his culture.

There is also the possibility, among higher religions, of experiencing consistent meaning in life and enjoying

guidance and expansiveness. The kind of religious experience that most moderns seek not only provides, clarifies, and relates human yearnings, values, ideals, and purposes; it also provides facilities and incitements for the development of personality, sociality, and creativeness. Under the religious impulse, whether theistic or humanistic, men have joy in living; life leads somewhere. Religion at its best is out in front, ever beckoning and leading on, and, as Lippman put it, "mobilizing all man's scattered energies in one triumphant sense of his own infinite importance".

At the same time that religion binds the individual helpfully to the supernatural and gives him cosmic peace and a sense of supreme fulfillment, it also has great therapeutic value for him. It gives him aid, comfort, even solace, in meeting mundane life situations where his own unassisted practical knowledge and skill are felt by him to be inadequate. He is confronted with the recurrent crises, such as great natural catastrophes and the great transitions of life – marriage, incurable disease, widowhood, old age, the certainty of death. He has to cope with frustration and other emotional disturbance and anomie. His religious beliefs provide him with plausible explanations for many conditions which cause him great concern, and his religious faith makes possible fortitude, equanimity, and consolation, enabling him to endure colossal misfortune, fear, frustration, uncertainty, suffering, evil, and danger. Religion usually also includes a principle of compensation, mainly in a promised perfect future state.

The belief in immortality, where held, functions as a redress for the ills and disappointments of the here and now. The tensions accompanying a repressive consciousness of wrongdoing or sinning or some tormenting secret are relieved for the less self-contained or self-sufficient by confession, repentance, and penance. The feeling of individual inferiority, defeat, or humilation growing out of various social situations or individual deficiencies or failures is compensated for by communion in worship or prayer with a friendly, but all-victorious Father-God, as well as by sympathetic fellowship with others who share this faith, and by opportunities in religious acts for giving vent to emotions and energies.

In providing for these inner individual functions, religion undertakes in behalf of individual peace of mind and well-being services for which there is no other institution.

In addition to the functions of religion within man, there have always been the outer social functions for the community and society. The two have never been separable. Religion is vitally necessary in both societal maintenance and regulation.

The value-system of a community or society is always correlated with, and to a degree dependent upon, a more or

less shared system of religious beliefs and convictions. The religion supports, re-enforces, reaffirms, and maintains the fundamental values. Even in the United States, with its freedom of religious belief and worship and its vast denominational differentiation, there is a general consensus regarding the basic Christian values. This is demonstrated especially when there is awareness of radically different value orientation elsewhere; for example Americans rally to Christian values vis-a-vis those of atheistic communism. In America also all of our major religious bodies officially sanction a universalistic ethic which is reflective of our common religion. Even the non-church members – the freewheelers, marginal religionists and so on – have the values of Christian civilization internalized in them. Furthermore, religion tends to integrate the whole range of values from the highest or ultimate values of God to the intermediary and subordinate values; for example, those regarding material objects and pragmatic ends. Finally, it gives sanctity, more than human legitimacy, and even, through super-empirical reference, transcendent and supernatural importance to some values; for example, marriage as a sacrament, much law-breaking as sinful, occasionally the state as a divine instrument. It places certain values at least beyond questioning and tampering.

Closely related to this function is the fact that the religious system provides a body of ultimate ends for the society, which are compatible with the supreme eternal ends. This something leads to a conception of an over-all Social Plan with a meaning interpretable in terms of ultimate ends; for example, a plan that fulfills the will of God, which advances the Kingdom of God, which involves social life as part of the Grand Design. This explains some group ends and provides a justification of their primacy. It gives social guidance and direction and makes for programs of social action. Finally, it gives meaning to much social endeavor, and logic, consistency, and meaning to life. In general, there is no society so secularized as to be completely without religiously inspired transcendental ends.

Religion integrates and unifies. Some of the oldest, most persistent, and most cohesive forms of social groupings have grown out of religion. These groups have varied widely from mere families, primitive, totemic groups, and small modern cults and sects, to the memberships of great denominations, and great, widely dispersed world religions. Religion fosters group life in various ways. The common ultimate values, ends and goals fostered by religion are a most important factor. Without a system of values there can be no society. Where such a value system prevails, it always unifies all who possess it; it enables members of the society to operate as a system. The beliefs of a religion also reflecting the values are expressed in creeds, dogmas, and doctrines, and form what Durkheim calls a credo. As he points out, a religious group can not exist without a collective credo, and the more extensive

the credo, the more unified and strong is the group. The credo unifies and socializes men by attaching them completely to an identical body of doctrine; the more extensive and firm the body of doctrine, the firmer the group.

The religious symbolism, and especially the closely related rites and worship forms, constitute a powerful bond for the members of the particular faith. The religion, in fact, is an expression of the unity of the group, small or large. The common codes, for religious action as such and in their ethical aspects for everyday moral behavior, bind the devotees together. These are ways of jointly participating in significantly symbolized, standardized, and ordered religiously sanctified behavior. The codes are mechanism for training in, and directing and enforcing, uniform social interaction, and for continually and publicly reasserting the solidarity of the group.

Durkheim noted long ago that religion as "...a unified system of beliefs and practices relative to sacred things... unites into one single moral community... all those who adhere to them". His view is that every religion pertains to a community, and, conversely, every community is in one aspect a religious unit. This is brought out in the common religious ethos that prevails even in the denominationally diverse audiences at many secular semi-public and public occasions in the United States; and it is evidenced in the prayers offered, in the frequent religious allusions, and in the confirmation of points on religious grounds.

The unifying effect of religion is also brought out in the fact that historically peoples have clung together as more or less cohesive cultural units, with religion as the dominant bond, even though spatially dispersed and not politically organized. The Jews for 2500 years have been a prime example, though the adherents of any world or interpeople religion are cases in point. It might be pointed out that the integrating function of religion, for good or ill, has often supported or been identified with other groupings – political, nationality, language, class, racial, sociability, even economic.

Religion usually exercises a stabilizing-conserving function. As such it acts as an anchor for the people. There is a marked tendency for religions, once firmly established, to resist change, not only in their own doctrines and policies and practices, but also in secular affairs having religious relevance. It has thus been a significant factor in the conservation of social values, though also in some measure, an obstacle to the creation or diffusion of new ones. It tends to support the longstanding precious sentiments, the traditional ways of thinking, and the customary ways of living. As Yinger has pointed out, the "...reliance on symbols, on tradition, on sacred writings, on the cultivation of emotional feelings of identity and harmony with sacred values, turns one to the

```
past far more than to the future". Historically, religion has
also functioned as a tremendous engine of vindication,
enforcement, sanction, and perpetuation of various other
institutions.

At the same time that religion exercises a conserving
influence, it also energizes and motivates both individuals
and groups. Much of the important individual and social
action has been owing to religious incentives. The great
ultimate ends of religion have served as magnificent beacon
lights that lured people toward them with an almost
irresistible force, mobilizing energies and inducing
sacrifices; for example, the Crusades, mission efforts,
just wars. Much effort has been expended in the sincere
effort to apply the teaching and admonitions of religion. The
insuperable reward systems that most religions embody
have great motivating effects. Religion provides the most
attractive rewards, either in this world or the next, for
those who not merely abide by its norms, but who engage
in good works.

Religion usually acts as a powerful aid in social control,
enforcing what men should or should not do. Among
primitive peoples the sanctions and dictates of religion
were more binding than any of the other controls exercised
by the group; and in modern societies such influence is still
great. Religion has its own supernatural prescriptions that
are at the same time codes of behavior for the here and
now.
```

The Brown corpus text has been described on the level of parts of speech. Here is an overview of the 28 tags that have been used.

TABLE 3. List of Tags

adj	adjective
adv	adverb
aux	auxiliary verb
comma	comma
conj	coordinating conjunction
det	determiner
ex	existential there
inf	infinitive marker
number	cardinal number
part	particle
pers_pron	personal pronoun
poss_pron	possesive pronoun
pre_det	pre-determiner
prep	preposition
punct	punctuation
quote	quote

TABLE 3. List of Tags

wh_det	wh-determiner
wh_pron	wh-pronoun
n(pl)	plural noun
n(sg)	singular noun (including mass nouns)
pn(pl)	plural proper noun
pn(sg)	singular proper noun
v(base)	verb, base form
v(non3sg_pres)	verb, non-3rd person singular present
v(past)	verb, past tense
v(past_participle)	verb, past participle
v(pres_participle)	verb, present participle
v(sg3_pres)	verb, 3rd person singular present

C.3 The Doctor-Patient Dialogue Corpus

This corpus consists of transcriptions of doctor-patient dialogues prepared by Kós-Dienes (forthcoming). Based on this data, Kós-Dienes is analysing different aspects of communicative mechanisms, e.g. subactivity structure and feedback, in medical consultations. This is one of her transcriptions, the one that I have used in the thesis:

```
@  Recorded activity ID: A5008.1.1
@  Recorded activity date: 891004
@  Recorded activity title: Patient-Doctor Conversation: ERYSIPELAS
@  Short name: ERYSIPELAS
@  Tape(s): A5008, KA5008
@  Participant: P = Old Woman
@  Participant: D = Dr. X
@  Transcription name: TA5008.1.1
@  Transcriber(s): Unknown
@  Transcription date: 000000
@  Transcribed segments: All
@  Transcription system: MSO2.1
@  Character set: MAC
@  Checker(s):
@  Checking date:
@  Time coding: Yes

P: har det lugnat ner sig
D: det här kommer du nog att få ha kvar därför [att] (...)
P: [men]
P: doktorn är det inte inne i benet jag tycker det känns  så /
```

```
   och här på sidan
D: ja \
D: böj lite grann och sträck </> \
- <examination>
D: hade du haft sår här </>  här
 - <paper rustles>
P: (...)
D: ja hur länge sedan var det då
P: ja nu är det nog ett par månar sen minst / det gör ont
D: jaha jag förstår det\
- <examination>
D: det är ju fortfarande inflammerat i huden och här under [å] alldeles
   hårt [som]
P: [ja vet]
P: [innuti]
P: tror doktorn
D: nä det är inte i benet utan [de]
P: [men] det känns så
D: har du skadat den här foten någon gång
P: nähä jag har väldigt dåliga ben de har jag alltid haft men /
D: kan du ligga ner
P: vad tror doktorn om (...) det är väl också (...)
D: hm \
D: böj lite så /  nej så kan du vila foten sådär och slappna av i foten
   så ja hm <///> här är det ont
 - <examination>
P: ja <///>
 - <examination>
D: nu masserar jag här lite \
D: här måste du haft någon propp någon gång
P: ja ja har undrat hur det har vart [blått där] man skulle väl gå
   oftare än man gör kanske när det är något
D: [ja]
D: ja man gör inte så mycket åt de iallafall men men det är så hårt
(...)
P: jaha
D: har du någon salva att massera in här nu
P: vassa
D: har du någon salva du masserar in
P: ja ja fick ju utav av doktorn två [tuber]
D: [ja]
D: men det var för att huden skulle och den är ju ganska mjuk nu men
   just mot det här ömma [har] du haft något so heter Hirudoid salva
   någon gång
P: [ja ja fick]
P: nej det har ja aldrig haft
D: för då pratar vi alltså o salva mot de [hårda] och ömma här
P: [mot bara det jaha]
P: jah men jag fick den var det inte det jag fick av doktorn för jag
   sa att]
D: mjaa
P: jag hade mjukt det hela benen
D: jo men det var för att huden [skulle]
P: [jasså] jaha
D: bli mjukare
P: jaha <///>
 - <examination>
```

D: hade du varit och lämnat urinprov igår det var bra ([...)]
P: [syster] sa någonting om att jag har tatt blod(...)
D: ja
P: vava de
D: de var det här som [va] det kallas för rosfeber
P: [jasså]
P: jaha
D: ska vi passa på och se hur du har det med trycket också
P: ska jag ta av mig
D: ja det går nog bra
P: det går
D: det går fint (...) </> ja // hm <///> det håller sig hundraåtti genom nitti fastän (...)
- <noises as doctor measures blood pressure>
- <pumping noises>
P: jaha
D: det brukar va (...)
P: tack
D: ska du ta på och komma in till mig [istället]
P: [ja] tänkte säga det jag glömde säga till doktorn det brutna att jag var ju ute för en bussolycka
D: ja men det berättade du
P: gjorde jag det
D: ja
P: åh jag är nog väldigt [ ]
@ väl[digt]
D: [ja] det förstår jag nu bråkade vi ju med några av revbenen men fick du ont efteråt tror du
P: nää då men jag har inte\
P: men det är väl någonting som är stukat \
P: [å] så på morgonen när jag ska ur sängen [så] \
D: [hmm]
D: [ja]
P: men det är inget å göra kanske ///
D: <där ja>
- <examination>
P: ja
D: den smärtan kommer från lederna här mellan koterna och revbenet och så följer den upp längst kanten så här
P: och här då
D: hm det är samma sak fast lite längre ner \
D: ja ska inte trycka där mer \
D: men DE är väldigt svårt åå bota det där \
D: i och med att du har en lite böjd rygg som gärna blir ö här särskilt o man får en a där DE är ingenting som är sönder MEN det gör ju ont och blir
stelt
P: ja
D: så det är viktigt att du mjukar upp den på [morgonen]
P: [ska] försöka det
D: det är viktigt att du inte ligger med många kuddar [(...)]
P: [ja] det sa doktorn [ja]
D: [försök] å räta [ut] dig å sitt så att du kan ställa huvet lite å spärra [här] så går det bättre / men det kommer ju å bättra sig det bör ju (...)
P: [jaha]
P: [ja]

P: ja tack
D: ta på och kom in (...)
P: ja
D: stig fram och sitt ska du se \
D: gör du nu så att du tar fortfarande en Moduretic för blodtrycket
P: hmmm
D: den där kantiga tabletten (...)
P: det har jag väl ingen
D: nää du hade det [förut]
P: [ja] har haft det men den är nog slut
D: ja men tar du då två Lasix tabletter
P: ja det gör jag varje [dag] och två Kali Kalium
D: [(...)]
D: ja \
D: då kan vi fortsätta så \
D: sen eeh själva infektionen är ju behandlad i benet
P: ehh den
D: alltså bakterierna [denne] rosfeber [den] är behandlad så de har du inte kvar men du har ju de ömma och svullna å lite ömtåliga röda huden
P: [jasså]
P: [ehh]
P: jah
D: å de måste man fortsätta å behandla när det gäller de onda där ja tryckte å de gjorde ont där det känns liksom hårt och svullet där ska du massera in den här salvan som står som jag skriver
P: jaha
D: masseras en eller två gånger dagligen mot värk
P: jah
D: då mjukar de upp
P: ja
D: då får man mindre ont
D: [men] själva huden sen som blir lite torr och [fnasig]
P: [menar]
P: [ja] det har
P: [ja]
D: [då] har du de där krämerna som du fick förut men de är alltså inte mot värken
P: menar doktorn att de har förbindelse med bakterierna i urinen då
D: näh
P: men doktorn sa att
D: ja sa att samma sorts penicillin botar även bakterier i urinen [de] de verkar ju [i hela kroppen] / [de verkar ju i hela kroppen] å de har du ju inte så att urinprovet var ju alldeles bra
P: [jasså]
P: [så de också då]
P: [jaha jaha] ja de var väl skönt
D: ja men det är klart att du får så här på benen det beror på att du har en lite dålig cirkulation du har dom åderbrocken inuti \
D: det är väldigt svårt att operera det där de blir ofta sämre
P: ja ja vet dom vill inte [gärna] göra det heller
D: [nä]
D: näh å de e viktigt att du har stödstrumpor eller lindade ben va
P: ja ska köpa ett par stödstrumpor tror [jag] för ja har ju binder men de kanske håller bättre åt strumpor ja [vet]
D: [ja]
D: [nja] binder om man lindar binderna rätt [så] håller (...) men de

är ju krångligt åå linda så att
P: [ä de bättre]
P: jaja
D: nåt stöd måste du ha antingen då bindor ibland
P: ja
D: å strumpor ibland
P: jah
D: finns på apoteket och de finns i sjukvårdsaffärer
P: jah jah
D: men de är viktigt att du har det sen får vi väl hålla ett öga på dom där benen så att du bör komma tillbaks inom ett par månader vi ser hur det ser ut å e de några som helst problem innan så ska du höra av dig antingen hit eller till distriksssköterskan då som kan hjälpa [till] har du något mot värk å ta // både värk här och i benen
P: [jah]
P: de där me revbenen ja sa / ja va en gång på Carlanderska ja vet inte om de var magen eller va de va ja sökte för de va ju inte alls revbenen men då när syster innan jag skulle gå ja vet inte hon hjälpte mig med något då så da hon jasså fru Z har skadat revbenen då ser dom ju revbena på röntgen då tänkte ja
D: mja vissa gånger kan man se
P: ja å de e de som gör att ja är så uppstoppad och konstig för jag har böjt har ramlat platt på golvet med händerna på varsidan där
D: ja
P: så de måste vara brutet på något konstigt sätt
D: jo men de är ju så att de e [läkt]
P: [man] får (...) genomläkta [ja ja vet inte]
D: [(...)] skadan är ju läkt
P: ja
D: men att det är stelt [och] obehagligt de e en [annan sak] \
D: de e ju viktigt att du kan röra på dig som jag sa och de är inget fel att något smärtstillande ibland bara man håller sig till de milda sorter sorterna
P: [ja de e så]
P: [ja de e så]
P: jah
D: ja skriver ut en burk vanliga Alvedon tabletter som du har här
P: ja de e väl ingenting Alvedon tycker ja är så hemska finns de inget som är bättre
D: jo oom du ska ta dom så behöver du ta två åt gången om det ska göra någon nytta
P: jah
D: men dom kommer dom skadar inte i magen till exempel och dom ställer inte till några problem de finns i brustablettform om du vill ha det
P: ja de tror ja i så fall e bättre
D: hmm de e bättre än å svälja dom å så verkar dom lite fortare då
P: jah
D: men de e så med Alvedon att de märks inte så där alldeles på en gång att man har tagit dom om du tar ett par Treo eller Bamyl så märker du efter en liten stund nu hjälper [de] så är det men så är [det inte med Alvedon] utan de tar lite mer tid
P: [jah ja]
P: [näh]
P: jah
D: men de är nog den sorten du bör hålla dig till tror jag
P: jaha
D: men du får nog ta två om det ska hjälpa

```
P: jah ja ska [försöka]
D: [hur e de] på nätterna me de onda då
P: jo de går bra de värst å komma ur sängen [på] morn här på sidorna
D: [ja]
D: de e [då]
P: [men ee] de får väl gå de med
D: (...) då gör jag så att jag bokar en tid så du kallas hit om [ett
   par månader så får vi titta] på benen va hmm [så behöver du inte
   tänka på de]
P: [ja tack]
P: [(...)] e doktorn kvar då
D: ja då e ja kvar vi får se (...) före julen
P: byter de så ofta här
D: ja tyvärr e de så de e /// de får du skylla på politikerna
P: jaha (...) tala me dom
D: ja telefonnumret står där ute i väntrummet
P: tack så mycket </>
 - <skratt>
D: tack ska du ha
```

C.4 The PLUS Corpus Text[1]

The aim of the *PLUS project* was to produce a robust natural language understanding component allowing flexible and efficient Human-Computer interaction. Robustness was to be achieved by treating natural language as a communicative activity whose essential characteristic is to convey a meaning that is both appropriate and relevant contextually. The system was to be tolerant to corrupt input and flexible enough to allow a real dialogue with the user, using contextual and pragmatic knowledge.

To provide the project with relevant information for the design of system components, and to provide useful information for the test and evaluation tasks, corpus studies were performed.

A total number of 106 dialogues have been collected. Of these, 12 were collected as part of a pilot study (12 human-human telephone, 4 human-human terminal). The remaining 90 dialogues consist of 30 Wizard-of-Oz dialogues, 30 human-human terminal dialogues, and 30 human-human telephone dialogues. In each type, the three scenarios (car hire, restaurant, insurance) are represented by 10 dialogues each. The total number of sub-

1. The presentation of the PLUS corpus is taken from (Nivre, 1992).

jects (excepting the pilot study) was 20. 19 were students, and 1 was an administrative employee.

```
@ PLUS WOZ Dialogues
@ DATE Feb 1991
@ U = User
@ S = System
@ SCENARIO Car Hire


U: I need a hire car for tommorrow afternoon

S: Chauffeur-driven or self-drive?

U: self-drive please

S: In which area?

U: well I live in Entwistlee but need to travel to Rochdale
   and the airport

S: Do you require a one way rental or a return rental?

U: only one way

S: The following companies have branches both in Bolton and at
   Manchester Airport:

      Budget Rent-a-Car, 061-499 3042
      EuroDollar Rent-a-Car, 061-499 3320
      Hertz, 061-437 8208

U: what rate is charged for car hire by Budget rent-a-car?

S: Please contact the company

U: which is the most reputable company?

S: Define reputable

U: reputable means well known and trusted

S: No information available

U: are there any other companies

S: Please be more specific

U: can the company deliver the car to my house

S: Please wait ...

   A delivery service is available on the following one way rentals:

     Avis Rent-a-Car, 061-436 2020
     Hertz, 061-437 8208

U: and can I leave a hertz hire car at the airport?

S: Yes

U: thankyou

U: quit
```

GOTHENBURG MONOGRAPHS IN LINGUISTICS
Doctoral dissertations from the Department of Linguistics, Göteborg University, Sweden.

1. Lars-Gunnar Andersson, *Form and Function of Subordinate Clauses*, 1975.
2. Jens Allwood, *Linguistic Communication as Action and Co-operation*, 1976.
3. Pierre G. Javanaud, *The Vowel System of Lemosin: A Phonological Study*, 1981.
4. Sven Strömqvist, *Make-believe through Words: A Linguistic Study of Children's Play with a Doll's House*, 1984.
5. Elisabeth Ahlsén, *Discourse Patterns in Aphasia*, 1985.
6. Sally Boyd, *Language Survival: A Study of Language Contact, Language Shift, and Language Choice in Sweden*, 1985.
7. Richard Hirsch, *Argumentation, Information, and Interaction*, 1989.
8. Sören Sjöström, *Spatial Relations: Towards a Theory of Spatial Verbs, Prepositions, and Pronominal Adverbs in Swedish*, 1990.
9. Anders Eriksson, *Aspects of Swedish Speech Rhythm*, 1991.
10. Anders-Börje Andersson, *Second Language Learners' Acquisition of Grammatical Gender in Swedish*, 1992.
11. Joakim Nivre, *Situations, Meaning and Communication: A Situation Theoretic Approach to Meaning in Language and Communication*, 1992.
12. Kaarlo Voionmaa, *On the Semantics of Adult Verb Acquisition*, 1993.
13. Mats Dahllöf, *On the Semantics of Propositional Attitude Reports*, 1995.
14. Torbjörn Lager, *A Logical Approach to Computational Corpus Linguistics*, 1995.