LOGIC FORMS FOR WORDNET GLOSSES

Approved by:

________________________
Dr. Dan Moldovan

________________________
Dr. Hesham El-Rewini

________________________
Dr. Ian Gladwell

________________________
Dr. Peter Seidel

________________________
Dr. Jeff Tian
LOGIC FORMS FOR WORDNET GLOSSES

A Dissertation Presented to the Graduate Faculty of the
School of Engineering
Southern Methodist University

in
Partial Fulfillment of the Requirements
for the degree of
Doctor of Philosophy

with a
Major in Computer Science

by

Vasile Rus

B.Sc., Technical University of Cluj-Napoca, 1997
M.Sc., Southern Methodist University, 1999

May 18, 2002
ACKNOWLEDGMENTS

My deepest appreciation goes to my advisor, Professor Dan I. Moldovan, for his help in my graduate studies, guidance and encouragement when I was in trouble, which made it possible to achieve this work.

I would also like to thank the members of my committee Dr. Sanda M. Harabagiu for her advice and technical help, Dr. Hesham El-Rewini, Dr. Ian Gladwell, Dr. Peter Seidel and Dr. Jeff Tian for valuable insights.

I am very grateful to the faculty, staff and students of the Department of Computer Science for their help and support.

Special thanks to Rada, Roxana, Marius, Mihai and Finley, Paul, Razvan.

The following people helped improving the English in this dissertation: Love, Marilyn, Jason, Mark, Ryan and Troy.
Logic Forms for WordNet Glosses

Advisor: Professor Dan Moldovan
Doctor of Philosophy conferred May 18, 2002
Dissertation completed May 10, 2002

The need for performant information extraction techniques is continuously increasing, especially with the explosion of publicly available information on the Internet. Modern information extraction techniques are using more and more world knowledge but the derivation of knowledge from current resources turns out to be one of the toughest problems for the Natural Language Processing (NLP) community. A possible source of knowledge are machine readable dictionaries, if represented in a way suitable for knowledge derivation.

This dissertation shows how WordNet, a lexico-semantic machine readable dictionary, can be used as a source of information, by transforming concept definitions into Logical Forms (LFs). We prove the usefulness of our approach by showing how the LF notation can be used to boost up the performance of a Question Answering (QA) system. We present the logic notation and detail on the methodology used to create LFs from WordNet definitions.

We first show that LF is the most comprehensive representation available and introduce a few extensions to the original notation for the case of postmodifiers, comparatives and relative adverbs.

Given the fact that LFs are derived from the output of a parser we need high precision in part of speech tagging and parsing. We use a two level voting scheme between two state-of-the-art taggers at the first level, respectively between one of the taggers and WordNet syntactic categories. An important contribution of this dissertation is an improvement of almost 3% in performance achieved on tagging WordNet glosses compared to results obtained using state of the art taggers.
Additionally, we experiment two combination techniques to improve gloss parsing: constituent voting and parser switching. The best results are obtained for constituent voting using simple majority policy, respectively for parser switching with nearest neighbor techniques.

The main contribution of the dissertation is a method for deriving logic forms that combines a set of highly precise rules (transformations) with high recall heuristics. This method is scalable, flexible and easy to develop and maintain. LF transformations (LFTs) that generate LFs given a parse are selected based on probabilistic principles, i.e. they appear frequently enough to justify their selection. For ambiguous transformations we use simple neural networks to assign syntactic roles of arguments based on few features such as voice and type of preposition. The dissertation provides a set of heuristics to solve uncovered cases, one for each type of argument: subject, direct object, indirect object, prepositional object, modifier. In addition, an original solution for parsing base noun phrases in the larger context of logic forms derivation is described.

Finally we demonstrate how the logic notation and WordNet glosses in LF can boost up the performance of a Question Answering system. There are two types of questions for which we provide better answer ranking: (1) all the keywords from question can be found in the answer and the syntactic relations in the question are preserved in the answer (2) some keyword concepts are missing but lexico-semantic chains can be established between these keywords and concepts in the answer.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>xii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>xiv</td>
</tr>
<tr>
<td>Chapter</td>
<td></td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1. Goals of this Dissertation</td>
<td>3</td>
</tr>
<tr>
<td>1.2. WordNet</td>
<td>4</td>
</tr>
<tr>
<td>1.2.1. WordNet 1.7</td>
<td>8</td>
</tr>
<tr>
<td>1.3. Extended WordNet</td>
<td>8</td>
</tr>
<tr>
<td>1.4. Logic Form Transformation Approach</td>
<td>9</td>
</tr>
<tr>
<td>1.5. Part of Speech Tagging</td>
<td>12</td>
</tr>
<tr>
<td>1.6. Parsing Glosses</td>
<td>13</td>
</tr>
<tr>
<td>1.7. Logic Form Transformation</td>
<td>14</td>
</tr>
<tr>
<td>1.8. Application in Question Answering</td>
<td>15</td>
</tr>
<tr>
<td>1.9. Related Work</td>
<td>16</td>
</tr>
<tr>
<td>1.9.1. Dictionary Parsing</td>
<td>16</td>
</tr>
<tr>
<td>1.9.2. Tagging</td>
<td>17</td>
</tr>
<tr>
<td>1.9.3. Logic Form Derivation</td>
<td>19</td>
</tr>
<tr>
<td>1.9.4. Lexical Chains</td>
<td>19</td>
</tr>
<tr>
<td>1.9.5. Question Answering</td>
<td>20</td>
</tr>
<tr>
<td>1.10. Dissertation Outline</td>
<td>20</td>
</tr>
<tr>
<td>2. LOGIC REPRESENTATION</td>
<td>22</td>
</tr>
<tr>
<td>2.1. Brief History of Knowledge Representation Approaches</td>
<td>22</td>
</tr>
</tbody>
</table>
2.1. 70s - Early Days ................................................. 22
2.1.2. 80s-90s ....................................................... 25
2.2. Concept-Predicate Representation .................................. 28
   2.2.1. Advantages of Concept-Predicate Representation .......... 28
2.3. Logic Form Notation ............................................... 29
   2.3.1. Logic Form Guidelines ..................................... 30
   2.3.2. Verb Predicates .......................................... 31
   2.3.3. Complements ............................................ 32
   2.3.4. Conjunctions ............................................ 33
   2.3.5. Prepositions ............................................. 34
   2.3.6. Complex Nominals ....................................... 34
   2.3.7. Comparatives ........................................... 35
   2.3.8. Postmodifiers ........................................... 37
   2.3.9. Possessive Pronouns ................................... 38
   2.3.10. Relative Adverbs ...................................... 39
2.4. Conclusions ...................................................... 40
3. PART OF SPEECH TAGGING AND PARSING OF GLOSSES ........... 43
   3.1. Tokenizing Glosses ......................................... 43
   3.2. Tagging Glosses ........................................... 44
      3.2.1. Tagging Technology .................................... 45
         3.2.1.1. Supervised vs Unsupervised ......................... 45
         3.2.1.2. Rule Based Tagging ................................. 45
         3.2.1.3. Stochastic Tagging ................................ 46
         3.2.1.4. Unknown Words .................................... 47
         3.2.1.5. State-of-the-art in POS tagging .................. 48

vii
3.2.2. The Voting Scheme ........................................... 48
  3.2.2.1. WordNet - The Third Tagging Resource .......... 50
3.2.3. Tagging Definitions ....................................... 50
3.2.4. Experiments and Results .................................. 51
  3.2.4.1. Tagging noun.artifact and verb.social ........... 51
  3.2.4.2. Results ................................................ 53
3.3. Parsing Glosses ................................................ 53
  3.3.1. Parsing Technology ...................................... 54
    3.3.1.1. Performance Measures ............................ 56
    3.3.1.2. Algorithm 3.1: EVALB Transformation ........ 57
  3.3.2. Single Parser Performance on Glosses ............... 59
    3.3.2.1. Gloss Expansion ................................. 60
  3.3.3. Parsers Combination .................................... 61
    3.3.3.1. Forming an Ensemble of Parsers ................. 62
    3.3.3.2. Bagging a Parser by Sentences .................. 63
    3.3.3.3. Individual Parsers .............................. 64
    3.3.3.4. Constituent Voting .............................. 68
    3.3.3.5. Parser Switching ............................... 70
    3.3.3.6. Nearest Neighbor ............................... 72
    3.3.3.7. BestPR .......................................... 74
3.4. Conclusions .................................................. 75
4. LOGIC FORM TRANSFORMATION .................................. 77
  4.1. Deriving Logic Forms ...................................... 77
    4.1.1. Tokenizing, Tagging and Parsing .................. 78
    4.1.2. Transforming the Parse Tree ....................... 79
4.1.3. Bottom-Up Strategy and the Derivation Engine ............... 79
4.2. Probabilistic Selection of Rules ................................. 83
  4.2.1. Detecting the Rules ....................................... 83
  4.2.2. Select more Valuable Rules ............................... 85
4.3. Experiments and Results ...................................... 86
  4.3.1. Observations on Verb Related Rules ...................... 92
  4.3.2. Verb Sequences ........................................... 93
  4.3.3. Coordinated Verbs ....................................... 93
4.4. Learn Ambiguous Rules ........................................ 97
  4.4.1. The Perceptron ............................................ 98
  4.4.2. A Perceptron for Modelling Syntactic Roles ............ 98
  4.4.3. Results .................................................... 101
  4.4.4. Comparison with other Methods ......................... 101
4.5. Scalability ..................................................... 103
  4.5.1. Performance Measures .................................... 103
  4.5.2. Approaches ............................................... 104
    4.5.2.1. Statistical approach ................................. 104
    4.5.2.2. Heuristic-enhanced Statistical Approach .......... 106
    4.5.2.3. Incremental approach ............................... 107
  4.5.3. Comparison ............................................... 109
4.6. Conclusions .................................................... 109
5. BRACKETING COMPOUND NOUNS .................................... 111
  5.1. Introduction ................................................. 111
    5.1.1. Detection Errors ....................................... 112
    5.1.2. Interpretation Errors .................................. 113
5.2. Bracketing Coordinated Compound Nouns ......................... 114
  5.2.1. Coordinated Compound Nouns Detection ..................... 114
    5.2.1.1. WN-based Named Entity .............................. 115
    5.2.1.2. Compound Concept Detection ......................... 115
  5.2.2. Coordinated Compound Nouns Interpretation ............... 116
  5.2.3. Modifiers ................................................. 117
  5.2.4. Experiments and Results .................................. 119

5.3. Conclusions .................................................... 121

6. APPLICATION OF LOGIC FORMED GLOSSES IN QUESTION ANSWERING ........................................ 122
  6.1. Question Answering System ................................ 122
  6.2. From Logic Forms to Axioms ................................ 123
  6.3. Answer Extraction Procedure ................................ 124
    6.3.1. Transform Questions and Answers in Logic Forms .... 125
    6.3.2. Form Lexical Chains between Pairs of Concepts ....... 125
    6.3.3. Apply Unification on Lexical Chains .................... 126
    6.3.4. Extract Inferences to Provide Explanation ............ 127
      6.3.4.1. Hyper-inference .................................. 127
      6.3.4.2. Axiom-inference .................................... 127
    6.3.5. Integration of Axioms into the Input Context ........... 128
  6.4. Examples ....................................................... 129
    6.4.1. Example 1 .............................................. 129
    6.4.2. Example 2 .............................................. 130
  6.5. Q/A Results .................................................. 132
  6.6. Conclusions .................................................. 134
7. CONCLUSIONS ................................................................. 138

7.1. Summary ................................................................. 138

7.2. Goals Revisited ......................................................... 139

7.2.1. Concept Predicate Representation .............................. 139

7.2.2. High Performance Solution for Logic Form Derivation ...... 140

7.2.3. Boosting Performance of Tagging and Parsing .............. 141

7.2.4. Impact of WordNet Glosses on Question Answering ....... 141

7.3. Future Work ............................................................... 142

7.3.1. Extend Analysis To Other Parts of Speech ................... 142

7.3.2. Issues Related to Automatically Transform WordNet into a Knowledge Base ................................................. 143

7.3.3. Lexical Chains with More Types of Relations .............. 143

7.3.4. Implement Prover ...................................................... 144

7.3.5. Impact of Prover on Question Answering Performance ..... 144

7.4. Closing Words ............................................................ 144

REFERENCES ................................................................. 152
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1. The {limb} synset of WordNet and its related synsets</td>
<td>7</td>
</tr>
<tr>
<td>1.2. The {limb} synset of Extended WordNet and its related synsets</td>
<td>9</td>
</tr>
<tr>
<td>1.3. Parse tree for gloss of \textit{abbey}:n#1:{a monastery ruled by an abbot}.</td>
<td>11</td>
</tr>
<tr>
<td>1.4. The architecture of a Logic Form Transformer</td>
<td>11</td>
</tr>
<tr>
<td>2.1. Parse tree for gloss of \textit{tower}:n#1: (a structure taller than its diameter)</td>
<td>36</td>
</tr>
<tr>
<td>2.2. Parse tree for gloss of \textit{achromatic lens}:n#1</td>
<td>37</td>
</tr>
<tr>
<td>2.3. Parse tree for gloss of \textit{airdock}:n#1</td>
<td>40</td>
</tr>
<tr>
<td>2.4. Milestones in Knowledge Representation</td>
<td>41</td>
</tr>
<tr>
<td>3.1. The architecture of a Logic Form Transformer</td>
<td>44</td>
</tr>
<tr>
<td>3.2. Overview of the two level voting scheme</td>
<td>49</td>
</tr>
<tr>
<td>3.3. Parsing in the architecture of a Logic Form Transformer</td>
<td>54</td>
</tr>
<tr>
<td>3.4. Interpretations for \textit{I saw the man in the park with a telescope}</td>
<td>55</td>
</tr>
<tr>
<td>3.5. Parser performance as a function of training size</td>
<td>66</td>
</tr>
<tr>
<td>3.6. Number of bigrams vs Training Size (on WSJ)</td>
<td>67</td>
</tr>
<tr>
<td>3.7. The \textit{constituents voting} method</td>
<td>68</td>
</tr>
<tr>
<td>3.8. Parse Tree to constituents mapping</td>
<td>69</td>
</tr>
<tr>
<td>3.9. The \textit{parser switching} method</td>
<td>71</td>
</tr>
<tr>
<td>4.1. Transformer in the architecture of a Logic Form Transformer</td>
<td>78</td>
</tr>
<tr>
<td>4.2. An example of actions at leaves level</td>
<td>80</td>
</tr>
<tr>
<td>4.3. Up-Propagation highlighted</td>
<td>81</td>
</tr>
</tbody>
</table>
4.4. Horizontal Propagation Highlighted ........................................ 82
4.5. Transforming a coordinated NP into simpler NPs ..................... 90
4.6. Examples of POS tag transformations in the parse tree .............. 91
4.7. Parse tree for gloss of \{go, proceed, move\} .......................... 94
4.8. A perceptron ................................................................. 98
4.9. Parse tree for gloss of pneumatic.hammer:n#1 ....................... 99
4.10. Parse tree for gloss of air.intake:n#1 ................................ 100
4.11. A perceptron with two inputs ......................................... 100
4.12. The decision tree equivalent with the two inputs perceptron ...... 102
4.13. Performance of Statistical Approach ................................. 105
4.14. Number of rules necessary with an incremental approach ......... 108
5.1. Coordinated Compound Nouns and their Logic Forms.................. 112
6.1. Falcon - A System Architecture Overview ............................ 123
6.2. A partial illustration of chains and paths for question 045 ......... 131
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1. WordNet synsets for word form <em>chocolate</em></td>
<td>5</td>
</tr>
<tr>
<td>1.2. WordNet 1.6 Overview</td>
<td>5</td>
</tr>
<tr>
<td>1.3. WordNet relations</td>
<td>6</td>
</tr>
<tr>
<td>1.4. WordNet 1.7 Overview</td>
<td>8</td>
</tr>
<tr>
<td>1.5. Examples of LFT rules</td>
<td>10</td>
</tr>
<tr>
<td>2.1. Examples of conjunction predicates</td>
<td>33</td>
</tr>
<tr>
<td>2.2. Examples of preposition predicates</td>
<td>34</td>
</tr>
<tr>
<td>2.3. Examples of complex nominals predicates</td>
<td>35</td>
</tr>
<tr>
<td>2.4. Examples of comparative predicates</td>
<td>36</td>
</tr>
<tr>
<td>2.5. Examples of postmodifier predicates</td>
<td>38</td>
</tr>
<tr>
<td>2.6. Examples of relative adverb predicates</td>
<td>39</td>
</tr>
<tr>
<td>3.1. Taggers accuracy on <em>noun.artifact</em> and <em>verb.social</em></td>
<td>51</td>
</tr>
<tr>
<td>3.2. Brill’s and MXPOST taggers agreement</td>
<td>53</td>
</tr>
<tr>
<td>3.3. Results for constituent voting using five parsers</td>
<td>70</td>
</tr>
<tr>
<td>3.4. Limits and baseline for parser switching</td>
<td>72</td>
</tr>
<tr>
<td>3.5. Results for parser switching: nearest neighbor and bestPR</td>
<td>75</td>
</tr>
<tr>
<td>4.1. Examples of LFT rules</td>
<td>78</td>
</tr>
<tr>
<td>4.2. Size of grammar for WordNet glosses per part of speech</td>
<td>83</td>
</tr>
<tr>
<td>4.3. Top 10 most frequent rules in 10,000 noun glosses</td>
<td>84</td>
</tr>
<tr>
<td>4.4. Examples of LFT for determined and undetermined NPs</td>
<td>86</td>
</tr>
</tbody>
</table>
4.5. Examples of rules with prenominal modifiers .......................... 87
4.6. Coordinated base NPs (without determiners and plurals) ............... 88
4.7. Examples of rules having prenominal modifiers .......................... 90
4.8. Postmodifiers in noun.artifact hierarchy ............................... 93
4.9. Results on four grammar rules ........................................... 101
4.10. Performance of Statistical Approach ................................. 105
4.11. Number of rules in an increasing number of glosses ................. 107
4.12. Comparison of main approaches for 4,000 glosses .................... 109
  5.1. Alternative bracketing solutions and their logic forms ................. 114
  5.2. Distribution of main source of errors at detection .................. 119
  5.3. Performance of the proposed heuristics ............................. 120
  6.1. Axioms extracted from the gloss of verb kill:v#1 .................... 124
  6.2. Axioms extracted from the gloss of adjective Colombian:a#1 ....... 125
  6.3. QLF and ALF as the prove for question Q045 proceeds ............... 130
  6.4. Experiments with TREC questions and comparison with Falcon ...... 133
  6.5. Experiments with TREC questions and comparison with Falcon (cont’d) 134
  6.6. Argument binding for some Question-Answer pairs .................... 135
  6.7. Argument binding for some Question-Answer pairs (cont’d) .......... 136
  6.8. Statistics for questions that benefit from WordNet ................. 137

xv
To My Grandparents and My Parents
Chapter 1

INTRODUCTION

The expansion of the Internet over the last decade has led to an explosion of information available in electronic format. Consequently, the need for powerful information extraction systems is greater than ever. Modern information extraction techniques rely more and more on world knowledge and the problem becomes how and from what sources this knowledge can be obtained.

Consider question 471 from the ninth Text REtrieval Conference (TREC-9) Question Answering competition \(^1\)\cite{62}:

\textit{What year did Hitler die?}

The answer to this question was “Hitler committed suicide in 1945”. To justify that this is a plausible answer one needs extra knowledge. From the WordNet \cite{43} gloss of \textit{suicide:n}\#1 (first sense of \textit{suicide}) we have \{\textit{killing yourself}\} and from the first sense of verb \textit{kill:\#1} we have \{\textit{cause to die}\} which provides the justification required. However, to \textit{automatically} derive a justification, the initial question, answer and WordNet glosses must be transformed into a computational representation and automated inference procedures must be defined.

\footnote{The National Institute for Standards and Technology (NIST), an agency of the U.S. Department of Commerce’s Technology Administration initiated the Text REtrieval Conference (TREC) series \ldots to encourage research on information retrieval from large collections of data \ldots. A special track of the TREC series, starting with TREC-8, was the Question-Answering track, a track designed to take a step closer to information retrieval rather than document retrieval. Participants are given a large document set and a set of questions. For each question, the system returns a text snippet containing the answer and a document ID that supports the answer.}
This dissertation describes a methodology to transform WordNet glosses into a knowledge representation called the Logic Form (LF), and further into axioms that may enable reasoning mechanisms.

LF is first order logic and syntactically simple. It was first introduced by Hobbs [32] and enhanced with semantic and syntactic information by Harabagiu and Moldovan [25]. It has two main advantages:

- it uses concept predicates, resulting in a less ambiguous representation and
- it has positional syntactic arguments that ease other NLP tasks such as textual inference.

A predicate is generated for every noun, verb, adjective or adverb encountered in any gloss. The name of the predicate is a concatenation of the word’s base form, the part-of-speech and the WordNet semantic sense, thus capturing the full lexical and semantic information. Arguments for predicates are of two types: \( e \) - for events specified by action verbs, respectively \( x \) - for entities.

The argument’s position is also important as it encodes syntactic information. For instance, the second argument for a verb is the syntactic subject, the third is the direct object and the fourth is the indirect object.

The logic form for the previous question and answer are

Q: Hitlern#1(x1) & die:vn#1(e1’,x1’) & in(e1’,x2’) & TIME_STAMP(x2’)  
A: Hitlern#1(x1) & commit:vn#1(e1,x1,x2) & suiciden#1(x2) & in(e1,x3) & TIME_STAMP(x3) and for the glosses we have kill:n#1(e1,x1,x1) respectively cause:n#1(e1,x1,e2) & die:vn#1(e2,x1).

The axioms corresponding to the two glosses are

\( \text{suicide:n#1(x1)} \rightarrow \text{kill:n#1(e1,x2,x2)} \)

respectively

\( \text{kill:vn#1(e1,x1,x2)} \rightarrow \text{cause:n#1(e2,x1,e2)} \) & \( \text{die:vn#1(e2,x2)} \).

The question logic form is enhanced with the question answer type TIME STAMP. The first axiom is specific to a nominalisation: a noun on the left hand side (suicide) leads to a verb on the right hand side (kill). The second axiom exhibits
syntactic functional changes for x2 which is the direct object for kill and becomes subject for die. Using the axioms, the answer logic form is logically transformed into:
A: Hitler:n#1(x1) & commit:v#1(e1,x1,e2) & kill:v#1(e2,x1,x1) & in(e2,x3) & TIME_STAMP(x3)
and then into:
A: Hitler:n#1(x1) & commit:v#1(e1,x1,e2) & cause:v#1(e2,x1,e3) & die:v#1(e3,x1) & in(e3,x3) & TIME_STAMP(x3)

Every relation in which a substituted predicate is involved is replicated: for example TIME_STAMP(x3) is attached to the newly introduced event e3 (not all replicas are shown for simplicity of exhibition). At this point all question predicates can be unified with answer predicates and their arguments can be easily bound to arguments from the answer logic form. As the predicates in the answer logic form are all considered to be true one infers that the question logic form is also true.

1.1. Goals of this Dissertation

The main goal of this dissertation is to demonstrate the feasibility of transforming textual English into first order logic forms enhanced with syntactic information. When this kind of transformation is applied to a general purpose ontology like WordNet such an attempt can be viewed as a first step toward automated construction of generally applicable knowledge bases.

The more specific goals of the dissertation are outlined below with the reminder that they will be reviewed in the final chapter Conclusions.

- Show that concept based representation is an emerging formalism that combines the features of semantic nets and natural based representations AND enhances the original notation for underspecified or uncovered cases. Chapter 2 of the dissertation focuses on the overall problem of choosing the best representation formalism to represent WordNet glosses. After a brief overview of the evolution of knowledge representation over the last few decades, we argue how the representation to be chosen and provide our own contributions to
the analysis of the applicability of that knowledge representation to WordNet
glosses.

- **Provide a high performance solution to derive logic forms for WordNet glosses.** The approach we use to derive logic forms takes advantage of structural information provided by a syntactic parser. On top of the parser we select and develop a set of derivation rules that assign syntactic functions such as syntactic subject, direct object, indirect object, prepositional object, modifiers to predicates’ arguments. By analyzing tens of thousands of parse trees for glosses we might be able to observe some structural patterns in the trees that could lead to highly accurate logic forms with minimal implementation efforts. Chapter 4 addresses this issue.

- **Prove that tagging and parsing combination techniques are fruitful when applied to WordNet glosses.** As subtasks of the derivation process, part of speech tagging and syntactic parsing are two main issues that we address and try to boost their performance above state of the art results. The methodology to attain this goal is explained in Chapter 3.

- **Demonstrate the applicability of glosses in logic forms to real applications such as Question Answering.** There are two types of questions for which knowledge intensive solutions are required: (1) all the keywords from the question can be found in the answer and the syntactic relations in the question are preserved in the answer and (2) some keyword concepts are missing but lexico-semantic chains in some ontology (such as WordNet) can be established between these keywords and concepts in the answer. Automated solutions need a computational representation of knowledge and Chapter 6 illustrates how the information from glosses in logic forms are suitable for this task.

### 1.2. WordNet

WordNet is an electronic lexical database developed at Princeton University based on linguistic principles. It is divided into four data files containing data for
nouns, verbs, adjectives and adverbs. In WordNet the basic unit is a *synset* - a set of synonymous words which refer to a common semantic concept. Words may be formed in more than one synset: the noun *chocolate* belongs to three different synsets as given in Table 1.1. The first sense of each word in WordNet is the most frequent sense.

<table>
<thead>
<tr>
<th>Synset</th>
<th>Gloss</th>
</tr>
</thead>
<tbody>
<tr>
<td>{cocoa, chocolate hot chocolate}</td>
<td>(made from baking chocolate or cocoa powder and milk and sugar)</td>
</tr>
<tr>
<td>{chocolate}</td>
<td>(made from roasted ground cacao beans)</td>
</tr>
<tr>
<td>{chocolate, coffee, deep brown umber, burnt umber}</td>
<td>(a medium to dark brown color)</td>
</tr>
</tbody>
</table>

A small textual definition is attached to each synset via a *gloss* relation. An example is also included in the gloss for the large majority of synsets. Table 1.1 shows a few synsets with their corresponding glosses.

WordNet 1.6 contains 99,642 synsets and 121,962 unique words. The number of words, synsets and senses per part of speech are shown in Table 1.2.

<table>
<thead>
<tr>
<th>POS</th>
<th>Unique Strings</th>
<th>Synsets</th>
<th>Total Senses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noun</td>
<td>94474</td>
<td>66025</td>
<td>116317</td>
</tr>
<tr>
<td>Verb</td>
<td>10319</td>
<td>12127</td>
<td>22066</td>
</tr>
<tr>
<td>Adjective</td>
<td>20170</td>
<td>17915</td>
<td>29881</td>
</tr>
<tr>
<td>Adverb</td>
<td>4546</td>
<td>3575</td>
<td>5677</td>
</tr>
<tr>
<td>Totals</td>
<td>121962</td>
<td>99642</td>
<td>173941</td>
</tr>
</tbody>
</table>

A list of pointers is attached to each synset and these pointers express relations between synsets (see Table 1.3 for a list of typical WordNet relations).
The most important relation is the hypernymy relation. Noun and verb synsets are hierarchically organized based on the hypernymy relation.

For example the synset \{professor\} is a hyponym of the synset \{academician, academic, faculty member\}. In other words, there is a hypernymy relation from the former synset pointing to the latter, and a hyponymy relation in the form of a pointer from the latter synset to the former.

<table>
<thead>
<tr>
<th>WordNet Relation</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypernymy</td>
<td>equivalent with ISA</td>
</tr>
<tr>
<td>Hyponymy</td>
<td>reverse ISA</td>
</tr>
<tr>
<td>Synonymy</td>
<td>implicitly available among words from same synset</td>
</tr>
<tr>
<td>Antonymy</td>
<td>leads to a synset with opposite meaning</td>
</tr>
<tr>
<td>Gloss</td>
<td>points to the gloss</td>
</tr>
<tr>
<td>Meronymy</td>
<td>points to the whole in a part-whole relation</td>
</tr>
<tr>
<td>Holonymy</td>
<td>points to the part in a part-whole relation</td>
</tr>
<tr>
<td>Attribute</td>
<td>points to/from an attribute</td>
</tr>
<tr>
<td>Pertain</td>
<td>morphologically based relation between a noun and an adjective or an adjective and adverb</td>
</tr>
<tr>
<td>Cause</td>
<td>points to the cause of another action</td>
</tr>
<tr>
<td>Entailment</td>
<td>points to the implication of another action</td>
</tr>
</tbody>
</table>

Adjectives and adverbs are organized in clusters based on similarity and antonymy relations.

There are no links among synsets belonging to different parts of speech. Figure 1.2 illustrates the synset \{limb\} together with synsets to which it relates via labeled relations.

The usage of WordNet covers a large spectrum of applications from word sense disambiguation to search engines:
- information extraction [69]
- information retrieval [41]
Figure 1.1. The \{\text{limb}\} synset of WordNet and its related synsets

- question answering [51]
- word sense disambiguation [42]
- text inference [26]
- coreference, coherence and metonymy [21]
- knowledge acquisition [28] [46]
- internet search engines [47]

Despite the fact that WordNet has not been developed to be used as a knowledge base many researchers tried to use it this way and many weaknesses of WordNet when used in knowledge based tasks were pinpointed. The most important of the lacking features are given here:

- The lack of connections between different parts of speech hierarchies
- Limited number of connections between topically related words
- The lack of morphological relations
- The absence of case relations and selectional restrictions
- Some concepts (word senses) and relations are missing
• Some glosses are empty
• Lack of uniformity and consistency in the definitions

1.2.1. WordNet 1.7

WordNet is an ongoing project and new, improved releases are provided every few years. The most recent version of WordNet, 1.7 has more synsets and additional relations.

Table 1.4. WordNet 1.7 Overview

<table>
<thead>
<tr>
<th>POS</th>
<th>Unique Strings</th>
<th>Synsets</th>
<th>Total Senses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noun</td>
<td>107930</td>
<td>74488</td>
<td>132407</td>
</tr>
<tr>
<td>Verb</td>
<td>10806</td>
<td>12754</td>
<td>23255</td>
</tr>
<tr>
<td>Adjective</td>
<td>21365</td>
<td>18523</td>
<td>31077</td>
</tr>
<tr>
<td>Adverb</td>
<td>4583</td>
<td>3612</td>
<td>5721</td>
</tr>
<tr>
<td>Totals</td>
<td>144684</td>
<td>109377</td>
<td>192460</td>
</tr>
</tbody>
</table>

All our work in this dissertation was done using WordNet 1.6 except the coordinated compound nouns derivation in *Chapter 5* which used the latest release WordNet 1.7.

1.3. Extended WordNet

The purpose of the Extended WordNet [25] project is to eliminate most of the weaknesses of WordNet. The main idea is to take advantage of the rich information available in WordNet glosses, particularly in the textual definition attached to the synsets. It aims to semantically tag and parse the glosses, transform them into logic forms and provide topical relations among synsets.

Extended WordNet will increase the connectivity among synsets by at least one order of magnitude. The {limb} synset, previously discussed, will be linked in Extended WordNet to synsets from its gloss (some belonging to a different part of speech): jointed:a#1, appendage:n#1, animal:n#1, used:v#1, locomotion:n#1,
Figure 1.2. The \{limb\} synset of Extended WordNet and its related synsets

grasping:n\#2, arm:n\#1, leg:n\#2, wing:n\#2, flipper:n\#2 (see Figure 1.3). This constitutes a qualitative leap in WordNet’s organization.

1.4. Logic Form Transformation Approach

Our approach for transforming WordNet glosses into logic form is to follow the parser. For each grammar rule obtained from the output of the parser, there are one or more transformation rules which produce the corresponding logic formula. There are two classes of rules: (1) intra-phrase and (2) inter-phrase transformation rules. The intra-phrase transformation rules generate predicates for every noun, verb, adjective or adverb. They also assign the variables that describe dependencies local to the phrase. The inter-phrase transformation rules provide the arguments of the verb predicates, preposition predicates and inter-phrasal conjunctions. Verb predicate arguments are identified by recognizing the syntactic subject and object of the respective verb, based on a few grammar rules and relative pronoun interpretation. Dependencies between adjectival (adverbial) phrases and noun (verb) phrases are
predicted based on vicinity. Both intra- and inter-phrase transformation rules are produced from the parser.

### Table 1.5. Examples of LFT rules

<table>
<thead>
<tr>
<th>Grammar Rule</th>
<th>Transformation Rule (LFT)</th>
<th>Phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP → DT NN</td>
<td>noun/NN → noun(x₁)</td>
<td>(NP (a/DT monastery/NN))</td>
</tr>
<tr>
<td>NP → DT JJ NN</td>
<td>adj/JJ noun/NN → noun(x₁) &amp; adj(x₁)</td>
<td>(NP (a/DT short/JJ sleep/NN))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grammar rule</th>
<th>Transformation Rule (LFT)</th>
<th>Phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>PP → IN NP</td>
<td>prep/IN noun/NP(x) → prep(x, e) &amp; noun(x)</td>
<td>(PP (by/IN (NP an abbot))</td>
</tr>
<tr>
<td>VP → VP PP</td>
<td>verb/VP-PASS by/PP(x) → verb(e, x, e) &amp; by (e, x)</td>
<td>(VP (ruled/VBN by/PP))</td>
</tr>
<tr>
<td>NP → NP VP</td>
<td>noun/NP(x) verb/VP-PASS(e, x₁, x₂) → noun(x₂) &amp; verb(e, x₁, x₂)</td>
<td>(NP (NP ... )</td>
</tr>
</tbody>
</table>

Consider the gloss of the WordNet concept of abbey:n#3: \{a monastery ruled by an abbot\}. The parse tree for this gloss is given in Figure 1.4.

The logic form corresponding to this tree is: \{monastery:n(x₁) & rule:v(e₁, x₂, x₁) & by(e₁, x₂) & abbot(x₂):n \}. Table 1.5 illustrates examples of transformation rules that make the derivation of LF possible.

We are faced with several problems: (1) POS tagging accuracy that directly influences the accuracy of the parser and consequently the precision in deriving logic forms, (2) parsing accuracy and (3) selection of logic form transformations.

The architecture of the system is given in Figure 1.4. The first module, Pre-processing, extracts definitions from WordNet glosses and tokenizes them based on Treebank specifications.

Part of Speech tagging assigns POS tags to each word in gloss. This step is required for parsing, which is the third module. The task of parsing is to provide a phrase structure for a given gloss. The phrase structure indicates the main entities...
a monastery ruled by an abbot

in the gloss and establishes hierarchical relations among them. From this phrasal structure it is easier to derive syntactic relations such subject.

Figure 1.3. Parse tree for gloss of abbey:m#1: {a monastery ruled by an abbot}.

Figure 1.4. The architecture of a Logic Form Transformer

From this structural representation in the form of a tree we extract Logic Form Transformations (LFT) that are applied in the Transformer module which also represents the final step in LF derivation of glosses. Each LFT is mapped into one or more interpretation procedures.
1.5. Part of Speech Tagging

The task of part of speech (POS) tagging is to assign a syntactic category such as NN (common noun), NNP (proper noun), NNS (common noun plural) to each encountered token in a sequence of words.

If we run the MXPOST [52] part of speech tagger on the gloss of abbey:n#3, it results:

MXPOST: a_DT monastery_NN ruled_VBN by_IN an_DT abbot_JJ

The tag of abbot is wrongly assigned by MXPOST. If we were to use this tagger, the parser would generate an incorrect parse tree for this gloss and consequently an incorrect logic form would be derived. A solution to improve the accuracy of assigned tags is necessary.

The state of the art in part of speech tagging is around 95% accuracy for rule-based part of speech taggers [8] and stochastic taggers [52]. As pointed out in [52], the convergence of accuracy of different approaches to tagging can be explained by either that all techniques miss the correct predictors to cover the residue, or more likely that the performance of corpus based algorithms cannot be higher due to consistency problems.

Thus, instead of trying to improve the models that may lead to nothing due to inconsistencies, we use a voting scheme that is based on the output of two state of the art taggers to improve the performance. If the two taggers agree we assess that tag as being correct. For words that have different tags we ask the user to manually decide which one if any are correct.

If we reconsider the previous example and tag the gloss of abbey:n#3 using Brill’s tagger [8] and MXPOST tagger [52] we obtain the following tags:

Brill’s: a_DT monastery/NN ruled/VBN by/IN an_DT abbot/NN
MXPOST: a_DT monastery_NN ruled_VBN by_IN an_DT abbot_JJ

Except the tag for the last word, abbot, the two taggers agree. The tag for abbot is automatically assigned using a second voting scheme between Brill’s tagger
and WordNet tags: abbot is a noun in WordNet which leads to an agreement with ’NN’, the tag assigned by Brill’s tagger. The dissertation shows how a two-level voting can lead to better than state of the art results on glosses at the expense of user’s intervention on a small amount of data.

1.6. Parsing Glosses

Parsing is the task of delimiting phrases of a sentence and describing the relations between them. The parser is given an unmarked sentence and it is required to perform these annotations. We will guide our work based on Penn Treebank parsing style. In their annotation, which is a combination of many grammatical theories, chunks of text are delimited by brackets and identified by labels. The bracketings can be viewed as forming multipath trees with a unique path from each word to the root of the tree. Part of speech tags are the preterminal nodes in this tree, and every word has an associated part of speech tag. In parser evaluations, part of speech tagging is treated as a separate task, and subsequently those nodes are treated differently from the rest of the tree.

Due to the lack of a WordNet treebank the best approach to boost parsing accuracy on WordNet glosses would be to use combination methods. There are two major techniques for parser combination: constituent voting and parser switching. Before applying either of these techniques, we must solve the problem of obtaining an assembly of state-of-the-art parsers. There is a shortage of publicly available parsers. As an alternative solution we started from an in-house implementation of Collins’ statistical parser, built by Mihai Surdeanu, and trained the parser on different data.

The limitations of the statistical parser we use are mainly due to three different causes: (1) model errors (2) estimation errors (3) sparse data or insufficient training data. Modelling errors come from the inadequacies of the model. We often make the models weak or inaccurate because we know there is not enough data available to accurately estimate the parameters of a better model. Estimation errors come from
our lack of access to the true probabilities or parameters which shape out our model. We cannot utilize complex models without accurate probability estimates and we can only produce accurate estimates for small parameter spaces given our limited data, which turns to be the most desirable resource. Multiple parsers would have to be the result of a unified research effort in which the errors made by one parser are made a priority target for the developer of another parser. The combination techniques we develop lead to better than state-of-the-art results for gloss parsing. The main disadvantage of using constituent voting is the lack of confidence of the voted structure: some constituents may overlap and thus leading to a non-tree structure. Simple majority voting offers the best results when using constituent voting, respectively nearest neighbor offers the best results when parser switching is used.

1.7. Logic Form Transformation

As mentioned earlier we have one or more transformation rules for each grammar rule. Given the large number of grammar rules, it is almost impossible to consider all of them.

For example, the number of different rules extracted from the parse trees of the entire noun hierarchy in WordNet is 5,392. However, for the case of glosses for each grammar phrase there are several rules covering most of the cases.

For instance, for grammar rules with a nonterminal S on the left hand side, the top ten most frequent rules account for 99% of all occurrences of such rules, although there are 35 distinct rules on the 10,000 randomly selected noun glosses.

Based on the previous observation we develop a method based on probabilistic principles for the selection of most valuable rules, i.e. we consider those rules that appear in a sufficient large number of glosses to justify their selection. This way we select 70 rules that offer a performance of more than 80%. After applying those rules some glosses may have unfilled arguments. To boost the performance we develop a set of heuristics for each type of missing argument. A heuristic was designed for each type of argument.
• Subject - previous phrase head argument. If verb is passive, then use the prepositional object argument of the following by preposition
• Direct object - first following phrase head argument (or second for ditransitives) or surface subject if the verb is in passive
• Indirect object - second following noun phrase head argument (or first) for ditransitives
• prepositional head - previous phrase head argument
• prepositional argument - following phrase head argument
• adjective/adverbs - following noun/verb phrase head argument or previous noun/verb phrase argument if there is none following
• default - generate a new argument that does not exist.

This method leads to a performance of 89% on WordNet noun glosses.

1.8. Application in Question Answering

The logic form notation can be used in Question Answering systems to boost their performance. There are two types of questions for which answer ranking can be improved: (1) all the keywords from the question are found in the answer and (2) some keywords are missing but lexical paths can be established between missing keyword concepts and concepts in the answer using lexico-semantic relations in WordNet.

For the first type of question, we check whether the syntactic relations among keywords in the question are preserved in the answer. The answer that has the closest such syntactic relation preservation is boosted to the first place. The second type of question requires more processing to improve performance. To be useful into a Question Answering system a pair synset-definition is transformed in one or more axioms. Question and answer are manually disambiguated by attaching WordNet synsets to each content word. Paths between pairs of concepts, one from the question and one from the answer, are retrieved following hypernymy relations, respectively using axioms. From the many paths that are found we pick only those that lead to a
successful proof, which implies successful predicate matching and successful argument bounding.

In TREC-8&9 Question Answering tracks 893 test questions were provided: the
answer for 542 of them contained the selected keyword concepts from the question and
13 out of 542 fall into our first type of questions; from the remaining 351 questions
for 52 one can establish lexical paths in WordNet from answer concepts to question
concepts and thus falling into our second type of questions.

Consider question 45 from TREC-8 test set of questions for the Q/A track:

*Q045: When did Lucelly Garcia, former ambassador of Colombia to Honduras, die?*

The answer is found in “Several gunmen on a highway leading to the Colombian city
of Ibagué murdered Colombian Ambassador to Honduras Lucelly Garcia today”. Since
die:#1 is not present, the answer needs an explanation. There are many lexical chains
in WordNet between die:#1 and murder:#1 but there is only one that successfully
allows for a proof and this is murder:#1 hyper kill:#1 gloss: die: #1. Chapter 6 offers
details on how to select the right chain and how proofs should be obtained.

### 1.9. Related Work

Our work on processing WordNet glosses’ definitions resembles that of parsing
definitions of electronic versions of traditional dictionaries or machine readable dicionaries (MRD). The research in this area focused on extracting genus, unlabeled or
labeled relations, and building taxonomies.

#### 1.9.1. Dictionary Parsing

Early work on extracting information from MRD focused on identifying the
taxonomic nature of the genus terms in the *Merriam-Webster Pocket Dictionary* by
Amsler [1]. Chodorow [12] took further the previous work and aimed to extract taxon-
omy from the definitions. Chodorow noticed that definitions generally contain a
genus term and differentia that distinguishes the term being defined from other terms
belonging to the same genus. He developed a heuristic based pattern-matching algo-
rithm to identify the genus terms in definitions and structure them in a hierarchical taxonomy.

Other efforts focused on discovering relations among words. Semantic relatedness was measured through a comparison of the networked contexts surrounding words [66]. Labeled relations, while being more difficult to obtain, provide greater power for handling many aspects of NLP. Richardson [54] uses a broad coverage parser to extract labeled relations from LDOCE (Longman Dictionary of Contemporary English) and shows how they can be helpful in attaching ambiguous prepositional phrases by specifying the relation that holds between the object of the preposition and the constituent being modified by the prepositional phrase. Vanderwende [63][64][65] describes in detail how to extract semantic relations comprising MindNet [54]. A broad-coverage parser is essential to this process.

The main objective of the Dictionary Parsing Project at Information Sciences Institute [35] is to identify semantic relations in the definitions of 1913 Webster’s 2nd International Dictionary (W2) and build a semantic network, including an ontology. They use Brill’s tagger and a broad coverage parser to preprocess the definitions. The relations are extracted using a heuristic-based approach.

1.9.2. Tagging

There are many approaches to part of speech tagging. If one assigns the most frequent tag (as computed from tagged corpora) to each word, a baseline of 92% is obtained as reported in [8]. However to boost up the performance in the high 90s is quite challenging.

**Rule Based Tagging**

Typical rule based approaches [8] use contextual information to assign tags to unknown or ambiguous words. These rules are often known as context frame rules. As an example, a context frame rule might say something like: If an ambiguous/unknown word X is preceded by a determiner and followed by a noun, tag it as an adjective. In addition to contextual information, many taggers use morphological information to
aid in the disambiguation process. One such rule might be: if an ambiguous/unknown word ends in an -ing and is preceded by a verb, label it a verb (depending on your theory of grammar, of course). Rule based taggers most commonly require supervised training; but, very recently there has been a great deal of interest in automatic induction of rules. One approach to automatic rule induction is to run an untagged text through a tagger and see how it performs. A human then goes through the output of this first phase and corrects any erroneously tagged words. The properly tagged text is then submitted to the tagger, which learns correction rules by comparing the two sets of data. Several iterations of this process are sometimes necessary.

Stochastic Tagging

The term stochastic tagger may refer to any number of different approaches to the problem of POS tagging. Any model that somehow incorporates frequencies or probabilities, i.e. statistics, may be properly labeled stochastic. Example of stochastic taggers are given in [70], [52], [40], [37]. The simplest stochastic taggers disambiguate words based solely on the probability that a word occurs with a particular tag. In other words, the tag encountered most frequently in the training set is the one assigned to an ambiguous instance of that word.

The problem with this approach is that while it may yield a valid tag for a given word, it can also yield inadmissible sequences of tags. An alternative to the word frequency approach is to calculate the probability of a given sequence of tags occurring. This is sometimes referred to as the n-gram approach, referring to the fact that the best tag for a given word is determined by the probability that it occurs with the n previous tags. The most common algorithm for implementing an n-gram approach is known as the Viterbi Algorithm, a search algorithm which avoids the polynomial expansion of a breadth first search by trimming the search tree at each level using the best N Maximum Likelihood Estimates (where n represents the number of tags of the following word). The next level of complexity that can be introduced into a stochastic tagger combines the previous two approaches, using both tag sequence probabilities and word frequency measurements. This is known as a Hidden Markov
Model. The assumptions underlying this model are the following: each hidden tag state produces a word in the sentence. Each word is: (1) either uncorrelated with all the other words and their tags or (2) it is probabilistically depending on the previous $N$ tags only.

1.9.3. Logic Form Derivation

The derivation of logic forms using a rule based system was used in DIAGRAM [55]. DIAGRAM explicitly distinguishes four basic sentence types: imperative, declarative, propositional interrogative and argument interrogative. All four types are analyzed in terms of three basic functions: subject, predicate and an indicator of mood. DIAGRAM’s grammar was extended with the Linguistic String Project grammar [59] to form DIALOGIC, the syntactic analyser and semantic translator used in the TACITUS [34] project. DIALOGIC produced first-order logic representations for sentences, encoding everything that can be determined by purely syntactic means, without recourse to the context or to world knowledge. DIAGRAM was a pioneering project in the field of deriving first order logic representations from free text. However, they do not specify any performance value other than large coverage of English Grammar and thus is hard to compare it with our work.

1.9.4. Lexical Chains

Harabagiu and Moldovan in [26] use lexical chains among concepts to extract information unstated in a text but implied. Their approach for finding inferences is to search for semantic connections in WordNet that correspond to lexical relations between local contexts. A marker propagation mechanism is used to exploit the parallelism available in the process of path finding. Also, a number of filtering mechanisms based on lexico-syntactic relations are proposed to limit the number of paths along which the inferences are done.

Hirst [31] uses chains of semantically related words to detect malapropisms. A malapropism is a correctly spelled word that does not fit in the context where it is used.
because it is the result of a spelling error on a different word that was intended. Their algorithm to detect lexical chains uses WordNet to automatically quantify semantic relations between words. Chains identified by the algorithm may have two major problems: over- or under-chaining. Under-chaining - the inability to link a pair of related words - might be caused by an inadequacy of WordNet’s set of relations, a lack of connections in WordNet, a lack of consistency in the semantic proximity expressed by WordNet’s links, and a poor algorithm for chaining. Over-chaining - the linking of two poorly related words - might happen whenever two semantically distant words are close to each other in WordNet’s graph. Over-chaining often results in the merging of two chains.

Barzilay [2] uses lexical chains in texts for summarization. Summarization proceeds in three steps: the original text is first segmented, lexical chains are then constructed, strong chains are identified and significant sentences are extracted from text. This algorithm relies heavily on WordNet as a lexical knowledge base. It remarks that the main drawback of WordNet is the limited relations among syntactic categories (nouns are related to adjectives through *attribute* relations, adjectives refer to nouns with *attribute* and *pertain* relations, adverbs refer to adjectives with *pertain* relations).

1.9.5. Question Answering

In [24][50], a logic-based approach to Question Answering is presented. Both questions and answers are transformed into a loosely logic form in which relations are unlabeled. With the aid of Horn-like axioms and a backchaining prover questions are justified and their correctness assessed.

Vicedo [67] uses a WordNet-powered approach to question answering. He builds the context of the expected answer based on question type, definition terms - a definition term is a keyword from question that defines characteristics of the expected answer, and synonyms, one-level hyponyms and all hyperonyms of definition terms.
1.10. Dissertation Outline

Chapter 2 presents in detail the logic notation we use and a brief history of knowledge representation over the years. This chapter describes how nouns, verbs, modifiers, prepositions and conjunctions are transformed into predicates and how to fill their arguments. We propose few extension of the original notation for the cases of postmodifiers, comparatives and relative adverbs. The extensions are of such manner as to subscribe to the general principles of the original notation.

Chapter 3 discusses how tagging and parsing of WordNet glosses is performed. The voting scheme we developed uses Brill’s part of speech tagger, MXPOST tagger and WordNet tags to improve on accuracy of tagging glosses and to detect possible wrong tags. Additionally we apply combination techniques for parsing glosses with best results for simple majority constituents voting, respectively nearest neighbor techniques for parsing switching techniques.

Chapter 4 elaborates on a procedure for automatic selection of derivation rules based on stochastic principles: determine rules that provide the best coverage. In addition we develop a set of heuristic, one for each type of syntactic argument, to boost up the overall performance of derivation to 89%.

Chapter 5 treats the special case of coordinated compound nouns and presents a solution for bracketing them.

Chapter 6 presents an application of WordNet glosses in Logic Form to Question Answering. There are two types of questions whose answers can be better ranked using syntactic information coded in LF and WordNet glosses in logic form.

Chapter 7, Future Work, reviews the initial stated goals of the dissertation and offers future plans for extending the derivation of logic forms for glosses to other parts of speech (verbs, nouns, adjective and adverbs), to experiment with new methods for logic form derivation such as pattern matching - especially for glosses that have difficult structures (many coordination as in verb glosses), to fully implement the logic prover and to study the impact of WordNet axioms on Question Answering.
Chapter 2
LOGIC REPRESENTATION

This chapter presents the knowledge representation, called logic form, that we use and extend throughout this work.

In order to automatically process human knowledge, there is a need to find an appropriate computational representation and then to apply automated procedures to reason about it. In the early days the most successful approaches built systems around semantic networks enhanced with efficient reasoning capabilities attached to nodes in the network.

With the advent of TACITUS [34], there was a change of paradigm in knowledge representation proposed by Hobbs in [32], in that the representation was based on natural language and efficiencies were removed from the notation for the sake of simplicity.

The work of Hobbs was further improved by Harabagiu, Miller and Moldovan [25] who proposed a first order representation based on concept predicates, rather than on base form predicates, leading to a qualitative leap in the NL-KR field.

2.1. Brief History of Knowledge Representation Approaches

After a short overview of the evolution of knowledge representation systems over the years we will present the guidelines of logic form representation followed by few extensions for comparatives, postmodifiers and possessives that we propose.

2.1.1. 70s - Early Days

As we pointed out, during the early stages of knowledge based systems, the representation was built around semantic networks. A semantic network is a structure
for representing knowledge as a pattern of interconnected nodes and arcs [61]. The main elements of a semantic network are:

- Nodes in the net represent concepts of entities, attributes, events and states.
- Arcs in the net, usually called conceptual relations, represent relationships that hold between the concept nodes. Conceptual relations can represent linguistic cases, spatial, temporal, causal and logical connectives.
- Concept types are organized in a hierarchy according to level of generality, such as entity, living-thing, animal, carnivore, feline, cat. This hierarchy is often called a taxonomic hierarchy or subsumption hierarchy, since the instances of a general type such as animal subsumes the instances of a more specialized type such as cat.
- Relationships that hold for all concepts of a given type are inherited through the hierarchy by all subtypes.

The networks diverge on how to represent quantifiers and operators of logic or how to perform inferences. Some systems emphasize the ability to assert propositions and reason with them, and others place more emphasis on ways of defining new concepts in the type hierarchy.

**KL-ONE**

KL-ONE [6] is a language designed for the explicit representation of conceptual information. It is based on the idea of structured inheritance networks. It was designed for use in the construction of the knowledge base of a reasoning mechanism.

KL-ONE divides knowledge representation into two tasks: (1) assertion for making statements and (2) description for elaborating descriptions or terms. The desciptional part of KL-ONE allows someone to form a diversity of descriptive terms (compound description) out of other descriptive terms using some operators. The assertional part makes use of the terms from the descriptions to make statements about the world. The assertional capabilities in KL-ONE include statements of existence, description coreference, and identity of individual constants in a particular context. KL-ONE is an “object-oriented” language with the ability to attach procedures and
data to structures in the network, through interpretive hooks that specify the set of situations in which they are to be triggered.

**KRL**

KRL [5] is a Knowledge Representation Language that attempts to integrate procedural knowledge with a base described declaratively. The declarative knowledge is based on structured conceptual objects with associated descriptions.

Objects, relations, scenes and events are all examples of conceptual entities that can be associated with appropriate descriptions in KRL.

A description is fundamentally intensional - the structure of the description can be used in recognizing a conceptual entity and comparing it with others. These objects form a network of memory units with several different sorts of linkages.

Procedures can be associated directly with the internal structure of a conceptual object. This procedural attachment allows the steps for a particular operation to be determined by characteristics of the specific entities involved.

There are three underlying operations in the system: augmenting a description to incorporate new knowledge, matching two given descriptions to see if they are compatible and seeking referents for entities that match a specified description. KRL was designed for use in understander systems.

**LOOM**

LOOM is a knowledge representation system from KL-ONE family. LOOM, as with any KL-ONE-style system, divides the knowledge space into two partitions, called the terminological box and the assertional box, and using two distinct reasoners to carry out their inferences: terminological and assertional. It also had two additional partitions: the universal box and default box, each having its own associated component. A library of domain-specific reasoners complements the domain-independent reasoners.

A general characteristic of the KL family of knowledge representation systems is the fact that they embed reasoning mechanism into the net. Those mechanism lead to a very expensive approach for building knowledge bases.
2.1.2. 80s-90s

The 80s and 90s marked a change in the paradigm of knowledge representation. The change was to move away from the semantic network in the classical meaning of it as illustrated by the START system (presented in the next section) to a natural language based representation in the TACITUS project.

START

START is a Question Answering system developed at MIT by Katz [36]. The START natural language system (SynTactic Analysis using Reversible Transformations) consists of two modules which share the same grammar. The understanding module analyzes English text and produces a knowledge base which incorporates the information found in the text. Given an appropriate segment of the knowledge base, the generating module produces English sentences. A user can retrieve the information stored in the knowledge base by querying it in English. The system will then produce an English response.

START has been used by researchers at MIT and other universities and research laboratories to build and query knowledge bases using English. Given an English sentence containing various relative clauses, appositions, multiple levels of embedding, the START system first breaks it up into smaller units, called kernel sentences (usually containing one verb). Following a separate analysis of each kernel sentence, START rearranges the elements of all parse trees it constructs into a set of embedded representational structures. These structures are made up of a number of fields corresponding to various syntactic parameters in a sentence, but the three most salient parameters, the subject of a sentence, the object, and the relation between them are singled out as playing a special role in indexing. These parameters are explicitly represented in a discrimination network for efficient retrieval. As a result, all sentences analyzed by START are indexed as embedded ternary (or T) expressions (subject relation object). Certain other parameters (adjectives, possessive nouns, prepositional phrases, etc.) are used to create additional T-expressions in which prepositions and several special words may serve as relations.
For instance, the following simple sentence (1) “Bill surprised Hillary with his answer” produces two T-expressions: (2) \{\{Bill surprise Hillary\} with answer\} \{answer related-to Bill\}.

The remaining parameters - adverbs and their position, tense, auxiliaries, voice, negation, etc. - are recorded in a representational structure called a history. The history has a page pertaining to each sentence that yields the given T-expression. When the T-expressions are indexed in the knowledge base, START cross-references its three components and attaches the history to it. One can thus think of the resulting entry in the knowledge base as a “digested summary” of the syntactic structure of an English sentence.

In order to handle embedded sentences, START allows any T-expression to take another T-expression as its subject or object. START can analyze and generate sentences with arbitrarily complex embedded structures.

All previously mentioned systems (except START) attach procedures to nodes in an underlying semantic network. As Hobbs pointed out in [32] “decoupling the details of implementation, as efficiencies and procedures, from the representation itself would allow to focus on the real problems at hand: in natural language that is interpretation of discourse”. This philosophy was used in TACITUS [34].

\textit{Falcon}

Falcon [22] is a Question Answering system that uses a first order logic representation for representing both questions and answers.

The representation is very similar with the logic form with the difference that there are no labeled arguments. For example, there is no way to differentiate the subject from the direct object of a verb. This relaxation pays off in the derivation process. Given a correct parse tree, Falcon uses the deterministic rules to detect the heads of phrases and uses a set of up-propagation rules to link the predicates among them.

This representation has been successfully used in conjunction with sets of axioms to answer questions from large collections of text.
**TACITUS**

TACITUS was a project for interpreting text. Its underlying representation consists of first-order predicate calculus obtained through a large coverage syntactic and semantic translator DIALOGIC, which is an extension of DIAGRAM [55]. The main advantage of the representation was simplicity, allowing the researcher to focus on the real problems of text interpretation. TACITUS developed general procedures, together with the underlying theory, for using common sense and technical knowledge in the interpretation of written discourse. It includes five subareas:

- syntax and semantic translation
- commonsense knowledge
- domain knowledge
- deduction
- “local” pragmatics

DIALOGIC produces a logical form in first-order predicate calculus by encoding everything that can be determined by purely syntactic means, without recourse to context or world knowledge.

The advent of TACITUS [34] produced a change of paradigm in knowledge representation, which started to be visible in the START project proposed by Hobbs in [32], in that the representation was based on natural language and efficiencies were removed from the notation for the sake of simplicity.

The advantages of using natural language as a knowledge representation medium were highlighted in the Fall Symposium on “Knowledge Representation Systems based on Natural Language” [17] [18]:

- such systems are easy for people to use;
- most human knowledge is encoded and communicated via natural language;
- a system based on natural language can automatically create and update its knowledge base directly from natural language input;
- the representation is symbolic and uniform;
• same representational and inference mechanism could be used when utilizing previous knowledge for processing new natural language inputs;
• it is hard to match expressiveness and precision of NL, particularly in not formalized domains;
• many scientists believe that mental-level representation of knowledge is close in form to NL;

The work of Hobbs was further extended by Harabagiu and Moldovan [25] who proposed a first order representation based on concept predicates, rather than base form predicates, leading to a qualitative leap in the NL-KR field.

2.2. Concept-Predicate Representation

The concept predicate representation was proposed in [25] for the logic representation of WordNet glosses. It has several features that distinguish it from other representations:

• it has functional syntactic information embedded within in the form of positional arguments of verbs and prepositions;
• it has coarse categorial syntactic information corresponding to WordNet’s four parts of speech: noun, verb, adjective and adverb;
• it has semantic information in the form of WordNet senses leading to the concept predicates representation;

The concept based representation is less ambiguous than the base form based representation.

2.2.1. Advantages of Concept-Predicate Representation

In Hobbs’ representation [34], predicates are identical to natural language word base forms. This implies a lexical ambiguity at predicate level as in the following example.

*John bought a plant.*
The predicate plant(y) corresponding to the noun plant carries a lexical ambiguity in that the predicate may refer to the sense \{industrial plant\} denoted by plant1 or to the sense living organism denoted by plant2. In [33], the distinction is made by generating two predicates plant1(y) and plant2(y) with two corresponding axioms in the knowledge base that make the distinction between the two predicates. Generating those axioms in the knowledge base is a tremendous effort. Moreover, for each sentence that involves such a lexical ambiguous predicate, an additional effort will be needed to choose the right predicate at processing time. One can easily overcome this by adding the right WordNet sense in the predicate representation. In the case of plant example, if the semantics of the sentence is that “John bought a living organism” the corresponding predicate in our notation is plant:n#2(y) (second WordNet sense of noun plant). As one notices, both part of speech and sense number are required for unique identification of WordNet concepts in different applications that make use of the concept based representation. Thus, in LFT a predicate does not correspond to its base form only but to its WordNet concept. This brings dramatic advantages from a logical point of view, as we can exploit the general lexico-semantic relations available in WordNet among concepts. For example, one can easily subsume a concept predicate by its hypernym without the need of an explicit axiom in the knowledge base stating the subsumption. The subsumption is explicitly embedded in the WordNet hypernym relation.

We aim at transforming WordNet [43] glosses and English text into logic form representation. Using this representation we develop inference mechanisms that are independent of representation (WordNet relations are as general as possible) and can be applied in any scenario.

The next section describes in detail the principles standing behind the logic form representation. We focus on the syntactic information of predicates, i.e. subject, direct object and others. The implementation approach and details of how to derive logic forms are preserved for next two chapters.
2.3. Logic Form Notation

In LF the arguments of verb predicates are positional. The first argument states the eventuality of the event described by the verb, as Davidson proposed in [16] for action sentences. This allows for compositionality. The second position is the syntactic object, followed by direct object and indirect object. The positional representation of verb arguments is useful for text inference algorithms based on lexico-syntactic relations among WordNet concepts [26]. The positional representation is also simple, making automated processing easier.

The LFT codification acknowledges syntax-based relationships such as: (1) syntactic subjects, (2) syntactic objects, (3) prepositional attachments, (4) complex nominals, and (5) adjectival/adverbial adjuncts.

The logic format is preferred when it comes to reasoning and other logic manipulations in knowledge bases.

2.3.1. Logic Form Guidelines

There are two criteria guiding the design of logic form representation: (1) the notation should be as close as possible to English, and (2) the notation should be syntactically simple.

The approach is to derive the $LFT$ directly from the output of the syntactic parser. The parser resolves the structural and syntactic ambiguities, and the WSD provides semantics for the predicates. This way, we avoid the very hard problems of logic representation of natural language. We follow closely the successfully representation used by Hobbs in TACITUS [34]. Hobbs explains that for many linguistic applications it is acceptable to relax ontological scruples, intricate syntactic explanations, and the desire for efficient deductions in favor of a simpler notation closer to English.

For the logic representation of WordNet glosses plurals and sets, verb tenses, auxiliary verbs, quantifiers and modal operators, comparatives and negation are ignored. This decision is based on the desire to provide an acceptable and consistent
logic representation that otherwise would be infeasible. Other projects using different logical representations can still use the representation we provide and adapt it to their quantification.

A predicate is generated for every noun, verb, adjective or adverb encountered in any gloss. The name of the predicate is a concatenation of the word’s base form, the part-of-speech and the WordNet semantic sense, thus capturing the full lexical and semantic disambiguation. For example, the LFT of the gloss of \{terrorist\} is (a radical who employs terror as a political weapon). It will contain the predicates radical:n#3, employ:v#1, terror:n#1, political:a#1 and weapon:n#1. Predicates can be classified as: \textit{ground predicates} (noun predicates - they have a meaning by their own), satellite predicates (adjective, adverbs - they stand near a ground predicate or verb predicate), linking predicates (conjunctions, prepositions - they express/link other predicates together). Verb predicates have a special behaviour: they are both ground predicates as they have an argument on their own (first argument identifying the event) and linking predicates (express a relation among subject and objects).

### 2.3.2. Verb Predicates

In the spirit of the Davidsonian treatment of the action predicates \[16\], all verb predicates (as well as the nominalizations representing actions, events or states) have three arguments: \(\text{action/state/event-predicate}(e_i, x_1, x_2)\), where:

- \(e_i\) represents the \textit{eventuality} of the action, state or event \(i\) stated by the verb
- \(x_1\) represents the \textit{syntactic subject} of the action, event or state, and
- \(x_2\) represents the \textit{syntactic direct object} of the action, event or state.

For example, the LFT of (motor that converts thermal energy to mechanical work), which is the gloss of \{engine\}, is: \(\text{motor:n#1}(x_1) \& \text{convert:v#1}(e_1, x_1, x_2) \& \text{thermal:n#1}(x_2) \& \text{energy:n#1}(x_2) \& \text{to}(e_1, x_3) \& \text{mechanical:a#3}(x_3) \& \text{work:n#7}(x_3)\). Several clarifications are in order here. When the predicate is a ditransitive verb, its representation is \(\text{verb}(e_i, x_1, x_2, x_3)\).
For example: (a cudgel used to give someone a beating on the soles of the feet) is represented as: \( \text{cudgel}(x_2) \) \& \( \text{use}(e_1, x_1, x_2) \) \& \( \text{to}(e_1, e_2) \) \& \( \text{give}(e_2, x_1, x_2, x_3) \) \& \( \text{beating}(x_2) \) \& \( \text{someone}(x_3) \) \& \( \text{on}(x_2, x_4) \) \& \( \text{soles}(x_4) \) \& \( \text{of}(x_4, x_5) \) \( \text{foot}(x_5) \). This condition is detected by the presence of two noun phrases following a verb in active voice. The arguments of verb predicates are always in the order: subject, direct object, indirect object. When one of these syntactic roles is missing, its respective argument appears under the verb predicate, but that argument will not be used by any other predicate. This is a so-called “slot-allocation” representation since the position of the arguments is fixed for the purpose of a simpler notation. Since in WordNet glosses not many verbs have indirect objects, the argument \( x_3 \) is used only when necessary, otherwise it is omitted.

However, the arguments for the subjects and direct objects are always present, even when the verb does not have these syntactic roles. We found that this simple and consistent representation is easy to derive and use.

2.3.3. Complements

Complements role within a phrase is replicated in the LFTs. Predicates generated from modifiers share the same arguments with the predicates corresponding to the phrase heads. Adjective predicates share the same argument as the predicate corresponding to the noun they modify. An exemplification is the LFT of the gloss of \{acorn tube\}, which maps (a small vacuum tube) into \[ \text{vacuum tube} \#1(x_1) \) \& \text{small}\(#1(x_1)\]. Similarly, the argument of adverbial predicate is the argument marking the eventuality of the event/state/action they modify.

For example, the gloss of the verb synset \{vanish, fly\} is (decrease rapidly), producing the LFT = \[ \text{decrease}(e_1, x_1, x_2) \) \& \text{rapidly}(e_1)\].

So far we discussed about representation of content words. Next we focus on predicates that link together the predicates of content words leading to the compositional meaning of an utterance which is a desiderata in the knowledge representation world.
2.3.4. Conjunctions

Conjunctions are transformed in predicates, thereby enabling the aggregation of several predicates under the same syntactic role (e.g., subject, object or prepositional object). By convention, conjunction-predicates have a variable number of arguments, since they cover a variable number of predicates.

The first argument represents the “result” of the logical operation induced by the conjunction (e.g., a logical and in the case of the and conjunction, or a logical or in the case of the or conjunction). The rest of the arguments indicate the predicates covered by the conjunction, as they are arguments of those predicates as well.

Table 2.1. Examples of conjunction predicates

<table>
<thead>
<tr>
<th>Synset</th>
<th>Gloss</th>
<th>LFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>{acoustic}</td>
<td>(a remedy for hearing_loss or deafness)</td>
<td>remedy:n#1(x_2) &amp; for(x_1,x_2) &amp; or(x_2,x_3,x_4) &amp; hearing_loss:n#1(x_3) &amp; deafness:n#1(x_4)</td>
</tr>
<tr>
<td>{think_twice}</td>
<td>(consider and reconsider carefully)</td>
<td>and(e_1,e_2,e_3) &amp; consider:v#1(e_2,x_1,x_2) &amp; reconsider:v#1(e_3,x_1,x_2) &amp; carefully:r#1(e_1)</td>
</tr>
<tr>
<td>{ateleiosis, ateliosis}</td>
<td>(physical underdevelopment but normal intelligence)</td>
<td>physical:a#2(x_2) underdevelopment:n#1(x_2) &amp; but(x_1,x_2,x_3) &amp; normal:a#3(x_3) &amp; intelligence:n#1(x_3)</td>
</tr>
</tbody>
</table>

Table 2.1 illustrates several conjunction-predicates. In the case of the LFT for the gloss of {acoustic}, the predicate or indicates that the syntactic object of the preposition for is the result of the logical or between hearing_loss and deafness.

Predicate and in the LFT for the gloss of verb synset {think_twice} indicates that the genus is both consider and reconsider, and they are both modified by carefully.

Finally, the LFT of the gloss (a form of infantilism characterized by physical underdevelopment but normal intelligence) of {ateleiosis, ateliosis} illustrates the transformation of a conjunction between two antagonistic nouns. Both underdevelopment and intelligence relate to noun infantilism, only they do so via the conjunction
but, which cues a contradiction between the two. Such cues are important for textual processing (e.g. they indicate the coherence structure of texts [38]), thus it is important to incorporate them in the LFTs.

2.3.5. Prepositions

We also generate predicates for every preposition encountered in the gloss. The preposition predicates always have two arguments: the first argument corresponding to the predicate for the head of the phrase to which a prepositional phrase is attached, whereas the second argument corresponds to the prepositional object. Prepositions are linking predicates.

Table 2.2. Examples of preposition predicates

<table>
<thead>
<tr>
<th>Synset</th>
<th>Gloss</th>
<th>LFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>{empire}</td>
<td>a monarchy with an emperor as head_of_state</td>
<td>monarchy: n#1 (x_1) &amp; with((x_1, x_2)) &amp; emperor: n#1((x_2)) &amp; as((x_2, x_3)) &amp; head_of_state: n#1((x_3))</td>
</tr>
<tr>
<td>{break out}</td>
<td>(take from stowage in preparation for usage)</td>
<td>take: v#7((e_1, x_1, x_2)) &amp; from((x_2, x_3)) &amp; stowage: n#2((x_3)) &amp; in((e_1, x_4)) &amp; preparation: n#3((x_3)) &amp; for((e_1, x_4)) &amp; usage: n#1((x_4))</td>
</tr>
</tbody>
</table>

This predicative treatment of prepositional attachments was first reported in [3]. Table 2.2 illustrates some transformations in preposition predicates. Prepositions are predicates that link together content predicates derived from nouns, verbs adjectives and adverbs.

2.3.6. Complex Nominals

Many complex nominals are currently encoded in WordNet as synset entries comprising several words, known as WordNet collocations (e.g. flea market, baseball team, joint venture). Still, many compound nouns are not encoded as WordNet entries, and need to be recognized as a single nominal. The way of doing this was first devised in TACITUS [34], when the predicate mn was first introduced. Similar to
conjunction predicates, the \( nn \) predicates can have a variable number of arguments, with the first one representing the result of the aggregation of the nouns corresponding to the remaining arguments.

Table 2.3. Examples of complex nominals predicates

<table>
<thead>
<tr>
<th>Synset</th>
<th>Gloss</th>
<th>LFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>{pork barreling}</td>
<td>(acquisition of government money for benefits to a specific locale)</td>
<td>acquisition:nn#1(( x_2 )) &amp; of(( x_1, x_2 )) &amp;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( nn(x_3, x_4, x_5) ) &amp; government:nn#1(( x_2 )) &amp;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>money:nn#1(( x_2 )) &amp; for(( e_1, x_3 )) &amp; benefits:nn#1(( x_2 )) &amp;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to(( e_1, x_3 )) &amp; specifica;#1(( e_1 )) &amp; locale:nn#1(( x_2 ))</td>
</tr>
<tr>
<td>{knuckleball, knuckler}</td>
<td>(a baseball pitch thrown with little speed or spin)</td>
<td>( nn(x_2, x_3, x_4) ) &amp; baseball:nn#1(( x_3 )) &amp;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pitch:nn#1(( x_4 )) &amp; throw:v#2(( e_1, x_1, x_2 )) &amp;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>with(( e_1, x_3 )) &amp; specifica;#1(( e_1 )) &amp; or(( x_1, x_1, x_5 )) &amp;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>speed:nn#2(( x_3 )) &amp; spin:nn#2(( x_3 ))</td>
</tr>
</tbody>
</table>

Table 2.3 illustrates the transformation of some complex nominals into their corresponding logic form predicates.

2.3.7. Comparatives

The LF notation presented in [48] ignores comparatives. This makes impossible the representation of glosses with comparatives and leads to a gap in the representation.

To fill this gap we propose here a way to deal with comparatives that maintains the simplicity and consistency of LF notation. Our approach would be to treat comparatives as introducing a relation among the compared entities, in a manner similar to the prepositional predicate.

Let us consider the definition of the concept \{tower\}: (a structure taller than its diameter). The question is whether adjective tall modifies both structure and diameter? In other words are both structure and diameter something tall? They both can be small and structure might be larger than diameter.

Hence a solution of the form: [structure:n(\( x_1 \)) \& tall:a(\( x_1 \)) \& diameter:n(\( x_2 \)) \& tall:a(\( x_2 \)) \& taller:a(\( x_1, x_2 \)) ] would be inappropriate from a semantic point of view.
Figure 2.1. Parse tree for gloss of tower:n#1: *(a structure taller than its diameter)*

With a closer look, one realizes that a comparative introduces more of a *relation* than a *relation and a modifier* as described in the previous solution and thus a more appropriate representation for the same example would be: \([\text{structure}(x_1):n \& \text{taller}(x_1,x_2) \& \text{diameter}(n(x_2))].*

Table 2.4. Examples of comparative predicates

<table>
<thead>
<tr>
<th>Synset</th>
<th>Gloss</th>
<th>LF</th>
</tr>
</thead>
<tbody>
<tr>
<td>{tower}</td>
<td><em>(a structure taller than its diameter)</em></td>
<td>(\text{structure}(x_1) &amp; \text{taller}(x_1,x_2) &amp; \text{diameter}(n(x_2)) &amp; \text{pos}(x_1,x_2)) )</td>
</tr>
<tr>
<td>{workstation}</td>
<td><em>(a desktop computer that is conventionally considered to be more powerful than a microcomputer)</em></td>
<td>(\text{desktop_computer}(n#1(x_1)) &amp; \text{consider}(v#1(e_1,x_2,x_1)) &amp; \text{conventionally}(r#1(e_1)) &amp; \text{more_powerful}(x_1,x_3) &amp; \text{microcomputer}(n#1(x_3)))</td>
</tr>
</tbody>
</table>

This solution leads to an uniform treatment for more than one linguistic case (prepositions, comparatives and postmodifiers - see next section), resulting therefore into a more cohesive representation.
a compound_lens system that forms an image free from chromatic_aboration

Figure 2.2. Parse tree for gloss of *achromatic_lens:n#1*

The next section shows how we treat the case of postmodifier that was not originally specified and required special treatment.

2.3.8. Postmodifiers

The original Logic Form representation specifies relations of type modifier but does not detail on special cases. A special case that needs further specifications refers to postmodifiers. Let us consider the postmodifier *free* in the definition of {achromatic_lens}: (a compound lens system that forms an image free from chromatic aberration).
From a semantic point of view *free* is the modifier of *image*. Trouble comes when one tries to provide a representation for preposition *from*. What phrase should stand as its prepositional head? One might say that it is appropriate to use modifier *free*. But *free* does not have an argument on its own since being a modifier it borrows the argument from its modifiee *image*.

Table 2.5. Examples of postmodifier predicates

<table>
<thead>
<tr>
<th>Synset</th>
<th>Gloss</th>
<th>LF</th>
</tr>
</thead>
<tbody>
<tr>
<td>{achromatic, lens}</td>
<td>(a compound lens system that forms an image free from chromatic aberration)</td>
<td>nn(x₁, x₂, x₃) &amp; compound, lens: n(x₂) &amp; system: n(x₃) &amp; form: v(e₁, x₁, x₄) &amp; image: n(x₄) &amp; free from(x₄, x₅) &amp; chromatic, aberration: n(x₅)</td>
</tr>
<tr>
<td>{transistor, junction transistor}</td>
<td>(a semiconductor device capable of amplification)</td>
<td>semiconductor, device: n(x₁) &amp; capable of f(x₁, x₂) &amp; amplification: n #1(x₂)</td>
</tr>
</tbody>
</table>

We can consider *image* as the prepositional head of *from* but this would be wrong as there is no such a relation as prepositional head - prepositional object between *image* and *chromatic aberration*. If we concatenate *free* and *from* into a special predicate *free from* and treat it as a relational predicate, somehow similar to a preposition, we may represent the above example as: [image: n(x₁) & free from(x₁, x₂) & chromatic, aberration: n(x₂)].

This solution maintains the simplicity and consistency of the notation and is the closest one to the semantic interpretation. It would be the task of the user of this representation to further interpret the new predicate according to her needs.

2.3.9. Possessive Pronouns

Possessive pronouns introduce a relationship between the head they stand by and the referent of the pronoun. In the gloss of {tower} the pronoun *its* introduces a possession relation between *structure* and *diameter*. To remedy the lack of specificity of the original notation regarding this case we propose to represent the relation using
the predicate \( \text{pos} \). Thus, for the previous example, we obtain \([\text{structure}: n(x_1) \& \text{pos}(x_1, x_2) \& \text{diameter}: n(x_2)]\). All possessive pronouns are represented using this predicate.

2.3.10. Relative Adverbs

Relative adverbs of the kind \textit{where}, \textit{when}, \textit{how}, and \textit{why} when introducing a relative clause should be represented as a predicate with two arguments: one referring to the argument of the relative clause or phrase and the other to the argument of the main clause or phrase.

We illustrate such a case using the definition of \{airdock\}, which is \(\text{(a large building at an airport where aircraft can be stored and maintained)}\).

Table 2.6. Examples of relative adverb predicates

<table>
<thead>
<tr>
<th>Synset</th>
<th>Gloss</th>
<th>LF</th>
</tr>
</thead>
<tbody>
<tr>
<td>{airdock}</td>
<td>(a large building at an airport where aircraft can be stored and maintained)</td>
<td>(\text{large}: a(x_1) &amp; \text{building}: n(x_1) &amp;\text{where}(e_2, x_1) &amp; \text{aircraft}: n(x_2) &amp; \text{and}(e_2, e_3, e_4) &amp; \text{store}: v(e_3, x_3, x_2) &amp; \text{maintain}: v(e_4, x_3, x_2))</td>
</tr>
<tr>
<td>{arc_lamp, arc_light}</td>
<td>(produces light when electric current flows across the gap between two electrodes)</td>
<td>(\text{produce}: v(e_1, x_1, x_2) &amp; \text{light}: n(x_1) &amp;\text{when}(e_1, e_2) &amp; \text{electric_current}: n(x_3) &amp; \text{flow}: v(e_2, x_3, x_4) &amp; \text{across}(e_2, x_3) &amp; \text{gap}: v(x_4) &amp; \text{between}(x_4, x_5) &amp; \text{two}(x_5) &amp; \text{electrode}: n(x_5))</td>
</tr>
</tbody>
</table>

The relation between the two sentences is embedded into the adverbial phrase \textit{where} and thus the corresponding logic form representation is \([\text{large}: a(x_1) \& \text{building}: n(x_1) \& \text{where}(e_2, x_1) \& \text{aircraft}: n(x_2) \& \text{and}(e_2, e_3, e_4) \& \text{store}: v(e_3, x_3, x_2) \& \text{maintain}: v(e_4, x_3, x_2)\].

This logic representation is same with the logic form representation of the imaginary sentence \{aircraft can be stored and maintained in a large building at an airport\}. One needs only the extra knowledge of equaling \textit{in} with \textit{where} to obtain the same representation for the exemplified gloss and imaginary sentence.
The previous similarity illustrates the abstractness power of logic forms: structurally different natural language representation are mapped onto the same logic form representation. Structural abstractization is also evident when representing the syntactic subject of a verb which can be expressed structurally with the verb in active voice, respectively in passive voice. The power of abstractness works even better at semantic level where, for different lexical representations of same concept, we have only one predicate.

The semantic tags of logic forms are not part of our work. We present methods for part of speech tagging and syntactic roles of arguments.
2.4. Conclusions

The logic form used for representing WordNet glosses is syntactically simple and close to English. The simplifications in representation do not affect its applicability for the tasks we plan to use it, e.g. Question Answering, as we will show later.

<table>
<thead>
<tr>
<th>Semantic (SEM) Nets</th>
<th>natural-language based(NL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semantic Nets</td>
<td>Morpheme based FOL</td>
</tr>
<tr>
<td>(KL-fam)</td>
<td>START FALCON</td>
</tr>
<tr>
<td></td>
<td>Concept based</td>
</tr>
<tr>
<td></td>
<td>(NL + SEM)</td>
</tr>
</tbody>
</table>

70s 80s 90s

Figure 2.4. Milestones in Knowledge Representation

The LFTs are highly dependable on the part of speech tagger and parser accuracy. The next chapter presents a solution to improve part of speech tagging accuracy and thus the accuracy of logic form derivation.

Figure 2.4 illustrates the milestones in knowledge representation from early works in the 60-70s, to the switch towards natural language based representation in the 80s and up to the concept-based representation proposed in the late 90s. The latter constitutes the focus of our work.
The concept based representation combines the advantages of both natural language based representations (using a natural language first order logic approach as in TACITUS) with the advantages of a lexico-semantic database (WordNet) that borrows only the semantic properties of semantic networks and not implementation details.

The concept based representation has syntax features embedded as positional information in the predicates and categorial information in the form of WordNet part of speech categories: noun, verb, adjective and adverb.

The main advantage of natural language approaches over semantic nets consists in the possibility of automatically building knowledge bases from the vast human knowledge embedded in electronic texts over the Internet.
Chapter 3
PART OF SPEECH TAGGING AND PARSING OF GLOSSES

Since the derivation of logic forms relies heavily on the output of the parser, and the parser’s accuracy relies on the tagger’s accuracy, a highly accurate tag process is paramount to our task of deriving logic forms for WordNet glosses.

We discuss in this chapter different approaches to the tagging problem and propose a two level voting scheme to improve the tagging accuracy on WordNet glosses. We also show how glosses can be extended for better parsing. By the end of the chapter some parser combination techniques are applied to an ensemble of parsers obtained from Collins’ statistical model that are trained on different partitions of the Penn Treebank (WSJ sections). The impact of the ideas in this chapter to the overall task is illustrated in Figure 3.

3.1. Tokenizing Glosses

Before they undergo the tagging process, glosses need to be tokenized. We use our own tokenizer that complies with the Treebank tokenization requirements:

- most punctuation is split from adjoining words;
- double quotes (" ) are changed to doubled single forward- and backward-quotes (" and ");
- verb contractions and the Anglo-Saxon genitive of nouns are split into their component morphemes, and each morpheme is tagged separately. Examples: children’s → children ’s, parents’ → parents ’, won’t → wo n’t, gonna → gon na, I’m → I ’m;
- hyphens adjoin separate words of similar concepts and thus are not separated from their adjoining words;
• abbreviations keep the terminating point: esp. remains esp.

Besides these generally applicable guidelines we propose extensions that are specific to the token space of glosses. These extensions have the purpose of easing the general process of deriving logic forms.

• drop also, as before in, of
• drop usually, frequently, often, formerly at the beginning of gloss
• drop specially, especially

Additionally, we identify a set of collocations that logic form derivation necessitates treatment of as a single token. Examples of such collocations are: back again, in or out, up and down, above and beyond, to and from, at least, as well as, in order to, way down, one over the other.

3.2. Tagging Glosses

After the tokenization stage, the next processing steps consist of part of speech tagging and syntactically parsing glosses. In this section we focus on the task of part of speech tagging. Let us consider the gloss of abbey:n#3 which is (a monastery ruled by an abbot). If we run the MXPOST [52] part of speech tagger we get:

\[ \text{MXPOST : } \text{a_DT monastery_NN ruled_VBN by_IN an_DT abbot_JJ} \]

The tag of abbot is incorrectly assigned by MXPOST. If we were to use this tagger we would provide the parser with an erroneous input and an incorrect parse tree would
be obtained for this gloss. An incorrect logic form is generated from an incorrect parse tree. A solution to improve the accuracy of assigned tags is needed.

It would be a tremendous effort to manually check all the tags for all definitions. To avoid this manual intervention we aim at detecting with the highest accuracy possible where the errors might occur by applying a voting scheme, and therefore reduce the human intervention.

3.2.1. Tagging Technology

There are many approaches to automated part of speech tagging. An overview of different approaches to the tagging problem will provide us with a better understanding of how we can improve the tags for WordNet glosses.

3.2.1.1. Supervised vs Unsupervised

One of the first distinctions that can be made among POS taggers is in terms of degree of automation of the training and tagging process. Supervised taggers typically rely on pre-tagged corpora to serve as the basis for creating any tools to be used throughout the tagging process, for example: the tagger dictionary, the word/tag frequencies, the tag sequence probabilities, and/or the rule set. Unsupervised models, on the other hand, are those that do not require a pretagged corpus but instead use sophisticated computational methods to automatically induce word groupings (i.e. tag sets) and based on those automatic groupings, either calculate the probabilistic information needed by stochastic taggers or induce the context rules needed by rule-based systems. Each of these approaches has pros and cons.

3.2.1.2. Rule Based Tagging

Typical rule based approaches use contextual information to assign tags to unknown or ambiguous words. These rules are often known as context frame rules. As an example, a context frame rule might be: If an ambiguous/unknown word X is preceded by a determiner and followed by a noun, tag it as an adjective.
In addition to contextual information, many taggers use morphological information to aid in the disambiguation process. One such rule might be: if an ambiguous/unknown word ends in an -ing and is preceded by a verb, label it as verb (depending on your theory of grammar, of course).

Some systems go beyond contextual and morphological information and include rules pertaining to factors such as capitalization and punctuation. Information of this type is of greater or lesser value depending on the language being tagged. In German for example, information about capitalization proves to be extremely useful in the tagging of unknown nouns.

Rule based taggers usually require supervised training; very recently there has been a great deal of interest in automatic induction of rules. One approach to automatic rule induction is to run an untagged text through a tagger and see how it performs. A human then goes through the output of this first phase and corrects any erroneously tagged words. The properly tagged text is then submitted to the tagger, which learns correction rules by comparing the two sets of data. Several iterations of this process are sometimes necessary.

3.2.1.3. Stochastic Tagging

The term 'stochastic tagger' can refer to any number of different approaches to the problem of POS tagging. Any model that somehow incorporates frequencies or probabilities, i.e. statistics, may be properly called stochastic.

The simplest stochastic taggers disambiguate words based solely on the probability that a word occurs with a particular tag. In other words, the tag encountered most frequently in the training set is the one assigned to an ambiguous instance of that word. The problem with this approach is that while it may yield a valid tag for a given word, it can also yield inadmissible sequences of tags.

An alternative to the word frequency approach is to calculate the probability of a given sequence of tags occurring. This is sometimes referred to as the n-gram approach, referring to the fact that the best tag for a given word is determined by
the probability that it occurs with the n previous tags. The most common algorithm for implementing an n-gram approach is known as the Viterbi Algorithm, a search algorithm which avoids the polynomial expansion of a breadth first search by "trimming" the search tree at each level using the best N Maximum Likelihood Estimates (where N represents the number of tags of the following word).

The next level of complexity that can be introduced into a stochastic tagger combines the previous two approaches, using both tag sequence probabilities and word frequency measurements. This is known as a Hidden Markov Model. The assumptions underlying this model are the following: each hidden tag state produces a word in the sentence. Each word is: (1) either uncorrelated with all the other words and their tags or (2) it is probabilistically depending on the previous N tags only.

3.2.1.4. Unknown Words

One problem remains outstanding in regards to all of the approaches discussed so far: How should unknown words be dealt with? Some rules in rule based taggers are designed to address this issue, but what happens in the stochastic models? It is impossible to calculate the probability that a given word occurs with a given tag if that word is unknown to the tagger. There are several potential solutions to this problem: One is the use of morphological information. In this case, the tagger calculates the probability that a suffix of an unknown word occurs with a particular tag. If an HMM is being used, the probability that a word containing that suffix occurs with a particular tag in the given sequence is calculated. Another solution is to assign a set of default tags (typically the set of open classes: N, V, Adj., Adv.) to unknown words, and to disambiguate using the probabilities that those tags occur at the end of the n-gram in question. Another possibility is to calculate the probability that each tag in the tag set occurs at the end of the n-gram, and to select the path with the highest probability. This is not the optimal solution if one is working with a large tag set.
We do not directly address this issue in this dissertation. When an unknown word is encountered, we let the tagger assign a tag and then we use WordNet syntactic category or user’s intervention.

3.2.1.5. State-of-the-art in POS tagging

The state of the art in part of speech tagging is approximately 95% accuracy for rule-based part of speech taggers [8] and stochastic taggers [52]. As pointed out in [52], the convergence of accuracy of different approaches to tagging can be explained as either all techniques miss the right predictors to cover the residue or more likely the performance of corpus based algorithms cannot be higher due to consistency problems. Thus, instead of trying to improve the models which may lead to nothing due to inconsistencies, we use a voting scheme that uses the output of two state of the art taggers to improve the performance. If the two taggers agree we assess that tag as being correct. For words that have different tags a second voting scheme is used and we ask the user to manually decide which one is correct.

3.2.2. The Voting Scheme

Two main issues need to be solved for the task of tagging glosses:

- error detection in a large tagged corpus which in our case is formed by all WordNet definitions;
- identify a methodology to increase tagging accuracy with none or minimum user intervention;

The solution adopted for error detection is to cast a vote: compare the output of two taggers. The solution to the second problem is to use a second comparison of two outputs using a third tag classification or to ask the user for manual assessment of the tag (see Figure 3.2).

The main steps of our approach are:

- extract all definitions and tokenize them
- apply the first and second taggers on definitions
Figure 3.2. Overview of the two level voting scheme
• *vote:* detect the agreement set for the two taggers

• for the set where the two taggers disagree, apply a new *voting scheme* using the first tagger and a third one

• evaluate the amount of data on which user intervention is required

There are several part of speech taggers available. We had several guiding principles in choosing the part of speech taggers:

• it should have state-of-the-art reported results which is about 94% on free text

• it should fit well with the parser (the parser should recognize tagger’s tag set)

• it should have compatible tag sets with other taggers

In addition, we looked for a tagger that can be easily improved on our specific data set. The two taggers that most satisfied our criteria are Brill’s rule-based part of speech tagger and MXPOST probabilistic tagger.

3.2.2.1. WordNet - The Third Tagging Resource

The third tagging resource we use is WordNet’s *coarse* classification of words in four syntactic categories: nouns, verbs, adjectives and adverbs.

3.2.3. Tagging Definitions

The tagged words in glosses are divided into two major classes: (1) those having identical tags assigned by both taggers (2) those with different tags by Brill’s and MXPOST. For the latter case another agreement is sought with WordNet, but only for words being monotagous (one tag) in Brill’s dictionary. For those words, there must be a single part of speech data file in WordNet that contains the stemmed form of the word. Any other case is considered for manual checking.

If we reconsider the previous example and tag the gloss of *abbey :n#3* using Brill’s tagger [8] and MXPOST tagger [52], we obtain the following tags.
Brill’s: a/DT monastery/NN ruled/VBN by/IN an/DT abbot/NN
MXPOST: a_DT monastery_NN ruled_VBN by_IN an_DT abbot_JJ

Except the last word tag, abbot, the two taggers agree. The tag for abbot is automatically assigned using a second voting scheme between Brill’s tagger and WordNet tags: abbot is a noun in WordNet which leads to an agreement with ’NN’, the tag assigned by Brill’s tagger.

3.2.4. Experiments and Results

To evaluate our method we selected to noun hierarchies and one verb hierarchy from WordNet.

3.2.4.1. Tagging noun.artifact and verb.social

We performed experiments on one noun hierarchy, noun.artifact, and one verb hierarchy, verb.social. They contain 9810, respectively 1007 synsets. After extracting the definitions and tagging them, we identified 109,143 tags for noun definitions, and 7,307 for verb definitions.

Table 3.1 presents the accuracy of Brill’s tagger, respectively MXPOST, on the two hierarchies, measured by taking a sample of 1000 tagged words and manually checking the tags.

<table>
<thead>
<tr>
<th>Tagger</th>
<th>noun.artifact</th>
<th>verb.social</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brill’s tagger</td>
<td>96.4%</td>
<td>96.8%</td>
</tr>
<tr>
<td>MXPOST</td>
<td>95.88%</td>
<td>96.01%</td>
</tr>
</tbody>
</table>

We noticed that one main source of errors in tagging is constituted by unknown words (i.e. words missing from the lexicon). Brill’s tagger assigns tags to such words based on few heuristics that consider the word suffixes. A common example is Americanism, which is not in Brill’s dictionary and is tagged as JJ, based on its
termination, and not as a noun as defined in WordNet. MXPOST assigns the tag based on several features apriori set. It turns out that a good idea would be to expand the dictionaries of the taggers with WordNet entries that are missing. That would involve for each entry in WordNet data files to check if that word and its morphological derivations are present in taggers’ lexicons. If not, add the WordNet entry in the dictionaries with the corresponding tags.

There are at least two main problems with this approach:

- A WordNet word might be present in two part of speech files and thus entitled to have more than one tag assigned to it. Several tags can be assigned to a dictionary entry by ordering them, based on their likeliness of being the correct tag for the word (usually the likeliness of a tag for a word is measured using the frequency of the tag for that word in large corpora). The more likely a tag for a word is correct, the closer to the beginning of the list the tag should be. Lack of frequency information about the tags for that missing word may place the less frequent tag first in the list and thus lead to an erroneous tag assignment for that word.

- Even if the word is present in only one WordNet file the correspondence from WordNet parts of speech to finer tag classifications used by taggers is very difficult to perform, especially for verbs where, starting with a base form, one needs to derive all its morphological derivations and select one or several, considering the context of the verb in the gloss. For the noun *may* entries for plurals need to be added with the list of tags {NNS}.

To overcome the problems encountered with expanding dictionary entries, instead of trying to derive all possible tags for a WordNet category, we choose to map assigned tags to WordNet categories. Whenever there is a missing entry in the dictionary, we check if there is a monotagamous entry in WordNet for the base form of the word and if the WordNet category is loosely the same with the tag assigned by the tagger. We say *loosely* because we will map NN, NNS, NNP, NNPS into *noun* and proceed similarly for verbs, adjectives and adverbs.
3.2.4.2. Results

The results are summarized in Table 3.2. We notice that Brill’s and MXPOST agree in about 91.60% of the cases (slightly below the baseline of assigning the most frequent tag to each word) with an accuracy on those cases of 98.78%.

Table 3.2. Brill’s and MXPOST taggers agreement

<table>
<thead>
<tr>
<th>Data</th>
<th>Agreement</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>noun.artifact</td>
<td>91.60%</td>
<td>98.78%</td>
</tr>
<tr>
<td>verb.social</td>
<td>91.54%</td>
<td>98.23%</td>
</tr>
</tbody>
</table>

For disagreement cases, using WordNet to validate monotagous words brings 0.94% improvement in agreement for nouns, and 0.88% for verbs. For those

In summary, we successfully tagged 91.57% words with an accuracy of 98.50% and 0.91% words with 100% accuracy. A human needs to check 7.52% of the words. Supposedly, the human does a perfect job and we will have an overall accuracy of 98.93%.

A high performance at part of speech level was mandatory as the next processing step, parsing, relies on parts of speech to offer correct bracketing for a given sentence.

3.3. Parsing Glosses

Parsing is the task of delimiting phrases of a sentence and describing the relations between them. The parser is given an unmarked sentence and is required to perform these annotations. We base our work on Penn Treebank parsing style. In their annotation, which is a combination of many grammatical theories, chunks of text are delimited by brackets and identified by labels.

The bracketing can be seen as forming multipath trees where there is a unique path from each word to the root of the tree. Part of speech tags are the preterminal nodes in this tree, and every word has a part of speech tag associated with it. In
Figure 3.3. Parsing in the architecture of a Logic Form Transformer

parser evaluations, part of speech tagging is treated as a separate task, so those nodes are treated differently from the rest of the tree.

3.3.1. Parsing Technology

The earliest work on corpus-based automatic parser induction dates to Black [4] who describes the metrics that are still used for measuring parser performance. Around the same time, Pereira and Schabes produced some experimental results on PCFG-style parser induction.

Early work on parsing using the Penn Treebank was done by Magerman [37], Brill [9] and Collins [13]. Magerman’s system controlled a left-to-right parser using a decision tree. Brill’s system used automatically-learned rules for transforming initially poor parse trees into better ones. Vilain and Day [68] produced a faster version of the transformation-based parser. Collins’ work was one of the first successful PCFG head-passing grammar-based systems for this task.

More recently Ratnaparkhi [53], Charniak [11] and Collins [14] have each independently developed statistical parsers using the same training and testing split of the Penn Treebank. Collins and Charniak both use a head-passing PCFG as the basis of their models, although the features they use for their models are different. Ratnaparkhi uses a maximum entropy classifier to control a machine that iteratively builds and prunes a parse tree from the bottom up.

The purpose of parsing is to remove as much ambiguity in a sentence that can be determined by syntax as possible.
Figure 3.4. Interpretations for *I saw the man in the park with a telescope*

For instance, consider this famous example of the sentence *I saw the man in the park with a telescope*, which can be interpreted differently in different contexts. Choosing between different potential interpretations (see Figure 3.4) is the task of the parser. Goodman’s work [19] develops some formal approaches to defining parsing systems and shows how to create parsers that directly maximize some given performance metrics. He gives separate automated parser induction algorithms that directly maximize recall and an approximation of precision. Also, he points out that there is a basic incompatibility between parsing with the goal of getting sentences correct and parsing with the goal of getting constituents correct. The two metrics have the same maximum point, namely when everything is parsed correctly, but in practice there is a tradeoff involved in maximizing them independently. Goodman also provides practical techniques for parsing with large vocabularies and large grammars. He presents experiments involving multi-pass pruning algorithms for parsing under computational time and space constraints.
The parsing community has recently had a large improvement in accuracy while suffering from a loss in speed. Caraballo and Charniak [10] address this issue by finding a good heuristic for searching in a PCFG-style parser.

Chelba and Jelinek have created an online parser which operates in a left-to-right manner like a pushdown automaton in order to better perform language modeling for speech recognition. They use a maximum likelihood technique to learn the controlling automaton for a shift-reduce parser.

The successes in parsing English text boosted the development of parsers for other languages including Czech [15] and Japanese [27]. Each of these languages has required a redesign or modification of the task. They each operate in a dependency representation. Each word (or chunk) is annotated with an arrow directed toward the word that it syntactically supports. In Czech this is required because the word order is much more liberal than in English. In Japanese, each phrase is guaranteed to modify a phrase that comes before it, but not necessarily the most recent phrase.

Parsing is an important component of many modern word processing applications. Its initial purpose was to determine if sentences conform to a grammar. It has progressed quite a bit since then, but this task has become important in commercial software. Translating between languages with differing word order, parsing is a crucial step.

There is a strong belief that once words and small phrases can be translated, transformations on the parse of a sentence can be used to rearrange large portions of text to make it conform to the expected ordering. In summarization statistical parsers are used to detect repetitive phrases that can be removed in order to make the text syntactically more concise.

Problems similar in structure to parsing were identified in other domains such as computational biology where an important issue is to determine the hierarchical physical structure of molecules that are created from sequences of RNA. Next we focus on few performance measures for parsing.
3.3.1.1. Performance Measures

Parsing performance is measured in a number of ways. Each starts with counting the three observable situations that can occur in a prospective parse. The parse trees are broken into constituents. Each constituent consists of a label and a span. We can then observe three situations: (a) The suggested constituent is in the suggested parse and in the correct parse and it is a correctly predicted constituent (b) The constituent suggested by our parser is not in the correct parse and a precision error occurs (c) The constituent in the correct parse is not in our parse, and we have a recall error. The metrics for parser performance are:

- Precision (P) is the fraction of the constituents that the parser produces that are correct: \[ \frac{a}{a+b} \]
- Recall (R) is the fraction of the correct constituents that the parser produces: \[ \frac{a}{a+c} \]
- F-measure is the harmonic mean of precision and recall. Geometrically F-measure represents the ratio of the area of the rectangle with corners (0,0) and (P,R) with respect to its perimeter, normalized such that the maximum value is 1.0. To calculate: \[ \frac{2PR}{P+R} \text{ or } \frac{2a+b+c}{2(a+b)(a+c)} \]
- Exact sentence accuracy reports the percent of attempted sentences that are parsed correctly (b=c=0).

Black [4] developed an evaluation system for comparing hand-coded parsing systems. The statistical parsing community has followed this design in performing evaluations, as does our approach.

The focus is on a labeled bracketing method of parser scoring. In the following denote with \( T_g \) denotes the parse tree generated for a sentence by the studied parser and \( T_r \) the correctly parsed reference tree.

The EVALB algorithm described below implements the performance guidelines given above. First it does some cleaning at parse tree level and then is identifies the category of each constituent.
3.3.1.2. Algorithm 3.1: EVALB Transformation

- Strip all epsilon productions from tree $T_g$, as most parsers do not generate epsilon productions.
- Remove all terminals that are POS-tagged with punctuation from both $T_g$ and $T_r$.
- Repeatedly remove all constituents from the tree that no longer span any tokens from the original sentence due to pruning.
- Create $S_r$ from the reference parse. This is the set of tuples $(s,e,l)$ where $s$ is the number of terminal nodes to the left of the left side of the constituent’s span, $e$ is the sum of $s$ and the number of terminal nodes dominated by the constituent, and $l$ is the label on the constituent. Similarly create $S_g$ from the $T_g$.
- Remove any constituent that dominates all the other nodes in $S_r$. Do the same for $S_g$. Every sentence has a topmost constituent spanning it, so we need not to count it. It is taken as given that all parsers produce it.
- Produce the error distribution numbers $a, b$ and $c$
- Compute $P$, $R$, $F$ and exact sentence accuracy measures

There are several ramifications of this algorithm that should be observed.

First, the parser may use punctuation to help perform the parse, but how the parser brackets punctuation has no effect on the final score. For example, it makes no difference where the final period attaches, or whether the quotes around a quotation are included in the constituent dominating it. Punctuation is ignored for purely historical reasons. Some of the earliest parsers represented punctuation as it is typed - most often as part of an adjacent word, whereas others treat punctuation as separate tokens.

Second, the set of productions used in sentence parsing is not restricted to the set found in the correct parse. Each constituent is identified only by its label and

---

1Epsilon productions appear in the corpus to encode traces describing special linguistic phenomena (e.g., wh-movement). They yield leaf nodes that do not correspond to observed tokens.
span. Its correctedness does not depend on the labels on its children. The parse has been simplified at this point to a set of triangles with labels on them.

Third, this algorithm has meaning for parses that are not necessarily trees. It works with any acyclic graph with the appropriate terminal nodes.

3.3.2. Single Parser Performance on Glosses

In this study an in-house implementation\(^2\) of a parser is used which follows the statistical parsing principles described in [14]. The parser is based on the probabilities between head-words in parse trees.

First, the statistical model assigns a probability to every candidate parse tree for a sentence. The most likely parse under the model is

\[
T_{\text{best}} = \arg \max_T P(T|S)
\]

where \(P(T|S)\) is the conditional probability of a sentence \(S\) being parsed with a tree \(T\). The task of the parser is to find \(T_{\text{best}}\). The key of the model is that any parse tree \(T\) is mapped into a set of base-NPs (B) and a set of dependencies (D): \(T=\{B, D\}\). The conditional probability is rewritten as

\[
P(T|S) = P(B, D|S) = P(B|S)P(D|S, B)
\]

The set of dependencies \(D\) is obtained by first detecting head words for each phrase and extracting tuples of the form (head-word, non-head child word) from each constituent. The baseNP model can be viewed as tagging the gaps between words with S(tart), C(ontinue), E(nd), B(etween) or N(ull) symbols, respectively meaning that the gap is at the start of a BaseNP, continues a baseNP, is at the end of a BaseNP, is between two adjacent baseNPs, or is between two words which are both not in BaseNPs. The parsing algorithm is a simple bottom-up chart parser. The parser searches through the space of all trees with nonterminal triples seen in the training data. The lower probabilities candidates can safely be discarded. To test the parser

\(^2\)Developed by Mihai Surdeanu
accuracy on glosses we extracted 400 glosses from noun.artifact hierarchy, cleaned them up and corrected the tags by hand. We parsed them and measured a 62.67% accuracy for parsing. This accuracy is not satisfactory and calls for improvements. The main source of errors derives from the tendency of the parser to structure any input line into a sentence phrase. For example, for gloss, (the operating part that transmits power to a mechanism), the parser incorrectly forms an S phrase: (TOP (S (NP (DT the) (VBG operating) (NN part) ) (NP (WDT that) ) (VP (VBZ transmits) (NP (NN power) ) (PP (TO to) (NP (DT a) (NN mechanism) ) ) ) ) (. .) ) ). This situation is more frequent for glosses containing relative clauses. There are 124 glosses (out of 400) containing relative clauses and in 60 (48.38%) of the cases the relative pronouns are placed in the wrong structure leading to an incorrect parse tree.

3.3.2.1. Gloss Expansion

To compensate for the tendency of the parser to build sentence structures, our solution is to feed in full sentences. Glosses are expanded to full sentences using several patterns as described below.

- For noun glosses, add the first word of the synset, followed by “be”. For example, the definition of {prophet, oracle} becomes Prophet is an authoritative person who divines the future.
- For verb glosses, add to “verb” is to to each gloss definition. Example: the definition of {divine} with sense 1 becomes To divine is to perceive intuitively or through some inexplicable perceptive powers.
- For adjective glosses, add It is something to each definition. Example: The definition of {authoritative, important} becomes It is something having authority or ascendancy or influence.
- For adverb glosses, add It means to each definition. For example: the gloss of {intuitively} is extended to It means in an intuitive manner.

Although the above patterns are sufficient for most of the cases, a few alternatives are used. For example for noun glosses we use the following alternatives: (1)
If noun gloss begins with MD or VBZ add only *noun* as in “Noun performs simple arithmetic functions” or “Noun can be changed to different settings” (2) If the gloss begins with RB, VBN, where, IN add *noun is something* as in “Noun is something designed to serve a specific function .“ (3) If the gloss begins with plurals NNS, NNPS add *Nouns are* as in “Nouns are projectiles to be fired from a gun”.

The improvement in parsing accuracy obtained by these expansion techniques is significant: from 62.67% accuracy on raw glosses to 85.25% accuracy on expanded glosses. Only 8 (6.45%) out of 124 glosses with relative clauses have the relative clause incorrectly identified when glosses are expanded prior to parsing.

To further improve parsing accuracy we apply several combination techniques. Due to lack of publicly available parsers our experiments are a simulation of a combination of parsers since individual parsers are siblings obtained from same underlying statistical model.

3.3.3. Parsers Combination

There are two main techniques for parser combination: constituent voting and parser switching. Before getting into the details of each technique, we have to solve the problem of obtaining an ensemble of state-of-the-art parsers. As we mentioned before there is a lack of publicly available parsers. As an alternative solution, we start from our own implementation of a statistical parser and train it on different data.

The limitation of the statistical parser we use is mainly due to three different things: (1) model errors (2) estimation error (3) sparse data or insufficient training data. Modeling errors come from the inadequacies of the model. In linguistic processes the model is usually hidden and we have to guess what the real model is.

We often make the models weak or inaccurate because we know there is not enough data to accurately estimate the parameters of a better model. Estimation errors come from our lack of access to the true probabilities or parameters which shape out our model. We cannot use complex models without accurate probability estimates and we can only produce accurate estimates for small parameter spaces
given our limited data. In the case of WordNet glosses, the lack of a WordNet
treebank of correctly parsed glosses leaves us with the impediment of not being able
to train a statistical model for glosses, or the cost of building such a treebank will
overcome the cost of combining parsers with different biases. The parsers would have
to be the result of a unified research effort in which the errors made by one parser
are made a priority target for the developer of another parser.

We would willingly accept five parsers with low exact sentence score of 40% as
long as they make those errors in such a way that at least two of the five are correct
on any given sentence (and the others abstain or are wrong). We could achieve 100%
sentence accuracy simply by selecting the parse that was suggested by two of the
parsers.

3.3.3.1. Forming an Ensemble of Parsers

The starting point of our process of obtaining the assembly is a very efficient
implementation of Collins’ statistical model developed by Mihai Surdeanu. We obtain
the individual parsers using bagging.

Efron and Tibshirani developed methods for estimating statistics describing a
data set using a machine-intensive technique called bootstrap estimation. In short,
they found that they could reduce the systematic biases introduced by many es-
timation techniques by aggregating estimates that they made on randomly drawn
representative resamplings of those datasets. The representative resamplings were
designed to be the same size as the original datasets, and each sample was chosen
uniformly at random with replacement.

Breiman’s work illustrated how bootstrap estimation can be used for machine
learning. The technique is called bagging, short for “bootstrap aggregating”. Bagging
attempts to find a set of classifiers which are consistent with the training data, dif-
ferent from each other, and smoothly distributed such that the most likely classifier
to be added to the ensemble is the classifier created based on the training data. The
main steps of the algorithm are given below.
Algorithm 4.1: Bagging Predictors [7]

Given: training set $\varphi = (y_i, x_i), i \in 1...m$ where $y_i$ is the label for example $x_i$, classification induction algorithm $\psi : Y \times X \rightarrow \Phi$ with classification algorithm $\phi \in \Phi$ and $\phi : X \rightarrow Y$.

- Create $k$ bootstrap replicates $\varphi$ by sampling $m$ items from $\varphi$ with replacement. Call them $\varphi_1...\varphi_k$.
- For each $j \in 1...k$, Let $\phi_j = \Psi(\varphi_j)$ be the classifier induced using $\varphi_j$ as the training set.
- If $Y$ is a discrete set then for each $x_i$ observed in the test set,
  
  $y_i = maj(\phi_1(x_i)...\phi_k(x_i))$. We are taking $y_i$ to be the value predicted by the most predictors, the majority vote.

There are two interesting qualitative properties of bagging. First, bagging relies on the lack of stability in the chosen classifier induction algorithm’s lack of stability. By this we mean the chosen algorithm should be easily perturbed. A small change in the training set should produce a significant change in the resulting classifier. Neural networks and decision trees are examples of unstable classifier systems, whereas k-nearest neighbor is a stable classifier. Secondly bagging is theoretically resistant to noise in the data and bias in the learning algorithm. Unfortunately it is resistant to bias in the learning algorithm even when that bias is favorable. In some cases classifier induction algorithms that perform well in isolation can perform poorly in ensemble for this reason. Empirical results have verified both of these claims.

The next section focuses on the particular problem of bagging a parser under the assumption that the granule of the data is a sentence (as compared to other approaches that use other atomic elements such as constituents).

3.3.3.2. Bagging a Parser by Sentences

Bagging for parsing was applied by Hajic [20], Henderson [30] and [56][57]. Henderson studies the problem of bagging for parsing using uniform distribution over sentences, respectively constituents or preferring shorter sentences.
Bagging a Parser [30]

Given corpus \( \text{corp} \) with size \( m = |\text{corp}| = \sum_s \text{corp}(s, t) \) and parser induction algorithm \( g \).

- Draw \( k \) bootstrap replicates \( \text{corp}^1 \ldots \text{corp}^k \) each containing \( m \) samples of \((s, t)\) pairs randomly picked from the domain of \( \text{corp} \) according to the distribution 
  \( D(s, t) = \frac{\text{corp}(s, t)}{|\text{corp}|} \). Each bootstrap replicate is a bag of samples, where each sample in a bag is drawn randomly with replacement from the bag corresponding to \( \text{corp} \).
- Create parser \( f^i = g(\text{corp}^i) \) for each \( i \). \( F_{ensemble} = \bigcup_i f^i \)
- Given a novel sentence \( \text{sent}_{test} \in \text{corp}_{test} \), combine the collection of hypotheses \( t_i = f^i(\text{sent}_{test}) \) using unweighted constituent voting.

We start out with some definitions. First let, \( s = (w_1, w_2, \ldots, w_n) \) be a sentence containing \( n \) words (or punctuation). We will represent a parse tree referring to that sentence as \( t = (i, j, l) : i, j \in 0 \ldots n, j \geq i, l \in \text{NT} \) where \((i, j)\) denotes the span of the constituent by giving the indices of the start and end points in the sentence. Index 0 is the position prior to the first word and index \( n \) represents the position after the final word. The label for the constituent is \( l \) from the set of possible nonterminal labels for the constituents, \( \text{NT} \). The constituents must be properly nested as well:

\[
\forall (i_a, j_a, l_a), (i_b, j_b, l_b) \in t (i_a \leq i_b \wedge j_a \geq j_b) \lor (i_a \geq i_b \wedge j_a \leq j_b)
\]

There is traditionally a formal dominance specified when \( i_a = i_b \) and \( j_a = j_b \), but we are not including it in our model. Typically it just follows simple global rules on the constituent labels, such as constituents marked as sentences dominating constituents as verb phrases.

3.3.3.3. Individual Parsers

Our training data is the WSJ treebank from University of Pennsylvania [39]. The idea is to partition the treebank into several independent subtraining sets and train the statistical model on each of them, thus leading to parsers with different
biases that will be flattened out by combination. What would be the minimum size of partitions that provides parsers with a high performance (more than 95% of performance of a parser trained on the whole WSJ treebank)?

Chart 3.5 illustrates the evolution of overall parsing performance, as tested on section 23 from WSJ, while the parser was trained on section 0, then 0-1, and finally 0-22. The numbers on x-axis represent percentage of sentences in training sections considered relative to the total number of sentences in sections 0-22 in Treebank (5% represents around 2500 sentences).

We notice that Precision/Recall reach 74.10/74.65% when trained only on the first section (1,921 sentences or approximately 3.95% of whole Treebank) and slow increases with more training data. The pattern observed might be explained by the fact that all sections have a large amount of similars but each section brings new bigrams and lexical information.

This justification is emphasized by a second experiment in which we explored the recall of unique bigrams on increasing number of sections from WSJ. In this experiment we detect a number of unique bigrams for left rules (non-head-child non-terminal, the parent non-terminal and the head-child non-terminal), right rules (the head-child non-terminal, the parent non-terminal and non-head-child non-terminal), and unary rules (parent non-terminal and the head-child non-terminal) in increasing sizes of training data from WSJ.

The chart 3.6 shows the evolution of the number of unique bigrams versus the training size. The y axis represents percentages of unique bigrams relative to all bigrams retrieved from entire WSJ treebank.

We observe an overlap of around 70-75%, on average, of common bigrams among two different sections in WSJ. This emphasizes the fact that one section catches around 70-75% of the syntactic patterns in the WSJ treebank and what makes the improvements to 85-86% performance are the peculiarities of each new training section.
Figure 3.5. Parser performance as a function of training size.
Figure 3.6. Number of bigrams vs Training Size (on WSJ)

The two experiments led us to the conclusion that an overall performance within 95% of the one reported when training on the whole WSJ would require treebank partitions of at least 10,000 sentences. This size of training data leads to a precision (83.62%) and recall (83.86%) which are more than 95% of the performance measures on the whole WSJ: precision of 85.25% and recall of 85.34%. This is in agreement with the study presented in [30]. Since WSJ has almost 50,000 sentences we can obtain 5 parsers with an accuracy within 95% of performance of a parser trained on the whole WSJ.

In the following experiments we combine the output from an ensemble of five parsers obtained by partitioning WSJ treebank in five smaller treebanks. The way we partition the treebank is different from bagging [30] in that the size of the bags is not comparable with the size of original treebank and there is no replication. We experiment with bagging and obtained comparable performance for individual parsers.

We compare the results of voting to the original parser (trained on the entire WSJ) and not to individual parsers (trained on a partition of WSJ) since our goal is not to study the problem of voting (see [30]) but to improve the gloss parsing. Voting
results, when compared to individual parsers, might look better. This is mainly due to the way we derive the individual parsers.

3.3.3.4. Constituent Voting

A given gloss is parsed by all individual parsers and each parse tree is then broken into constituent (non-terminal tag, start position and end position in sentence). The constituents are then selected based on different methods (for example, those appearing in at least half of the original trees). An intuitive overview of the method is given in Figure 3.3.3.4. Suppose we have the trees in the upper half of Figure 3.8. Each tree is mapped onto the set of constituents underneath and then voting is applied to those constituents. One may notice that every pair of trees have a common constituents: \((S, \text{start, end})\) which spans the entire input sentence. This constituent is not considered when voting and breaking ties.
Final constituents (as given by individual parsers) are selected based on democratic voting: if the constituent is formed in more than half of the candidate trees, then the constituent is kept, otherwise it is rejected. One of the main drawbacks of constituent voting is the fact that there is no guarantee to produce sets of constituents with no crossing brackets. However if the number of votes is greater than half of the parsers under consideration the resulting structure has no crossing constituents [30].

Table 3.3.3.4 gives results of voting at constituent level for an ensemble of five parsers. The constituents output by the voting scheme were compared against the prepared test data. The Original row shows the values for individual parser and then each row presents performance values when 1, 2, 3, 4 and 5 votes were necessary for a constituent to be picked as being correct. The best F-measure occurs when constituents win three votes out of 5 possible. We experimented also with an ensemble of 3, 7 and 9 parsers and the best F-measure occurred for the same case of simple majority: half classifiers plus one votes for a constituent to be picked.
The F-measure for the simple majority voting scheme offers both better precision and recall over individual parser. The main drawback is that not all the productions in the combined parse are found in the grammars of the member parsers. The results has overlapping constituents.

Table 3.3. Results for constituent voting using five parsers

<table>
<thead>
<tr>
<th>Votes</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
<th>Exact Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>92.47</td>
<td>91.29</td>
<td>91.87</td>
<td>65.67</td>
</tr>
<tr>
<td>1</td>
<td>87.73</td>
<td>95.95</td>
<td>91.65</td>
<td>59.56</td>
</tr>
<tr>
<td>2</td>
<td>91.84</td>
<td>94.90</td>
<td>93.34</td>
<td>67.45</td>
</tr>
<tr>
<td>3</td>
<td><strong>94.39</strong></td>
<td><strong>93.99</strong></td>
<td><strong>94.18</strong></td>
<td><strong>72.80</strong></td>
</tr>
<tr>
<td>4</td>
<td>95.66</td>
<td>90.96</td>
<td>93.25</td>
<td>63.85</td>
</tr>
<tr>
<td>5</td>
<td>96.70</td>
<td>86.40</td>
<td>91.26</td>
<td>46.22</td>
</tr>
</tbody>
</table>

This drawback becomes more critical when parsing is a middle task in an overall job and other processing follows, e.g. logic form transformation.

3.3.3.5. Parser Switching

From a set of candidate parse trees provided by different parsers for a given gloss we would like to pick a parse tree that is the most plausible. We use the same ensemble of five parsers and test data as for constituent voting.

First of all we estimate what would be the lower and upper limits and a baseline for parser switching.

The first row of the table 3.3.3.6 provides the performance measures for the case when one would pick the worst parser from the candidate parse trees while the second row provides results for the case when one would pick the best parser.

The third row represents a baseline: one picks a random parser from the candidate set. The question is whether or not we can do better than random picking. We experimented with two main techniques: nearest neighbor and bestPR. The main difference is in the way the relevant measures are computed.
Figure 3.9. The parser switching method
3.3.3.6. Nearest Neighbor

Nearest neighbor algorithm is a machine learning algorithm from the broader class of instance-based learning. Instance based learning postpones the learning tasks until a new instance must to be classified. Each time a new query instance is encountered, its relationship to the previously stored examples is examined in order to assign a target function value for the new instance.

The advantage of nearest neighbor is that it can focus on each instance and not apply similar learning patterns for all of them. Many techniques construct only a local approximation to the target function that applies in the neighborhood of the new query instance, and never construct an approximation designed to perform well over the entire space.

Table 3.4. Limits and baseline for parser switching

<table>
<thead>
<tr>
<th>Case</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
<th>Exact Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>92.47</td>
<td>91.29</td>
<td>91.87</td>
<td>65.67</td>
</tr>
<tr>
<td>Lower Limit</td>
<td>89.59</td>
<td>87.50</td>
<td>88.53</td>
<td>44.47</td>
</tr>
<tr>
<td>Upper Limit</td>
<td>95.62</td>
<td>95.49</td>
<td>95.55</td>
<td>79.03</td>
</tr>
<tr>
<td>Random</td>
<td>92.87</td>
<td>91.33</td>
<td>92.17</td>
<td>63.94</td>
</tr>
</tbody>
</table>

One disadvantage of instance-based approaches is that the cost of classifying new instances can be high. This is due to the fact that nearly all computation takes place at classification time rather than when the training examples are first encountered.

A second disadvantage of instance-based approaches, especially nearest neighbor approaches, is that they typically consider all attributes of the instances when attempting to retrieve similar training examples from memory. If the target concept depends on only a few of the many available attributes, then the instances that are truly most “similar” may as well be a large distance apart.
The standard nearest algorithm assumes all instances correspond to points in the n-dimensional space $\mathbb{R}$. The nearest neighbors of an instance are defined in terms of the standard Euclidean distance. More precisely, let an arbitrary instance $x$ be described by the feature vector

$$< a_1(x), a_2(x), ..., a_n(x) >$$

Then the distance between $x_i$ and $x_j$ is defined to be $d(x_i, x_j)$, where

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

In nearest neighbor learning the target function may be either discrete-valued or real-valued. Let us consider learning discrete valued target functions of the form $f : \mathbb{R}^n \rightarrow V$, where $V$ is the finite set $v_1, ..., v_s$ and $s$ is number of elements in the set.

The nearest neighbor algorithm for approximating a discrete valued target function is given below. As shown here, the value $rf(x_q)$ returned by this algorithm as its estimate of $f(x_q)$ is just the most common value of $f$ among the $k$ training examples nearest to $x_q$. If we choose $k = 1$, then the 1-Nearest Neighbor algorithm assigns to $f(x_q)$ the value $f(x_i)$ where $x_i$ is the training instance nearest to $x_q$. For larger values of $k$, the algorithm assigns the most common value among the $k$ nearest training examples.

**K-Nearest Neighbor** [44]

Training: for each training example $(c, f(x))$, add the example to the list of training examples.

Classification: given a query instance $x_q$ to be classified, let us denote the $k$ instances from training examples that are nearest to $x_q$.

$$f(x_q) = \arg \max_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.
The nearest neighbor technique applied to parsing consists of picking the parse tree candidate that is closest, in terms of common constituents, to all the other parser trees.

**K-Nearest Neighbor for Parsing**

Training: for each candidate parse tree $t_1, t_2, \ldots, t_n$ produced by a parser from the ensemble of $n$ parsers, map the tree into a bag of constituents and add the bag to the list of training examples.

Classification: select the bag/parser tree

$$t = \max_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = m$, with $m$ denoting the number of common constituents among $a$ and $b$, $f(x_i)$ returns the cardinality of a bag $x_i \in V$.

Each neighbor parse tree is broken onto a bag of constituents and then the similarity between each pair of bags is computed. The similarity in our experiments means the number of common constituents. Other similarity measures may be used as well (see [29] for editing distance). A variant of nearest neighbor algorithm is to weight the contribution of each of the $k$ neighbors according to their distance to the query point $x_q$, giving greater weight to closer neighbors. When a nearest neighbor algorithm stores $n$ examples from which it only considers $k$ when measuring distances among examples it is called local. When all $n$ examples are considered the method is global or Shepards’ method [60]. The difference among them is that the global variant is slower. We have not experimented with a weighted scheme yet.

3.3.3.7. BestPR

The bestPR technique has three main steps: (1) detect the common constituents in all candidate parse trees (the intersection) and consider this set of constituents as a maximum precision set, (2) place all constituents from all parse trees in a set (the union) and consider this as a maximum recall set and (3) pick the parse trees that offer the best precision and recall as compared to the intersection set, respectively union.
set. The rationale of bestPR is that once the number of voters increases unanimity voting (intersection) leads to best precision and single-vote (union) leads to best recall.

Table 3.5. Results for parser switching: nearest neighbor and bestPR

<table>
<thead>
<tr>
<th>Case</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
<th>Exact Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>92.47</td>
<td>91.29</td>
<td>91.87</td>
<td>65.67</td>
</tr>
<tr>
<td>Random</td>
<td>92.87</td>
<td>91.33</td>
<td>92.17</td>
<td>63.94</td>
</tr>
<tr>
<td>Nearest neighbour</td>
<td>93.46</td>
<td>93.86</td>
<td>93.65</td>
<td>72.10</td>
</tr>
<tr>
<td>BestPR</td>
<td>93.13</td>
<td>91.29</td>
<td>92.20</td>
<td>63.42</td>
</tr>
</tbody>
</table>

From Table 3.5 we notice that nearest neighbor provides with the best increase in F-measure while the bestPR technique does not perform as we might expect and this is mainly due to the fact that our classifiers are siblings of the same underlying model. Using nearest neighbor parser switching we managed to increase the exact sentence measure from 65.67% to 72.10%.

3.4. Conclusions

In this chapter we presented a voting scheme to detect incorrect tags in the WordNet glosses.

Our voting scheme is a two-step method: (1) compare the tags of two state-of-the-art taggers (2) compare the tags of one of the taggers with WordNet coarse tags for the disagreement part resulted from step (1). This method leads to a tagging accuracy of 98.50% for 91.57% of words, 100% accuracy for 0.91% of the words. Human intervention was required on 7.52% of words. An overall accuracy of 98.93% is achieved. All these measurement were performed on a subset of WN glosses.

Methods for parser combination were also presented, namely constituent voting, and parser switching. The main disadvantage of using constituent voting is the lack of confidence on the voted structure: some constituents may overlap and thus leading
to a non-tree structure. Simple majority voting offers the best results when using constituent voting, while nearest neighbor offers best results when parser switching is used.
Chapter 4
LOGIC FORM TRANSFORMATION

This chapter describes our experiments in deriving Logic Forms for WordNet glosses following the output of the parser. The approach combines a set of high precision transformations with a set of high recall heuristics. Glosses are parsed and from parse trees grammar rules are extracted. For each grammar rule one or more LF transformations need to be implemented. The number of grammar rules is too large to possibly implement LFTs for all of them. We present a procedure for automated selection of grammar rules that have the largest impact on the performance of LFT; i.e. rules that transform more glosses than others. The selection is based on probabilistic data collected from the entire WordNet. A bottom-up strategy is used to build the logic form around a parse tree by selectively triggering transformations as traversing the tree. When the top of the tree is reached a set of heuristics are applied for each unfilled argument. Tests performed on a subset of glosses show an overall performance of 89%.

4.1. Deriving Logic Forms

The implementation of deriving LFs relies on information provided by the syntactic parser. We have developed a set of transformation rules [25] that create predicates and assign them arguments. For every rule of the parser, we have a transformation rule which produces the corresponding logical form. There are two classes of rules: (1) intra-phrase and (2) inter-phrase transformation rules.

The intra-phrase transformation rules generate predicates for every noun, verb, adjective or adverb. They also assign the variables that describe dependencies local to the phrase.
The *inter-phrase transformation rules* provide the arguments of the verb predicates, preposition predicates and inter-phrasal conjunctions. Verb predicate arguments are identified by recognizing the syntactic subject and object of the respective verb, based on a few grammar rules and relative pronoun interpretation.

Dependencies between adjectival (adverbial) phases and noun (verb) phases are predicted based on vicinity.

Both intra- and inter-phrase transformation rules are produced from the parser. Table 4.1 illustrates examples of transformation rules.

### Table 4.1. Examples of LFT rules

<table>
<thead>
<tr>
<th>Intra-phrase transformation rules</th>
<th>Grammar Rule</th>
<th>Transformation Rule (LFT)</th>
<th>Phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP → DT NN</td>
<td>noun/NN → noun$(x_1)$</td>
<td>(NP (a/DT monastary/NN))</td>
<td></td>
</tr>
<tr>
<td>NP → DT JJ NN</td>
<td>adj/JJ noun/NN → noun$(x_1)$ &amp; adj$(x_1)$</td>
<td>(NP (a/DT short/JJ sleep/NN))</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inter-phrase transformation rules</th>
<th>Grammar rule</th>
<th>Transformation Rule (LFT)</th>
<th>Phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>PP → IN NP</td>
<td>prep/IN noun/NP$(x)$ → prep$(__ x)$ &amp; noun$(x)$</td>
<td>(PP (by/IN (NP an abbot)))</td>
<td></td>
</tr>
<tr>
<td>VP → VP PP</td>
<td>verb/VP-PASS by/PP$(_x)$ → verb$(e, x, _)$ &amp; by$(e, x)$</td>
<td>(VP (ruled/VBN by/PP))</td>
<td></td>
</tr>
<tr>
<td>NP → NP VP</td>
<td>noun/NP$(x_2)$ verb/VP-PASS$(e, x_1, _)$ → noun$(x_2)$ &amp; verb$(e, x_1, x_2)$</td>
<td>(NP (NP ... ))</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(VP ruled/VBN ... ))</td>
</tr>
</tbody>
</table>

78
4.1.1. Tokenizing, Tagging and Parsing

The tokenization, POS tagging and parsing processes were described in the previous chapter. We remind the reader that we use Treebank guidelines for tokenization and combination techniques (two levels voting for tagging, respectively constituent voting and parser switching for parsing) for tagging and parsing.

4.1.2. Transforming the Parse Tree

Before the derivation process starts building up the logic structure by generating arguments and placing them in the right slot (remember the fixed slot allocation), the parse tree is modified in order to simplify the derivation process. Details on this are provided later on in the chapter.

4.1.3. Bottom-Up Strategy and the Derivation Engine

We apply a bottom-up strategy for deriving logic forms: we start from leaves of a parse tree and then climbing up to the top. At each level arguments are generated and/or propagated (placing an argument into a relational predicate that would indicate a relationship with its fellow arguments), as necessary, to achieve a correct logic form when the top is reached.

The leaf level - arguments for content predicates are generated. When this is done the derivation steps up to the upper level and triggers inter-phrases rules associated with the grammar rule derived for each parent-children set of tags. At each level the derivation is designed as to assign as many arguments as possible. When the top is reached either all the arguments are filled out and thus the logic form is printed out or some arguments are missing and a set of default rules are triggered.

Leaf level

The first step of the derivation process identifies all nouns and verbs (modals, auxiliaries are already removed at this point in the transformation process of the tree described in the previous section) and assigns them a new unique argument: of types
Figure 4.2. An example of actions at leaves level

$x$ and $e$ for nouns and verbs respectively. Another major step in this phase is to generate the complex nominals $nn$ for each sequence of noun predicates (see Figure 4.1.3). The first and main argument of predicate $nn$ would be a newly unique $x$ argument that would stand for the compound meaning of the sequence. The rest of the arguments are borrowed or propagated from each noun predicate member. The number of members can be as large as necessary (see Figure 4.1.3).

Lastly the modifiers, adjectives and adverbs, are given the argument of the head of the phrase in which are included. When a modifier stands beside a complex nominal, that modifier needs to be attached to the first argument of the new predicate $nn$.

*Propagate the Arguments*

Once the arguments for content predicates and their modifiers are generated the main task of the engine is to go up the tree, level by level, and fill out the remaining missing arguments by selectively triggering LF transformations that would fill out the slots. An important operation, called *up-propagation* 4.1.3, is to propagate arguments
from lower levels to upper levels. It is unnecessary to up-propagate all the arguments from a lower level phrase to the upper one. We use the head detection rules to select the most important predicate from a lower phrase. The processing at each upper level considers only the arguments of the propagated predicates ignoring all others from subtrees. This head selection simplifies the derivation process as it leads to less transformations.

Each implemented transformation tries to propagate arguments from one predicate to another. This horizontal propagation is at one level, not among levels and not among different subtrees at same level (see Figure 4.1.3).

*Apply Heuristics for Missing Arguments*

The advantage of the proposed approach is that the implemented rules are highly accurate and whenever an argument is assigned one can tell with high precision that is correct. A missing argument indicates that case was not covered by the implemented rules. In other words this approach prefers high precision rules over high recall heuristics.
For the uncovered cases we have two choices: either the user may manually intervene and fill the argument or a set of heuristics can be designed to do the job. We chose to design a set of heuristics to solve the uncovered cases.

The heuristics will assure that every argument slot will be filled but they can not guarantee to find the proper argument for any empty slot. A heuristic was designed for each type of argument: subject, direct object, indirect object, prepositional head and prepositional object, etc.

- **Subject** - previous phrase head argument or if verb is in passive voice the prepositional object argument of the following by preposition
- **Direct object** - first following phrase head argument (or second for ditransitives) or surface subject if the verb is in passive
- **Indirect object** - second following noun phrase head argument (or first) for ditransitives
- **prepositional head** - previous phrase head argument
- **prepositional argument** - following phrase head argument
• *adjective/adverbs* - following noun/verb phrase head argument or previous noun/verb phrase argument if there is none following
• *default* - generate a new argument that does not exist.

4.2. Probabilistic Selection of Rules

As we outlined earlier in the dissertation, one of the most challenging problems for deriving LFs for natural language text is the fact that each grammar rule requires the implementation of one or more interpretation procedures to generate logic forms.

Table 4.2 shows the number of distinct grammar rules needed for the glosses in the entire WordNet. The total number of nearly 10,000 rules that should be implemented is by far too large to possibly implement all of them.

<table>
<thead>
<tr>
<th>Part of speech</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>noun</td>
<td>5,392</td>
</tr>
<tr>
<td>verb</td>
<td>1,837</td>
</tr>
<tr>
<td>adjectives</td>
<td>1,958</td>
</tr>
<tr>
<td>adverbs</td>
<td>639</td>
</tr>
<tr>
<td>Total</td>
<td>9,826</td>
</tr>
</tbody>
</table>

To deal with this coverage problem we devise a procedure in two steps (phases):
• implement first the most common grammar rules;
• adjust the performance by selecting more valuable rules;

4.2.1. Detecting the Rules

The first phase is basically an acquisition phase: from a representative corpus of glosses we derive the most common grammar cases and resolve them. The second step consists of an incremental selection of more rules to boost up the performance.

Although the total number of grammar rules is large, a small number of rules for a specific phrase covers a large percentage of all occurrences. This might be explained
by the relative structural uniformity of glosses: genus and differentia. This relative uniformity is more specific for nouns and verb glosses.

Based on this observation we implement the most common rules first and leave the others uncovered or at least postpone their resolution. No LFT is generated for grammar rules that are not frequent enough. The problem of precision translates in this case is a problem of coverage, i.e. the number of cases covered by the selected rules.

Table 4.3 shows the distribution of most common phrases for 10,000 randomly selected noun glosses, the number of unique grammar rules and the percentage covered by top ten rules. We observe that the coverage of top ten most frequent rules is above 90% for many important phrases.

Table 4.3. Top 10 most frequent rules in 10,000 noun glosses

<table>
<thead>
<tr>
<th>Phrase</th>
<th>Occurrences</th>
<th>Unique rules</th>
<th>Coverage of top ten</th>
</tr>
</thead>
<tbody>
<tr>
<td>base NP</td>
<td>33,643</td>
<td>857</td>
<td>.69</td>
</tr>
<tr>
<td>NP</td>
<td>11,408</td>
<td>244</td>
<td>.95</td>
</tr>
<tr>
<td>VP</td>
<td>19,415</td>
<td>450</td>
<td>.70</td>
</tr>
<tr>
<td>PP</td>
<td>12,315</td>
<td>40</td>
<td>.99</td>
</tr>
<tr>
<td>S</td>
<td>14,740</td>
<td>35</td>
<td>.99</td>
</tr>
</tbody>
</table>

To decide whether a rule is common or not we use an empirical criteria: if it brings a gain in coverage larger than a given threshold then we consider that case to be common. The value of the threshold should be a compromise between the effort spent to implement the rule and the gain in coverage brought by that rule. The value of the threshold in the case of WordNet glosses was empirically set to 1%.

Before determining which cases are most frequent we reduce the number of candidate rules: (1) several grammar rules are treated uniformly (2) complex phrase rules are reduced to simpler phrase rules. We will detail those techniques in the next section for the case of base NPs. At the end of the first stage we have a validated
set of most frequent rules $S_0$ and we denote with $G$ the set of rules uncovered yet. The next phase consists of adding more rules that are frequent enough to justify their selection.

4.2.2. Select more Valuable Rules

The second phase of the procedure consists in selecting more rules to boost up the performance obtained in the first phase. The new rules are determined through an iterative refinement process. The main steps of the second phase are depicted below.

```plaintext
procedure LFT($S_0$, $G$, $\tau$)
{
  $i = 0$; $S = S_0$; $\delta = 0$; $S' = \emptyset$;
  old_perf = 0; apply($S_0$, new_perf);
  $\delta = new\_perf - old\_perf$;
  while ($i \leq$ upper_limit) && ($\delta \leq \tau$)) {
    $S' = most\_frequent\_rules(G - S')$;
    $G = G - S'$;
    $S = S \cup S'$;
    old_perf = new_perf;
    apply($S$, new_perf);
    $\delta = new\_perf - old\_perf$;
    $i = i + 1$;
  }
  return (new_perf);
}
```

At each refinement step, a set of rules $S'$ is added to the set of rules obtained in previous steps $S$ (the initial set is $S_0$ obtained in phase one). The gain in coverage $\delta$ of this set is computed and compared against a threshold $\tau$: if the value of $\delta$ is lower than threshold $\tau$ the process stops. The rules in $S'$ are determined based on
frequency of occurrences. The cardinality of $S'$ determines how fast $\delta$ falls below $\tau$. A larger value would generate a smaller number of refinement steps but more effort spent on rules that bring little benefit, especially for the last steps. At the end of this phase the final set $S$ contains the grammar rules most representative for the target corpus.

Before proceeding to the next section we must emphasize that this iterative process allows a certain degree of flexibility which is beneficial for customizing the method of targeting data other than glosses.

4.3. Experiments and Results

To validate our procedure we made an experiment on a subset of WordNet 1.6 noun glosses.

The initial set of rules is formed by taking the most frequent rules for each grammar phrase detected in a corpus of 10,000 noun glosses randomly selected from the noun data file of WordNet 1.6. We expand, tag and parse them. From the parse trees obtained we extract all grammar rules and their number of occurrences then select the most frequent rules up to the point where the gain in coverage is less than 1%.

Table 4.4. Examples of LFT for determined and undetermined NPs

<table>
<thead>
<tr>
<th>Base NP rule</th>
<th>Example</th>
<th>LFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP $\rightarrow$ NN</td>
<td>(NP (NN sale))</td>
<td>sale(x1)</td>
</tr>
<tr>
<td>NP $\rightarrow$ DT NN</td>
<td>(NP (DT the) (NN sale))</td>
<td>sale(x1)</td>
</tr>
<tr>
<td>NP $\rightarrow$ DT JJ NN</td>
<td>(NP (DT the) (JJ public) (NN sale))</td>
<td>public(x1) &amp; sale(x1)</td>
</tr>
<tr>
<td>NP $\rightarrow$ JJ NN</td>
<td>(NP (JJ public) (NN sale))</td>
<td>public(x1) &amp; sale(x1)</td>
</tr>
</tbody>
</table>

The selection process is preceded by a preprocessing phase aiming at reducing the number of candidate rules and boost up the coverage of the selected ones.
Base NPs and VPs have a coverage around 70% which calls for improvements. A clear distinction is made between base NPs or non recursive NPs and NPs that have another phrase (NP, VP, ADJP or others) as one of its children.

The rationale of treating them separately is three fold: (1) from a semantic point of view, base NPs identify an entity where in contrast with recursive NPs which might involve relations and more complex predcations (2) base NPs represent a large class of NPs: 68% of all NPs as measured on TreeBank’s WSJ parsed data [39] (3) the coverage of the top ten rules is only 69% thus raising the necessity for improvements.

Table 4.5. Examples of rules with prenominal modifiers

<table>
<thead>
<tr>
<th>Base NP rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP → DT JJ NN</td>
</tr>
<tr>
<td>NP → DT VBG NN</td>
</tr>
<tr>
<td>NP → DT VBN NN</td>
</tr>
</tbody>
</table>

Our solution to boost up the coverage above 90% consists in performing a set of tag and parse tree transformations to reduce the number of candidate rules. Two basic techniques are used: (1) tag reduction and (2) transformations of parse trees.

Tag reduction is allowed due to simplifications in notation: (1) determiners are eliminated, (2) plurals are ignored and thus we can replace NNS with NN, (3) proper nouns are treated identically as common nouns and in consequence NNP is changed into NN and (4) everything in a prenominal position plays the function of a modifier.

Examples of rule reduction due to tag reduction are illustrated in Table 4.3. For verbs we ignore tenses: VBG, VBP, VBZ, VBN, VB are all mapped into VB.

Keeping the passive information is important for syntactic role detection and thus we add a new tag VP-PASS to indicate that the head of the VP is passive. Modals and auxiliaries are eliminated and negations are ignored. The set of reductions have a driven rationale each. For example, determined and undetermined base NPs both identify entities that have their own existence and are newly introduced.
Table 4.6. Coordinated base NPs (without determiners and plurals)

<table>
<thead>
<tr>
<th>Base NP rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP → JJ NN CC NN NN</td>
</tr>
<tr>
<td>NP → JJ NN CC NN NN</td>
</tr>
<tr>
<td>NP → JJ NN CC NN</td>
</tr>
<tr>
<td>NP → JJ NN CC NN NN</td>
</tr>
<tr>
<td>NP → JJ NN NN NN CC NN</td>
</tr>
<tr>
<td>NP → JJ NN CC NN</td>
</tr>
<tr>
<td>NP → JJ RB JJ NN CC NN NN</td>
</tr>
<tr>
<td>NP → NN CC NN</td>
</tr>
<tr>
<td>NP → NN CC JJ NN</td>
</tr>
<tr>
<td>NP → NN CC JJ NN</td>
</tr>
<tr>
<td>NP → NN CC NN</td>
</tr>
<tr>
<td>NP → NN CC NN CC NN CC NN</td>
</tr>
<tr>
<td>NP → NN CC NN CC NN</td>
</tr>
<tr>
<td>NP → NN CC NN NN</td>
</tr>
<tr>
<td>NP → NN CC NN</td>
</tr>
<tr>
<td>NP → NN CC NN RB</td>
</tr>
<tr>
<td>NP → NN CC NN</td>
</tr>
<tr>
<td>NP → NN CC NN CC NN</td>
</tr>
<tr>
<td>NP → NN CC NN NN</td>
</tr>
<tr>
<td>NP → NN NN CC JJ NN</td>
</tr>
<tr>
<td>NP → NN NN CC NN</td>
</tr>
<tr>
<td>NP → NN NN CC NN CC NN</td>
</tr>
</tbody>
</table>

88
The rationale of this reduction is that glosses are independent and general statements and there is no case for more processing as in a discourse, where a base NP might corefer with some other entity or might be a nominalisation of an event and thus corefering with that event. A gain in coverage of 11% is obtained by ignoring the determiner. Table 4 shows few equivalent rules that fall under our first assumption. Under this assumption the LF of the gloss of action:#8, which is (the operating part that transmits power to a mechanism), is the same with the LF of (operating part that transmits power to a mechanism). The fourth reduction treats everything in a prenominal position as a modifier. It allows a flat treatment for an entire set of rules as exemplified in 4.5 (where | stands for alternation or exclusive choices).

**VBG as a Premodifier**

Under this assumption the LFT of the gloss of action:n#8, which is (the operating part that transmits power to a mechanism), is the same as the LFT of (small part that transmits power to a mechanism). The role of operating in a prenominal position is that of a modifier and is different from the role it plays in a definition of the form (part operating a mechanism) where operating is an event that has as agent part.

LFT premodifier: operating(x1) & part(x1)
LFT postmodifier: part(x1) & operating(e1, x1, x2)

**VBN as a Premodifier**

The gloss of airbrake:n#2: (a vehicular brake that operates by compressed air) contains the past participle compressed in a premodifier position and we treat it as a modifier. This treatment is different from the imaginary case of (air compressed at high temperature) where compressed is an event whose direct object is air. Thus, we obtain two different representations for compressed in a premodifier and postmodifier position:

LFT premodifier: compressed(x1) & air(x1)
LFT postmodifier: air(x1) & compressed(e1, x2, x1)
Coordinated Base NPs

A special category of base NPs includes *coordinated base NPs*. A coordinated base NP is a sequence of primitive base NPs (NPs that do not include a coordinated conjunction) connected via a coordinated conjunction (and, or). There are 667 occurrences of coordinated base NPs in 247 distinct rules with an average of 2.7 occurrences per rule. To implement all of them would require a tremendous effort not justified by the return. The second technique consists of rearranging the parse trees so that more complex structures are reduced to simpler ones: more complex base NPs are rearranged into simpler ones (see Figure 4.3).

Table 4.7. Examples of rules having prenominal modifiers

<table>
<thead>
<tr>
<th>Base NP rule</th>
<th>Result rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP → DT JJ NN</td>
<td>NNS</td>
</tr>
<tr>
<td>NP → DT VBG NN</td>
<td>NNS</td>
</tr>
<tr>
<td>NP → DT VBN NN</td>
<td>NNS</td>
</tr>
</tbody>
</table>

As a consequence of those transformations the coverage of the top ten most frequent rules for base NPs and verb phrases jumps over 90%. For example for the gloss of *garnierite:*n#1: (a green mineral consisting of hydrated nickel magnesium silicate), the parse tree before and after simplifications are applied is shown in Figure 4.3. The determiner is removed from noun phrase a *green mineral* and the gerund VBG is changed to VB.
A third preprocessing phase is performed in order to ease the later treatment of postmodifiers according to the specifications in Chapter 2. Tuples of the form (ADJP, (JJ, PP)), (ADJP, (RB, JJ, PP)) are detected and the ADJP is replaced by a PP by combining the posmodifier JJ with the preposition PP in order to form a single predicate PPJ. Table 4.8 provides examples of postmodifiers detected in noun.artifact hierarchy.

Once these preprocessings are done, the main selection process is undertaken. At the end of the first phase we have a set of validated most frequent rules, say \( S_0 \). To test the overall performance of the rules found in phase one we have built a corpus of 1000 noun glosses from the artifact hierarchy. We expanded the glosses definitions, ran Brill’s tagger, corrected the tags and parsed them. Finally, we obtained the LFT for each expanded gloss manually.

We ran the set of rules in \( S_0 \) on our test data and compared the output with the LFTs obtained by hand. A coverage of 72.5% was achieved. We apply then the algorithm in phase two to boost up the performance on the 1000 glosses. The number
of rules picked up in a refinement step is 3. The threshold of 1% is reached in 7 steps for the 1000 WordNet glosses. The chart in Figure 2 shows the evolution of the improvement in accuracy $\delta$ achieved as the refinement number of steps increases. We stop at step 8 when the accuracy is 83% and the gain obtained at the last iteration on the 1000 glosses was less than the established threshold.

The results obtained are encouraging. One can get a better performance by playing with the two parameters ($\tau$, cardinality of $S'$) at the cost of a larger effort. After the selected rules where applied to the test set of 1,000 glosses for each unfilled argument, the corresponding heuristic was applied leading to an overall performance of 1,000 glosses.

4.3.1. Observations on Verb Related Rules

When deriving the logic form for verbs one should pay attention to some syntactic structures specific to the way verbs combine. We refer mainly to sequences of verbs when a verb introduces or prepares the following verb, e.g. (used for eating and lighting) from gloss of coalgrese:n#1, (used to volley) from gloss of badminton:n#1) and to verbs in coordinations such as (detect and counteract) from gloss of countersabotage:n#1.
4.3.2. Verb Sequences

When a verb precedes another verb or a series of verbs (in a coordination) we call it a *sequence*.

There is a special relation among the first verb and the following ones: they share their arguments. Example of such sequences are “keep firing”, “used to apply”.

<table>
<thead>
<tr>
<th>Postmodifiers</th>
<th>Synset</th>
</tr>
</thead>
<tbody>
<tr>
<td>free from</td>
<td>achromatic_lense:n#1</td>
</tr>
<tr>
<td>imitative of</td>
<td>human_vox:n#1</td>
</tr>
<tr>
<td>next to</td>
<td>index:n#5</td>
</tr>
<tr>
<td>available to</td>
<td>menu:n#3</td>
</tr>
<tr>
<td>available for</td>
<td>cheap_money:n#1</td>
</tr>
<tr>
<td>available at</td>
<td>menu:n#1</td>
</tr>
<tr>
<td>necessary for</td>
<td>cabin_cruiser:n#1</td>
</tr>
<tr>
<td>different from</td>
<td>window:n#7</td>
</tr>
<tr>
<td>capable of</td>
<td>buffered_aspirin:n#1</td>
</tr>
<tr>
<td>emblematic of</td>
<td>union:n#10</td>
</tr>
<tr>
<td>active against</td>
<td>antymycin:n#1</td>
</tr>
<tr>
<td>derivable from</td>
<td>antibiotic:n#1</td>
</tr>
<tr>
<td>relative to</td>
<td>guideline:n#1</td>
</tr>
<tr>
<td>equivalent to</td>
<td>megaton_bomb:n#1</td>
</tr>
<tr>
<td>ready to</td>
<td>blank:n#4</td>
</tr>
</tbody>
</table>

The first example is from gloss of *automatic_gun:n#1*: (a pistol that will keep firing until the ammunition is gone or the trigger is released). In this case *keep* and *fire* are both in active voice and they share the subject *pistol*.

The second example is from gloss of *Petri_dish:n#1*: (a shallow dish used to culture bacteria). Here, we can see that *used* is in passive voice while *culture* is in active voice. Both verb share an *impersonal/unspecified* subject (*use* has as direct object *dish*).
4.3.3. Coordinated Verbs

A very frequent case that has a special kind of behavior is represented by coordinated verbs.

In 1000 verb glosses there were 31 verb phrase coordinations (and 11 verb coordinations, VB CC VB, not discussed here). In 1000 noun glosses there were 25 verb phrase coordinations (and 4 verb coordinations, VB CC VB, not discussed here).

There are problems with the assignment of syntactic roles to both types of verbs as sometimes they share them while in other cases they do not, the referent of a relative pronoun may play different roles for different verbs in a coordination.

We illustrate this shortly for several cases that exhibit some interesting peculiarities and require special treatment. The solutions are based on a large spectrum of features such as type of preposition, clause type and lexical information which is very demanding.

The usual case

The tree in Figure 4.3.3 presents the usual case of the VP coordination where each verb, follow and take, has its own direct object and the subject is not specified. The example is the gloss of go, proceed, move: (to follow a procedure or take a course).
Subject

For the previous particular case a common subject for both verbs in the coordination was generated. In the next example this is not the case. The subject introduced by the preposition by is not shared among the two verbs of the coordinations. The referent aircraft of the relative pronoun plays different roles for fix and power, that is subject and direct object. In other words, the usual case of two verbs in coordination sharing same subject does not apply anymore in the gloss (an aircraft that has fixed a wing and is powered by propellers or jets) of synset {airplane, aeroplane, plane}.

The example (an antiarrhythmic drug that has potentially fatal side effects and is used to control serious heart rhythm problems only when safer agents have been ineffective) from gloss of {amiodarone, Cordarone} exhibits the same case of different roles for the referent with the modifications that the subject of the second verb is not explicitly introduced with the help of the preposition by.

Direct object

In gloss (take and maintain control over) of synset {hold} (sense 23) the verbs take and maintain should share the direct object control. However in gloss (to divide into triangles or give triangular form to) of synset triangulate: #1 the direct object “triangular form”, of the second verb should not be shared. In (a drug that prevents or alleviates nausea and vomiting), the gloss of synset {antiemetic, antiemetic drug}, the direct object “nausea and vomiting” needs to be shared.

A more problematic case is one in which the direct object of the second verb is an entire sentence. In this particular case, see gloss (cooperate or pretend to cooperate) of synset {play along, go along}, the direct object should not be shared and this is because the semantic interpretation of something like cooperate to cooperate does not make sense.

If it were for another verb instead of cooperate it would make sense. In the gloss (prevent from doing something or being in a certain state) of synset {prevent, keep}, the direct object of the verb doing should not be shared.
Indirect objects

The indirect objects attached to the singular verbs involved in a coordination sometimes should be shared sometimes not.

Let us consider the case of the gloss (resign or retire from a position) from synset \{leave office, quit, step down\} where the two verbs should share their indirect object a position.

In the next example (relax or remove controls of) from gloss \{decontrol\}, the unspecified indirect object introduced by \textit{of} should not be shared. In (bring into existence or make available by vote), the gloss of \{vote\} (sense 5), is not clear whether \textit{by} needs to be attached to \textit{make} only or to both verbs. In (a drug used to reduce stress or tension without reducing mental clarity), the gloss of synset (tranquilizer, tranquilizer, tranquilliser, antianxiety agent, ataractic drug, ataractic agent, ataractic), is not clear whether \textit{without} should introduce the indirect object to the coordination or only to the last verb.

Modifiers

Sometimes the modifiers of a singular verb in a coordination should be shared and sometimes not. Here is an example (to declare or make legally valid) from \{validate, formalize, formalise\}) where \textit{legally} need to be shared. And in (discard or do away with) (cashier: \#1) is where \textit{away} should not be shared.

A confusing example is the following one (cause to be no longer approved or accepted) from gloss \{decertify, derecognize\}: where it is not clear whether \textit{longer} needs to be shared or not. This ambiguity of sharing of objects for coordinated verbs is very frequent and we present below even more complex situations where is hard to decide which object belongs to which verb.

More complex cases

The gloss (to carry out or participate in an activity) of synset \{prosecute, engage, pursue\} presents a situation in which the indirect object of \textit{participate} should be shared with \textit{carry out} as a direct object for it. In other words, the same NP, \textit{an activity}, can play different syntactic roles for different verbs in a coordination. In
declare or make legally valid, the gloss of synset \{validate, formalize, formalise\}, the
direct object is missing. It is not yet clear whether or not it needs to be introduced.
And legally valid should modify the coordination in this case. In the next case from
gloss of \{ash-pan\} (a receptacle fitted beneath the grate in which ashes collect and are
removed) illustrates a case where the subject of the first verb is the direct object of
the second. The main purpose of all previous examples is to illustrate the complexity
of the task of assigning roles to verbs in coordinations. Next we describe the solution
we adopted.

Proposed Solution

Due to the diversity of cases we opted for an uniform solution: whenever a
coordination is involved the arguments of the verbs are shared unless an argument
is available. For example for the case of (a drug that prevents or alleviates nausea
and vomiting) the logic form is drug:n(x1) & prevent:v(e1, x1, x2) & or(e2, e1, e3)
& alleviate:v(e3, x1, x2) & and(x2, x3, x4) & nausean(x3) & vomiting:n(x4). Note
that the subject - x1 and the direct object - x2 are shared by both verbs in the
coordination.

4.4. Learn Ambiguous Rules

One main problem in deriving logic forms for glosses is the fact that the cor-
responding logic form transformations (LFT) for each grammar rule are manually
developed. The LFT takes care of assigning the arguments for predicates according
to the structural information embedded in the grammar rule and a few extra features.
We present here how machine learning techniques can be applied to verbs involved in
simple grammar rules as: \( VP_1 \rightarrow VP_2 PP \) where there is an ambiguity regarding the
syntactic role that the object of the preposition plays for the verb: syntactic subject
when \( VP_2 \) is passive and the preposition is 'by' or indirect object when the verb is
active and the preposition is not-'by'. For such simple cases a perceptron having as
input few syntactic (voice of verb) and lexical features ('by' or not 'by') and as output
two values interpreted as syntactic subject or indirect object is suitable.
4.4.1. The Perceptron

The perceptron is a type of Artificial Neural Networks (ANN) having a single cell. A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise. Given inputs $x_1, x_2, ..., x_n$, the output computed by the perceptron is

$$o(x_1, x_2, ..., x_n) = \begin{cases} 
1 & \text{if } w_0 + w_1 x_1 + ... + w_n x_n > 0 \\
-1 & \text{otherwise}
\end{cases}$$

The learning problem is to determine values for the weights that causes the perceptron to produce correct output values for each of the given training examples.

We use the perceptron training rule which updates the weights using the rule:

$$w_i = w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

where $\eta$ is a positive constant called the learning rate, $t$ is the target output for current training example and $o$ is the output generated by the perceptron.

4.4.2. A Perceptron for Modelling Syntactic Roles

Let us consider the rule of $VP \rightarrow VB \ PP$ in gloss of `pneumatic_hammer:n#1`, respectively `air_intake:n#1`.

The gloss of `pneumatic_hammer:n#1` is (a hammer driven by compressed_air) and the corresponding parse tree is given in figure 4.4.2. The most challenging task
is to assign the arguments of the verb predicate *drive*. The prepositional object *compressed_air* plays the role of subject for the verb in passive *driven*. We can determine that by looking at the *voice* of the verb and preposition *by*. The corresponding logic form transformation of the gloss is: \( \text{hammer}(x_1) \land \text{drive}(e_1, x_2, x_1) \land \text{compressed_air}(x_2) \).

For the case of gloss of *air_intake*:\#1 which is (a duct that admits air to be mixed with fuel) and the parse tree is given in figure 4.4.2. The corresponding logic form is: \( \text{duct}(x_1) \land \text{admits}(e_1, x_1, x_2) \land \text{air}(x_2) \land \text{mix}(e_2, x_3, x_2, x_4) \land \text{with}(e_2, x_4) \land \text{fuel}(x_4) \) in which the prepositional object *fuel* plays a role of indirect object for verb *mix*.

In both examples we use two main features: the voice of the verb and the fact that the preposition is *by* or *not-by*. Another implicit feature is the prepositional attachment provided by the parser. It is not explicitly specified because the perceptron we are developing is associated with a grammar rule which already embeds this kind of information. To handle these two different cases and others we design a perceptron having two inputs associated with two features:

- **voice of the verb**: 0 - the verb is active, 1 - the verb is passive
- **type of preposition**: 0 - the preposition is *not-by*, the preposition is *by*
Figure 4.10. Parse tree for gloss of air intake:n#1

Figure 4.11. A perceptron with two inputs
The output of the network establishes the syntactic role, in other words what argument position the prepositional head will fulfill: 1 for subject, -1 for indirect object.

4.4.3. Results

We experimented with the following four grammar rules for which we defined few features presented later

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

Table 4.4.3 presents the size of training set, the size of the test set, the weights obtained and accuracy of the perceptron on the test sets containing up to 4% noise. The training and test sets were derived from parse trees of 1000 WordNet glosses.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Training Set Size</th>
<th>Test Set Size</th>
<th>Weights</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP \rightarrow VB PP</td>
<td>180</td>
<td>21</td>
<td>(-1.4; 1.1; 1.1)</td>
<td>95.3%</td>
</tr>
<tr>
<td>VP \rightarrow VP PP PP</td>
<td>199</td>
<td>28</td>
<td>(-1.6; 1.5; 1.3)</td>
<td>96.5%</td>
</tr>
<tr>
<td>NP \rightarrow NP PP</td>
<td>180</td>
<td>23</td>
<td>(-1.4; 1.3; 1.3)</td>
<td>95.7%</td>
</tr>
<tr>
<td>S \rightarrow NP VP</td>
<td>172</td>
<td>21</td>
<td>(-1.4; 1.2; 1.2)</td>
<td>95.3%</td>
</tr>
</tbody>
</table>

Preparation of the two sets required a preprocessing phase consisting of extracting the definitions from glosses, extend them to full sentences, tokenize and POS tag them. After the parser was run on them we checked and manually corrected the parsed glosses so that the training size is free of errors.

The tagging was done with the two-levele voting method between the two taggers: MXPOST and Brill’s tagger and also with WordNet categories.
Figure 4.12. The decision tree equivalent with the two inputs perceptron

4.4.4. Comparison with other Methods

Because the training sets are linearly separable on the two input features, the resulted neural network is equivalent with an AND function and with the decision tree given in figure 4.4.4.

The decision tree can be mapped into the following IF-THEN rules which are the learned rules that use few extra features based on lexico-syntactic aspects of each rule:

IF ((verb=PASSIVE) AND (prep=’by’))
     THEN syntactic role = subject
IF ((verb=PASSIVE) AND (prep=not-’by’))
     THEN syntactic role = indirect object
IF ((verb=ACTIVE) AND (prep=’by’))
     THEN syntactic role = indirect object
IF ((verb=ACTIVE) AND (prep=not-’by’))
     THEN syntactic role = indirect object

The perceptron model presented for learning syntactic roles for the derivation of LF for WordNet glosses yields high accuracy for glosses even in the presence of noise. The model can be extended to free text easily, but more features need to be added in order to cover other cases. For example, our two-input perceptron would
assign a wrong subject role for Monday in the sentence A package was delivered by Monday as opposed to A package was delivered by the university. To be able to correctly cover such an example a new feature need to be added: the semantic class of prepositional object, that is if the prepositional head is an agent (i.e. a hyponym of agent:n#1 in WordNet) it plays the role of subject, otherwise it is indirect object or temporal adverbial phrase. The problem of detecting syntactic roles for verbs can be described in a more general way: given a verb and a prepositional head in a gloss, one needs to design an ANN that detects if there is any syntactic relations among the two. We see three main features that need to be used: attachment, i.e. the prepositional head is attached via a preposition to the verb - this information is embedded in the parse tree and we use it in the form of grammar rules, the voice of the verb and the type of preposition. We prefer to model the problem using only two features because we consider this way to fit better in our rule-based approach to the LF problem. Furthermore, for free text the general way to approach the issue of detecting syntactic roles would require four features: attachment from parse tree, voice of the verb, type of preposition, semantic category of the prepositional head.

4.5. Scalability

We show that a statistical approach for logic form derivation is scalable, flexible and easy to implement and maintain and compare it with an incremental approach. Results on a data set of 4,000 glosses are reported.

4.5.1. Performance Measures

Before discussing different approaches, two performance measures are defined. Each is more useful than the other in some context.

First, we define predicate level performance as the number of predicates with correct arguments divided by the total number of predicates. This measure gives a more fine-grained look at the derivation process and illustrates the power of a method without considering the application which uses the glosses.
Gloss level performance is defined as the number of full glosses correctly transformed into logic forms divided by the total number of glosses attempted. This new measure catches contextual capabilities of a method in that it gives an idea of how well a method performs at gloss level. It is a more appropriate measure when one tries to see the impact of using full glosses in logic forms to applications such as Question Answering.

4.5.2. Approaches

We discuss few different approaches to derive logic forms and provide a comparison of them.

4.5.2.1. Statistical approach

In a statistical approach, one tries to identify a set of rules that are representative for glosses and lead to same performance measure on an increasing number of glosses. The size of the representative set should be small enough so that the costs of extending and maintaining the system are not high but large enough so that the performance on glosses is as high as possible.

To identify most representative rules, the phrasal structure of entire WordNet is analysed. We parsed the glosses, detected all rules, sorted them on their frequency and picked most frequent ones for each phrase: S, NP, VP, etc. First we pick the top 10 most frequent rules for each phrase. Then, we choose rules in increments of 3 up. We stopped when a rule appears in such a small number of glosses that it is faster to solve those glosses by hand than to implement a rule for them. It is faster to do it manually because most of the gloss is already in logic form and the user has to solve only one or two arguments. We identified 81 rules (133 siblings). The process of selecting the rules is described at the beginning of the chapter and it contains two main steps: (1) reduce number of candidate rules (2) select most frequent ones.

From Table 4.5.2.1 and Figure 4.13 we see that the performance is stable for an increasing number of glosses. There is no need of extra effort (to implement new rules)
Figure 4.13. Performance of Statistical Approach

Table 4.10. Performance of Statistical Approach

<table>
<thead>
<tr>
<th>Nr of Gosses</th>
<th>Nr of incomplete gosses</th>
<th>Gloss-level Performance(%)</th>
<th>Predicate-level Performance (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>16</td>
<td>85.00</td>
<td>96.6</td>
</tr>
<tr>
<td>200</td>
<td>35</td>
<td>83.5</td>
<td>96.0</td>
</tr>
<tr>
<td>500</td>
<td>95</td>
<td>82.00</td>
<td>95.6</td>
</tr>
<tr>
<td>1000</td>
<td>178</td>
<td>83.2</td>
<td>96.4</td>
</tr>
<tr>
<td>2000</td>
<td>354</td>
<td>82.8</td>
<td>95.7</td>
</tr>
<tr>
<td>3000</td>
<td>454</td>
<td>85.2</td>
<td>95.6</td>
</tr>
<tr>
<td>4000</td>
<td>496</td>
<td>87.6</td>
<td>96.8</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>84.18</td>
<td>96.1</td>
</tr>
</tbody>
</table>
for larger sets of data in order to keep the performance at the same level (as opposed to the incremental approach discussed later). The average performance measure is 84.18%.

The small size of the set of rules and no effort necessary when going to a larger number of glosses are two major advantages of this approach. This approach is also flexible in that when a bigger or new WordNet is released a simple statistical analysis of gloss phrasal structure is necessary to detect most valuable rules and develop transformations for them.

The main drawback of the statistical approach is that the performance is not 100%. Gloses with empty arguments can be solved manually by human intervention.

If one quantifies the cost of editing a gloss in terms of how many arguments it needs to fill then because most of glosses have only one or two missing arguments the cost to manually correct them is not very expensive. Only 3.9% of predicates have one or more missing arguments. The number of glosses to be checked is about 16% which for a test set of 4,000 means 640 glosses. This can be done in a couple of days (most of the gloss is already in logic form and the user only needs to fill in the missing arguments).

4.5.2.2. Heuristic-enhanced Statistical Approach

One can avoid the human intervention by filling empty arguments with a set of heuristics tailored to which type of argument: subject, direct object, etc. This approach derives logic forms for glosses first using the set of rules determined as described by the statistical approach and then by applying a set of heuristics. The advantage of using heuristics is in many cases the user only has to approve the logic form of the gloss and no editing is necessary. For example, the logic form of gloss (a building where animals are butchered) attached to synset {abattoir, butchery, shambles, slaughterhouse} is generated as:

\[
\text{building} (x2) \land \text{where} (e5, x2) \land \text{animal} (x3) \land \text{butcher} (e5, \ldots, x3)
\]
and by adding heuristics the missing argument is automatically filled as indicated below:

\[
\text{building}(x_2) \land \text{where}(e_5, x_2) \land \text{animal}(x_3) \land \text{butcher}(e_5, x_4, x_3)
\]

The user only has to acknowledge this gloss and no editing is necessary as would be the case if it were not for the heuristics. The acknowledgement is equivalent to 0 editing operations. For 1,000 glosses the heuristics provide correct arguments for 5.7% of glosses for which a user does not have to edit anything.

**4.5.2.3. Incremental approach**

This is the most straightforward approach. It starts with a small set of glosses and derives logic form transformations for all grammar rules in parse trees. Then, the set of glosses is increased, and new transformation are derived for new rules.

**Table 4.11. Number of rules in an increasing number of glosses**

<table>
<thead>
<tr>
<th>Nr of glosses</th>
<th>Nr of unique rules</th>
<th>Total Nr of rules</th>
<th>Average rules per gloss</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>184</td>
<td>761</td>
<td>7.61</td>
</tr>
<tr>
<td>200</td>
<td>261</td>
<td>1589</td>
<td>7.94</td>
</tr>
<tr>
<td>500</td>
<td>452</td>
<td>4077</td>
<td>8.15</td>
</tr>
<tr>
<td>1000</td>
<td>630</td>
<td>7900</td>
<td>7.9</td>
</tr>
<tr>
<td>2000</td>
<td>784</td>
<td>15863</td>
<td>7.93</td>
</tr>
<tr>
<td>3000</td>
<td>862</td>
<td>24311</td>
<td>8.10</td>
</tr>
<tr>
<td>4000</td>
<td>997</td>
<td>33501</td>
<td>8.37</td>
</tr>
</tbody>
</table>

The main goal of this approach is to generate perfect logic forms for all glosses. The number of distinct rules that need to be implemented as a function of number of glosses is given in Table 4.14 and Figure 4.5.2.3. The numbers in this table are collected from 4,000 correctly parsed noun glosses.

The main drawback of this approach is its lack of scalability. As the number of glosses increases so does the number of rules and thus for larger data sets a larger
Figure 4.14. Number of rules necessary with an incremental approach

effort is necessary to keep the same performance. In addition, when the number of rules implemented is large adding a new rule is very expensive. This is due to conflicting rules and rules that behave differently in different contexts. An example is $S \rightarrow NP \ VP$ which when derived for a set of glosses with verbs in active voice needs only one logic form transformation.

When new glosses with verbs in passive voice are added a sibling of the main rules that deals with passive voice is required. VP in passive voice indicates that the previous NP is a direct object instead of subject. An estimated 1/2 man work day is needed when trying to add a new rule. To integrate 997 rules (as many as there are in 4,000 glosses) would take an estimated 498 days or almost 2 years. When a new rule has several siblings the amount of time spent it is larger. The cost of maintaining such a large software system with 1,000 rules is tremendous. And when considering that in all noun glosses there are an estimated few thousand different rules the amount of time necessary to implement them is of order of years.

Additionally, this approach is not flexible. When new data appears all the rules need to be checked against new cases and possible rule variations adopted. New
rules must be integrated into the system and make sure there are no conflicts. Those operations are very expensive.

4.5.3. Comparison

Table 4.5.3 illustrates the main characteristics of each of the approaches described. The main difference between the heuristic-enhanced and the statistical approach regards the time spent to manually fill empty slots: the heuristics enhanced is faster because for some arguments there is no editing necessary. The incremental approach is expensive in terms of developing and maintenance and lacks flexibility.

Table 4.12. Comparison of main approaches for 4,000 glosses

<table>
<thead>
<tr>
<th>Approach</th>
<th>Rules</th>
<th>Flexibility</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistical</td>
<td>81</td>
<td>Yes</td>
<td>low</td>
</tr>
<tr>
<td>Statistical + Heuristics</td>
<td>81 + 9</td>
<td>Yes</td>
<td>low (faster)</td>
</tr>
<tr>
<td>Incremental</td>
<td>997</td>
<td>No</td>
<td>high</td>
</tr>
</tbody>
</table>

If one quantifies the development time using the number of rules the statistical approach is much less expensive even if 3.9% predicates need to be manually checked. This is doubled by the fact that when a rule based system reaches thousands of rules it is very expensive to maintain and extend.

4.6. Conclusions

Here we presented our ongoing work on deriving Logic Form Transformation for WordNet glosses, as part of the Extended WordNet project. To obtain the LFT we follow the output of a state of the art parser. Grammar rules are extracted from parse trees and LFTs are implemented for the most frequent ones. The main performance measure of this approach translates into a coverage measure, that is
how many grammar rule occurrences are covered by the selected rules. We presented a procedure in two phases to solve the coverage problem and obtained an overall coverage of 83% on 1000 glosses. If the set of heuristics is also applied on same test set of 1000 glosses the overall performance reaches 89%. The parser’s accuracy and irregularities in some WordNet glosses are the main sources of errors in our approach.
Chapter 5
BRACKETING COMPOUND NOUNS

This chapter includes a method for bracketing coordinated compound nouns in the larger context of deriving first order logic forms. The method consists of two phases: (1) detection - extract candidate compound nouns from a parse tree and (2) interpretation - provide a bracketing solution for each candidate. Bracketing performance on 525 coordinations is 87.42%. This compares with a 52.35% baseline precision when a default split by the coordination is applied [58].

5.1. Introduction

Our approach [49] to derive the logic form is to use structural information available in a syntactic tree [48]. Compound nouns are sequences of nouns that have an enhanced meaning together, which is occasionally different from their individual meanings. An example is goat hair where the two nouns refer to the hair of goat which is different though related to the general concept of hair and different from the goat concept. A coordinated compound noun is one or more compound nouns linked via a coordinated conjunction as in goat hair and camel hair. The usage of coordinated compound nouns features a language laziness in the form of an ellipsis: goat and camel hair.

Figure 5.1 shows two possible bracketing alternatives and their corresponding logic form for the previous coordinated compound noun: goat and camel hair. There is a larger range of alternatives to choose from when modifiers are involved, e.g. (common house and field crickets) from gloss of Acheta.m#1 leads to three different bracketing possibilities. The paper presents a method to bracket coordinated compound nouns for further deriving logic form.
Figure 5.1. Coordinated Compound Nouns and their Logic Forms

When bracketing coordinated compound nouns, several types of problems are encountered. Those errors can be classified in two distinct classes: detection of coordinated compound nouns and interpretation of coordinated compound nouns.

5.1.1. Detection Errors

A noun phrase having a sequence of tags of the form “NN[PS] CC NN[PS] NN[PS]” after the elimination of determiners and modifiers does not always lead to a coordinated compound noun ([] means alternative extensions to capture plurals NNS and proper nouns NNP).

The distinction between real coordinated compound nouns and false cases is not obvious.

- **POS Tag errors** Due to part of speech (POS) tag errors the next sequence of tags can be incorrectly viewed as a possible coordinated compound noun: “a/DT bench/NN and/CC press/NN weights/NNS” (from the gloss of bench press:n#1: (a weightlifting exercise in which you lie on your back on a bench and press weights upward)). In this example press is wrongly tagged as an NN.

- **Compound Concepts** A coordination may contain a simple compound concept that is present in WordNet. E.g.: “NNP Mississippi CC and NNP Great NNPS Lakes” from the gloss attached to rock_bass:n#1 includes the compound concept Great_Lakes. An entire pattern may form a single concept,
e.g. the concept Health and Human Services:n#1 in gloss (the position of the head of the Department of Health and Human Services) of Secretary of Health and Human Services:n#1.

- Parse errors The parser may wrongly detect the base NP that includes the coordinated compound noun, either as a consequence of a tag error or a pure parsing error. Parsing errors are more frequent when modifiers surround the coordinated compound noun. An example is other/JJ colors/NNS and/CC a/DT cue/NN ball/NN which is wrongly considered a base NP. This example is from the gloss of snooker:n#1: (form of pool played with 15 red balls and six balls of other colors and a cue ball) where we can see that there is an and among three different NPs: “(NP 15 red balls)” and “(NP six balls of other colors) and (NP a cue ball)”.

In addition to detection errors there are interpretation errors that appear after correctly detecting coordinated compound nouns.

5.1.2. Interpretation Errors

Interpretation errors are strongly linked to bracketing. For example, the coordinated compound noun “hair/NN or/CC finger/NN nails/NNS” from gloss (scissors for cutting hair or finger nails) of clipper:n#4 should be bracketed as (NP (NN hair)) (CC and) (NNS finger, nails)) and its logic form would be: [and(x1, x2, x3) & hair(x2) & finger, nail(x3)] while (peach/NN or/CC almond/NN trees/NNS) in gloss (willow of the western United States with leaves like those of peach or almond trees) of peachleaf willow:n#1 should be bracketed as (NP (NP (NN peach) (CC and) (NN almond))) (NP (NNS trees))) and its logic form would be: [and(x1, x2, x3) & peach(x2) & almond(x3) & mn(x4, x1, x5) & tree(x5)].

Table 5.1 provides alternative bracketings and the corresponding logic form for both examples. The last column indicates the truth value of the bracketing: true or false. The synset in which the coordination can be found has been omitted for space availability reasons.
5.2. Bracketing Coordinated Compound Nouns

To derive the logic form for coordinated compound nouns we designed our own algorithm that takes advantage of POS tagging, structural information from a parser, the semantic sense of nouns involved, and type of coordinated conjunctions. It consists of two steps: (i) provide a bracketing for the coordinated compound noun (ii) apply the rule-based approach presented in [48]. In turn, the bracketing consists of two steps: (1) detection and (2) interpretation.

5.2.1. Coordinated Compound Nouns Detection

In the following, we use a simplified frame: \( n_1 \text{ cc } n_2 \ h \), where \( n_1 \) is the first noun, \( n_2 \) is the second noun and \( h \) is the head noun in a coordinated base NP of the form NN CC NN NN.

Table 5.1. Alternative bracketing solutions and their logic forms

<table>
<thead>
<tr>
<th>Coord</th>
<th>Bracketing</th>
<th>Logic Form</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. hair/NN</td>
<td>1.1 ( (NP (NP (NN hair)) \text{ (CC and)} (NP (NN finger) (NNS nails))) )</td>
<td>[ \text{and}(x1,x2,x3) &amp; \text{hair} : n(x2) \text{nn}(x3,x4,x5) &amp; \text{finger} : x4 &amp; \text{nail} : x5 ]</td>
<td>No</td>
</tr>
<tr>
<td>or/CC finger</td>
<td>1.2 ( (NP (NP (NN hair)) \text{ (CC and)} (NP (NN finger)) (NP (NNS nails))) )</td>
<td>[ \text{nn}(x1,x2,x3) &amp; \text{hair} : n(x4) \text{and}(x2,x4,x5) \text{finger} : x5 &amp; \text{nail} : x3 ]</td>
<td>No</td>
</tr>
<tr>
<td>nails/NNS</td>
<td>1.3 ( (NP (NN hair)) \text{ (CC and)} (NNS finger, nails) )</td>
<td>[ \text{and}(x1,x2,x3) &amp; \text{hair} : n(x2) \text{finger, nail} : x3 ]</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>2. peach/NN or/CC almand/NN trees/NNS</td>
<td>[ \text{peach}(x1) &amp; \text{and}(x2,x1,x3) &amp; \text{nn}(x3,x4,x5) &amp; \text{almond}(x4) &amp; \text{tree}(x5) ]</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>2.1 ( (NP (NP (NN peach)) \text{ (CC and)} (NP (NN almand) (NNS trees))) )</td>
<td>[ \text{and}(x1,x2,x3) &amp; \text{peach}(x2) &amp; \text{almond}(x3) &amp; \text{nn}(x4,x1,x5) &amp; \text{tree}(x5) ]</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>2.2 ( (NP (NP (NN peach)) \text{ (CC and)} (NN almand)) (NP (NNS trees))) )</td>
<td>[ \text{and}(x1,x2,x3) &amp; \text{peach}(x2) &amp; \text{almond}(x3) &amp; \text{nn}(x4,x1,x5) &amp; \text{tree}(x5) ]</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>2.3 ( (NP (NP (NN peach)) \text{ (CC and)} (NP (NNS almand, trees))) )</td>
<td>[ \text{and}(x1,x2,x3) &amp; \text{peach}(x2) &amp; \text{almond, tree}(x3) ]</td>
<td>No</td>
</tr>
</tbody>
</table>
As we have previously seen, there are many factors that influence the accuracy of detecting coordinated compound nouns.

5.2.1.1. WN-based Named Entity

Named Entity (NE) detection plays an important role in detecting coordinated compound nouns. NEs appear in coordinated base NPs in three forms: (i) simple (ii) coordinated and (iii) all-way. Simple NEs are in the form of a two word concept in coordination with another concept with which it does not share the head such as Cannot Islands in “Spain/NNP and/CC Canary/NNP Islands/NNPS” from the gloss (any of various light dry strong white wine from Spain and Canary Islands) of the concept sack:n#4. Coordinated NEs share the head as in “North/NNP and/CC Central/NNP America/NNP” from gloss (tropical marine bivalve found chiefly off eastern Asia and Pacific coast of North and Central America) attached to concept pearl_oyster:n#1. An example of all-way coordinated base NP is Health and Human Services:n#1.

We treat NE detection separately from compound concept detection due to practical reasons. NEs are easily detected by the presence of an uppercase first letter and then by acknowledging its existence in WordNet as a concept. To distinguish among the three types presented before we form tentative concepts $n_1 h$, respectively $n_2 h$ and check their existence in WordNet. For example, for Spain and Canary Islands we form Spain Islands and Canary Islands. Since only Canary Islands exists in WordNet, we classify the coordination as of type (ii).

5.2.1.2. Compound Concept Detection

Undetected common compound concepts may also lead to false coordinated compound noun detection. For example in gloss of water_gas:n#1, hydrogen/NN and/CC carbon/NN monoxide/NN can be misinterpreted as a coordinated compound noun unless one identifies carbon_monoxide as a single concept in WordNet. On the other hand for the base NP bond/NN or/CC stock/NN shares/NNS from gloss of
flotation:n#2, stock_shares can be considered as a single concept [45] in some specific domains, whereas generally speaking it is not. Our option would be to go for the general case that can be easily tested by checking whether stock_shares has an entry in WordNet or not. We form pairs of concepts n1 h and n2 h and look them up in WordNet. This test works only inside base NPs and not for every sequence of two or more words. Here is a counter example: covering/VBG designed/VBN to/TO be/VB worn/VBN on/IN a/DT person/NN 's/POS body/NN which is the gloss of clothing:n#1 were worn_on is identified as a WordNet concept and it should not be considered for the case of this gloss. We treat common compound nouns separately from NEs because of future extensions of this work in which we envision using some dictionaries of NEs to compensate for the incompleteness of WordNet in this aspect. Extensions of WordNet with other common compound concepts is a more challenging task. The reader is referred to [45] for more details.

5.2.2. Coordinated Compound Nouns Interpretation

At this point the possible compound concepts are identified and the main task of this phase is to provide a correct bracketing for the coordinated base NP. To provide the right bracketing we apply few heuristics in the same order as described below.

The first heuristic deals with a special case when \( n_1 = n_2 = n \).

**Heuristic 1**: if \( (n_1 = n_2) \) then \( bracket (n_1 \ cc \ (n_2 \ h)) \)

**Rationale.** The presence of the same word on both sides of the coordination shows the user’s desire to refer to two different sets: one described by \( n \) and another formed by instances of a compound concept modified by \( n \).

For example the gloss (the industry that makes steel and steel products) of concept steel_industry:n#1 contains the coordination steel and steel products. **Heuristic 1** solves this coordination by bracketing it as \( (NP \ (NP \ (NN \ steel)) \ (CC \ and) \ (NP \ (NN \ steel) \ (NN \ products))) \).

**Heuristic 2**: if \( ((n_1 \ h \ IS \ in \ WordNet) \ AND \ (n_2 \ h \ IS \ in \ WordNet)) \) then 
\( bracket \ ((n_1 \ cc \ n_2) \ h)) \)
Rationale. If \( n_1 \) and \( n_2 \) are concepts in WordNet and some author wants to coordinate them he will most likely use a compact form for the coordination for language efficiency reasons (fewer words means less effort to express himself), especially when the head of the two compound nouns is the same.

For example, the coordination North and Central America from the gloss (tropical marine bivalve found chiefly off eastern Asia and Pacific coast of North and Central America) attached to pearl_oyster n\#1 has North America and Central America as concepts that are found in WordNet and thus the bracketing is (NP (NP (NN North) (CC and) (Central)) (NP (NN America))).

The next heuristic deals with the situation when \( n_1 \) and \( n_2 \) are siblings of common concept \( c \). It requires disambiguation information which is not an hardly accepted assumption, keeping in mind that the glosses will soon be disambiguated as part of the XWN project. For the test cases we manually tagged the sense of nouns in coordinations.

**Heuristic 3:** if ((\( n_1 \) IS sibling of \( c \)) AND (\( n_2 \) IS sibling of \( c \))) then bracket ((\( n_1 \) ce \( n_2 \) \( h \)))

Rationale. When two satellite nouns in a coordination are siblings of the same concept, the author indented to express two different subsets of the common concept, and in order to emphasize this relation places them in a coordination with a shared head.

An example solved by this heuristic is tomato and potato plants from the gloss of concept tomato_hornworm:n\#1: (large green white-striped hawkmoth larva that feeds on tomato and potato plants). The bracketing provided is (NP (NP (NN tomato) (CC and) (NN potato)) (NP (NN plants))).

When none of the previous heuristics trigger a specific coordination the default approach of splitting by the coordination is applied. All previous discussions were focused on coordinations that did not contain modifiers. Next we discuss about coordinations with modifiers.
5.2.3. Modifiers

There are many coordinations (29.94% of all coordinated compound nouns found on the noun hierarchy) that contain modifiers, and hence we face the issue of what concept they modify. For a frame such as $jj^* n_1 \ cc n_2 \ h$ the modifier $jj$ may be attached to $n_1$, to $(n_1 \ cc n_2)$ or $(n_1 \ cc n_2 \ h)$ (* means one or more occurrences of modifiers). An example is that of (an/DT annual/JJ school/NN or/CC university/NN reunion/NN) from the gloss (an annual school or university reunion for graduate) of concept homecoming:n#1, where annual should modify both school reunion, respectively university reunion.

Sometimes the presence of a modifier provides us with a hint for bracketing: large in gloss (extremely active cylindrical squid with short strong arms and large rhombic terminal fins) of concept ommastrephes:n#1, acting as a modifier after the coordination indicates that terminal fins is a noun phrase of its own and the bracketing should be: (NP (NP (JJ short) (JJ strong) (NNS arms)) (CC and) (NP (JJ large) (JJ rhombic) (NN terminal) (NN fins))). We designed several heuristics that attach the modifiers to their modifiees.

**Heuristic 4:** if (exists $jj_1$ AND $jj_2$) then bracket ($(jj_1 \ n_1) \ cc (jj_2 \ n_2 \ h)$) This heuristic says that when there is a modifier in front of the first noun $n_1$ and after the coordination $cc$, the coordination separates two different concepts which do not share the head. The rationale is that when the speaker adds a modifier to both $n_1,n_2$ he wants to emphasize a distinction between the two.

**Heuristic 5:** if ((exists $jj_1$) AND ($jj_2$ IS NOT) AND ($cc = \text{"or"}$)) then bracket ($(jj_1 \ n_1 \ cc n_2) \ mn$) This second heuristic simply says that when there is no modifier after the coordination and the coordination is or then $j_1$ is attached to the whole coordinated compound noun. The rationale is that when only one modifier is present it is most likely that the speaker wants to attach the modifier to both nouns. The presence of or comes from an empirical observation that this heuristic does not apply when an and is used. The example given in the first paragraph of this section (from gloss of homecoming:n#1) is covered by this heuristic.

118
HEURISTIC 6: \( \text{if}(\text{NOT exists } j_j1) \text{AND}(\text{exists } j_j2)) \text{ then bracket}(n_1 \text{ cc } (j_j2 \text{ n}_2 \text{ h})) \)

The rationale for this heuristic is simple: whenever the speaker places a modifier before two following nouns the modifier most likely attaches to the compound concept of the two nouns. An example is *mid-waters* and *deep slope waters* from gloss of dory:n#2. The right bracketing for this example is (NP (NP (NNS mid-waters)) (CC and) (NP (JJ deep) (NN slope) (NNS waters))).

After the bracketing is done a set of transformation rules are applied to trees in order to obtain logic forms (see [48]).

5.2.4. Experiments and Results

All our experiments were performed on a set of 522 coordinated compound nouns extracted from the WordNet noun hierarchy: 298 simple coordination (without modifiers) and 224 with modifiers. Experiments were done for each subset.

Table 5.2. Distribution of main source of errors at detection.

<table>
<thead>
<tr>
<th>Type</th>
<th>Percentage</th>
<th>Eliminated</th>
</tr>
</thead>
<tbody>
<tr>
<td>POS Tagging</td>
<td>9.45%</td>
<td>6.78%</td>
</tr>
<tr>
<td>Compound Concepts</td>
<td>8.78%</td>
<td>4.91%</td>
</tr>
<tr>
<td>Name Entity</td>
<td>25%</td>
<td>24.59%</td>
</tr>
<tr>
<td>Parse</td>
<td>1%</td>
<td>0%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>44.93%</td>
<td>36.28%</td>
</tr>
</tbody>
</table>

First we focus on the 298 simple coordinations. The definitions are extracted from glosses, tagged using Brill’s tagger, expanded to full sentences to minimize the parser’s errors and then parsed. Base NPs are detected that contain this sequence of tags: “NN[PS] CC NN[PS] NN[PS]”. The overall detection precision, defined as the number of correctly identified coordinated compound nouns over the total number, is \( P_D = 55.07\% \). This simple detection approach is poor in terms of precision.

Table 5.2 shows the distribution of the main source of errors at detection. Most errors are due to coordinated Named Entities, followed by POS tagging errors and
compound concepts identification. Parse errors are insignificant (1%). After applying the solutions proposed for each source of errors the overall precision jumps to $P_D = 94.46\%$. The taggers disagree on 152 cases out of which 78 or 51.31% are automatically assigned using a new set of rules, 38 are passed to WordNet agreement and 16 (out of 38) or 10.52% (42% of the 38 passed cases) are automatically corrected. The user intervenes in 47 cases. The NE module recognizes NE in 62 coordinations: 26 are simple, 31 are coordinated and 5 are all-way. The classification module misclassifies only one coordination: Central/NP and CC South/NP Africa/NP of gloss of Bantu:n.#1: (a member of any of a large number of linguistically related peoples of Central and South Africa). South Africa is a concept in WordNet while in this particular gloss it refers to the region of southern Africa. Common coordinated compound nouns are identified based on semantic information tagged manually.

Table 5.3. Performance of the proposed heuristics

<table>
<thead>
<tr>
<th>Name</th>
<th>Tempted</th>
<th>Solved</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heuristic 1</td>
<td>6</td>
<td>6</td>
<td>100%</td>
</tr>
<tr>
<td>Heuristic 2</td>
<td>112</td>
<td>103</td>
<td>91.96%</td>
</tr>
<tr>
<td>Heuristic 3</td>
<td>61</td>
<td>53</td>
<td>86.88%</td>
</tr>
<tr>
<td>Heuristic 4</td>
<td>120</td>
<td>114</td>
<td>95%</td>
</tr>
<tr>
<td>Heuristic 5</td>
<td>58</td>
<td>51</td>
<td>87.93%</td>
</tr>
<tr>
<td>Heuristic 6</td>
<td>46</td>
<td>46</td>
<td>100%</td>
</tr>
</tbody>
</table>

Detected coordinations are pipelined into the interpretation module which applies the three heuristics presented previously (the default approach of splitting by the coordination is applied in 44 cases). Using a straightforward approach such as providing a bracketing by splitting along the coordination: $(n_1) \cc (n_2) h$ leads to a bracketing precision of 38.48%.

Using the heuristics the bracketing precision jumps to 83.52%. The notable difference between the precision of the two approaches is primarily explained by the large number of coordinated proper compound nouns such as South and Central America.
For coordinations with modifiers we tested the performance of attaching the modifiers on 224 cases. The very first heuristic applied on 120 of them, the second on 58 and the third heuristic to 46. The performance of each heuristic is given in the fourth column of Table 5.3.

On the 525 cases the overall performance is 87.42% measured as number of correctly bracketed coordinations divided by all coordinations. The default approach of splitting by coordinations leads to 52.35% precision when applied to all 525 test cases.

5.3. Conclusions

We presented in this chapter a set of heuristics to deal with the problem of bracketing coordinated compound nouns in the larger context of deriving logic forms. We experimented with 525 candidate coordinations for which 87.42% bracketing precision was achieved.
Chapter 6
APPLICATION OF LOGIC FORMED GLOSSES IN QUESTION ANSWERING

This chapter addresses the problem of answer explanation/correctness in a Question-Answering system following a simple principle: the answer to a question is correct if all the words from the questions together with their syntactic relations are present in the answer. As this is seldom the case, we use a new correctness criteria: an answer to a question is considered to be correct if one can establish lexico-semantic paths from concepts in the question to concepts in the answer and logical proofs can be derived along those paths.

6.1. Question Answering System

In this chapter we use Falcon, a Question Answering system developed at SMU, in order to retrieve candidate answers for our work. The main modules of Falcon are shown in Figure 6.1.

The Question Processing processes a given question in order to detect the question type, the answer type and the relevant keywords that need to be submitted to the Paragraph Retrieval module.

The Paragraph Retrieval module uses an information retrieval system that provides a set of documents that contains a subset of the keywords in the question. Those documents are then scanned to detect paragraphs that contain the set of words. The output from the Paragraph Retrieval module is then fed into the Answer Extraction module.

The task of Answer Extraction is to detect in the candidate paragraphs the precise answer to the question. For example for the question *Who discovered the atom?* the answer should be in the form of a name. If several names are in a paragraph
the answer extraction module should guide itself to the correct one. Details on how to do that can be found in [22].

6.2. From Logic Forms to Axioms

The LFT of WordNet glosses are used in our Question Answering system [22] to extract some answers and provide answer explanations. To be useful, the glosses’ logic forms need to be transformed in axioms. One possibility is to generate several axioms for each gloss, from simple to more complex, as illustrated in Table 6.3.1.

A \{synset - definition\} pair can be viewed as an axiom of the underlying concept. For each such pair one or more axioms can be generated: (1) a fact (no left hand side) for synsets with empty definitions (2) a single axiom for the usual gloss (3) many axioms - for definitions that exhibit linguistic parallelism.

There are some specific aspects in the derivation of axioms for each part of speech. Usually the *nouns* definition consists of a genus and differentia. The template
for deriving noun axioms is: \( \text{concept}(x) \rightarrow \text{genus}(x) \) & \( \text{differentia}(x) \). Notice the propagation of arguments from the left hand side to the genus and differentia, without significant syntactic changes.

Table 6.1. Axioms extracted from the gloss of verb kill:v#1

| kill:v#1(e, x, y, z) ↔ cause(e1, x, y, z) & die(e2, y) |
| kill:v#1(e, x, y, z) ↔ put(e, x, y, w) & to(e, w) & death(w) |

The verbs also exhibit the same structural properties and the derivation is simple for the case of definitions containing only one verb. In the case of definitions consisting of a series of verbs, the derivation of axioms should take care of the syntactic function changes of the arguments on the right hand side from their counterparts on the left hand side.

Consider kill:v#1 \( \rightarrow \{ \text{“cause to die”} \} \). The axiom is kill:v#1(e1, x1, x2, x3) \( \rightarrow \) cause(e2, x1, e3, x3) & die(e3, x2). One notices the change of \( x2 \) from a direct object role for kill to a subject role for die. Also event \( e1 \) expands in two other events \( e1, e2 \).

In the case of adjectives which modify nouns, the axioms borrow a virtual head noun as shown here: American:a#1(x1) \( \rightarrow \) off(x1, x2) & United_States_of_America(x2). Similarly, since adverbs modify verbs - their arguments borrow the event of the verb. Adverb fast:x#1 has an axiom: fast:x#1(e) \( \rightarrow \) quickly(e).

6.3. Answer Extraction Procedure

The answer extraction procedure consists of four steps: transform questions and answers in logic forms, form lexical chains between pairs of concepts, apply unification on lexical chains and extract inferences to provide explanation. Each step requires special processing and resources. We detail each step of the answer extraction procedure in the next sections.
6.3.1. Transform Questions and Answers in Logic Forms

For each question, the information retrieval part of the QA system provides a set of candidate paragraphs that may contain the question answer. Thus, the input to the logic prover consists of the question and a candidate paragraph that needs to be evaluated.

The first step here is to transform the question and the candidate paragraph into logic forms called QLF and ALF (from answer logic form) (see also [50]).

Table 6.2. Axioms extracted from the gloss of adjective Colombian

<table>
<thead>
<tr>
<th>Axiom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colombian(x1) ↔ of(x1, x2) &amp; Columbia(x2)</td>
</tr>
<tr>
<td>Colombian(x1) ↔ relate(e1, x1, x2) &amp; Columbia(x2)</td>
</tr>
<tr>
<td>Colombian(x1) ↔ characteristic(x1) &amp; of(x1, x2) &amp; Columbia(x2)</td>
</tr>
<tr>
<td>Colombian(x1) ↔ of(x1, x2) &amp; people(x2) &amp; of(x2, x3) &amp; Columbia(x3)</td>
</tr>
<tr>
<td>Colombian(x1) ↔ relate(e1, x1, x2) &amp; people(x2) &amp; of(x2, x3) &amp; Columbia(x3)</td>
</tr>
<tr>
<td>Colombian(x1) ↔ characteristic(x1) &amp; of(x1, x2) &amp; people(x2) &amp; of(x2, x3) &amp; Columbia(x3)</td>
</tr>
</tbody>
</table>

If all the keywords from a question are found in the answer paragraph we only need to check if the syntactic relations are preserved. This is done via unification, as explained later. As the unification for one predicate in the question can be successful for several predicates in the answer, a depth first search algorithm is applied to find all possible unifications.

6.3.2. Form Lexical Chains between Pairs of Concepts

If some keywords are missing we attempt to establish possible lexical chains between pair of concepts, one in QLF and one in ALF, to check whether or not they are semantically linked.

A chain between a pair of concepts is a sequence of other concepts that are linked via hypernymy relations and/or axioms. A chain is established when two paths, each starting from different concepts intersect (they have a common WordNet concept).
Many chains may be found that link a pair of concepts. To evaluate all of them would be too costly and unnecessary. Thus a filtering mechanism is required. We do this using two heuristics: (1) keep only the shortest lexical chains (2) if several short chains exist, pick those that contain more hypernymy relations.

procedure chainsFind(qConcepts, aConcepts)
{
    input:
    qConcepts - set of question concepts
    aConcepts - set of answer concepts
    qPaths = 0;
    for(i=0;i<=card(qConcepts);i++)
        qPaths = qPaths ∪ pathsFrom(qConcepts[i]);
    aPaths = 0;
    for(i=0;i<=card(aConcepts);i++)
        aPaths = aPaths ∪ pathsFrom(aConcepts[i]);
    chains = 0;
    for(i=0;i<=card(qConcepts);i++)
        for(j=0;j<=card(qConcepts);j++)
            chains = chains ∪ intersect(qConcepts[i], aConcepts[j]);
    return (chains);}

The rationale behind these is that the shorter the chain, the stronger the semantic relation between the pair of concepts, and hypernymy relations are preferred over axioms. A formal description of the chains finding algorithm is given below.

6.3.3. Apply Unification on Lexical Chains

Once chains are established between a pair of concepts from the QLF and ALF, the logic form representation provides us with a mechanism for performing unification. This checks the syntactic constraints. Two concepts along the chain are unified if their predicates and arguments match.
In a successful unification the question predicate arguments will be bound to the answer predicate arguments and the QLF and ALF will be updated to reflect the new status of the arguments. Step 0 in Table 6.4.1 shows the QLF and ALF, respectively. In step 1, the matching is performed between some predicates, e.g. Lucelly_Garcia from QLF and ALF, and x1 is bound to x1' which is reflected in the new QLF shown in step 1.

6.3.4. Extract Inferences to Provide Explanation

The concepts along those lexical chains that survive the unification test lead to inferences that explain the answer. It is only necessary to retrieve the concepts along the chain and the hypernymy and axioms explain the relation between them.

6.3.4.1. Hyper-inference

If a hyper link is encountered, the predicate is replaced with its hypernym with the same arguments. For example from poison:v\#2(e, x1, x2, x3) following a hyper link we infer kill:v\#1(e, x1, x2, x3).

This type of inference is sound as a more specific concept always implies a more general one and the hypernymy relation is theoretically equivalent to ISA type of relation.

More elaborate discussion of reasoning mechanisms in the context of WordNet relations and combinations of relations is presented in [23].

6.3.4.2. Axiom-inference

Inference with axioms is trickier, as a concept may have several alternative axioms available. An axiom selection operation needs to be performed for such cases.

Table 6.3.1 illustrates six alternative axioms for colombian:a\#1. When the inference engine needs to infer using one of the axioms of colombian:a\#1 we use contextual information from the answer and question to select one. There are several features that the selected axiom should have:
it should contain the next concept in the chain, so that the chain can be followed. There are some alternative definitions for a concept that contain different concepts. We prefer the alternative that contains the next concept in the chain. For example, the gloss of kill:v#1 contains two alternatives as given in Table 6.1. If the next concept in the chain were put:v#2 the second axiom would be picked. If the next concept in the chain were die:v#1 the first alternative would be chosen.

• if all alternative axioms contain that concept, as is the case of colombian:a#1, we choose the one that preserves more relations from the question. For example, in question Q045 Colombian:n#1 is the object of a of relation, thus the first axiom of Colombia, out of six, is preferred.

6.3.5. Integration of Axioms into the Input Context

Once the derivation paths are established and the axiom to be used is selected, there is one more important step to be done. The integration of axioms into the logic form of the input question/answer. As the logic form contains syntactic relations, the derivation procedure should propagate the syntactic relations the underlying concept participates in to the concepts in the gloss. We will illustrate this for the integration of an axiom belonging to each part of speech.

Whenever a noun is expanded with its gloss, the newly derived context should include the genus of the gloss within each relation the original noun was involved. For example, consider the reduced context former:a#3(x1) & ambassador:n#1(x1) & of(x1, x2) & Colombia:n#1(x2) and suppose we want to expand with ambassador(x1) ↔ diplomat:n#1(x1) & of(x1, x2) & highest:a#1(x2) & rank:n#2(x2). After integration, the newly context would be former:a#3(x1) & ambassador:n#1(x1) & of(x1, x2) & Colombian:n#1(x2) & diplomat:n#1(x1) & of(x1, x3) & highest:a#1(x3) & rank:n#2(x3).

The integration of a verb gloss is slightly more complicated. Let us consider the axiom star:v#1(e, x1, x2, x3) ↔ feature:v#1(e, x1, x2, x4) & as(e, x3) & star:n#4(x4)
that needs to be integrated into the context show:n#1(x1) & star:v#1(e, x1, x2) & person:n#1(x2), where show:n#1 could be “Shopping” and person:n#1 could be “Jude Law” of question Q719. The newly formed context is show:n#1(x1) & star:v#1(e, x1, x2) & person:n#1(x2) & feature:v#1(e, x1, x2, x3) & as(e, x3) & star:n#4(x3). If in the given context there is no indirect object for star:v#1, feature:v#1 will keep the one from gloss, namely star:n#4. Otherwise, the relation embedded in the preposition would have to be duplicated to link feature:v#1 to both the indirect object from the context and the one from WordNet gloss.

Adjectives and adverbs are easier to integrate. They usually stand as a modifier to the head in the given context. Thus, if we consider colombian:n#1(x) ↔ of(x, y) & Colombian:a#1 & ambassador:n#1(x1), the newly derived context is ambassador:n#1(x1) & of(x1, x2) & Colombian:n#1(x2). The main advantage of this kind of inference is that the relation between ambassador:n#1 and Colombian:n#1 is explicitly stated and the equivalence of question context ambassador:n#1(x1) & of(x1, x2) & Colombian:n#1(x2) and newly derived context ambassador:n#1(x1) & of(x1, x2) & Colombia:n#1(x2) can be simply determined by string matching at predicate and argument level.

6.4. Examples

For each type of questions that we treat we provide here an illustrative example and consider step by step the usage of axioms.

6.4.1. Example 1

Consider the TREC question:

Q045: When did Lucelly Garcia, former ambassador of Colombia to Honduras, die? The answer is found in “Several gunmen on a highway leading to the Colombian city of Ibagué murdered Colombian Ambassador to Honduras Lucelly Garcia today”. As illustrated in Table 6.4.1 at Step 0 we are able to match a few predicates: Lucelly_Garcia, ambassador, TIME-STAMP.
With the help of the axioms, chains are found: from *Colombian* in the answer to *Colombia* in the question, respectively from *murder* to *die* (see Figure 1). For *former* there is no link to a concept in the answer and we just ignore it (as being a modifier of an already matched predicate *ambassador*).

| Step 0 | QLF: Lucelly\_Garcia(x1) & former(x1) & ambassador(x1) & off(x1, x2) & Colombia(x2) & to(x1, x3) & Honduras(x3) & die(e1, x1) & TIME-Stamp(e1)  
|        | ALF: gunman(x2') & murder(e1', x2', x1') & Colombian(x1') & ambassador(x1') & to(x1', x3') & Honduras(x3') & Lucelly\_Garcia(x1') & TIME-Stamp(e1') |
| Step 1 | QLF: off(x1', x2) & Colombia(x2) & die(e1, x1')  
|        | ALF: gunman(x2') & murder(e1', x2', x1') & off(x1', x7') & Colombia(x7') & ambassador(x1') & to(x1', x3') & Honduras(x3') & Lucelly\_Garcia(x1') & TIME-Stamp(e1') |
| Step 2 | QLF: die(e1, x1')  
|        | ALF: gunman(x2') & kill(e1', x2', x1') & intentionally(e1') & with(e1', x8') & premeditation(x8') & off(x1', x7') & Colombia(x7') & ambassador(x1') & to(x2', x3') & Honduras(x3') & Lucelly\_Garcia(x1') & TIME-Stamp(e1') |
| Step 3 | QLF: die(e1, x1')  
|        | ALF: gunman(x2') & cause(e2', x2', x3') & die(e3', x1') & intentionally(e1') & with(e1', x') & premeditation(x8') & off(x1', x7') & Colombia(x7') & ambassador(x1') & to(x1', x3') & Honduras(x3') & Lucelly\_Garcia(x1') & TIME-Stamp(e2') & TIME-Stamp(e3') |

The ALF in Step 1 shows *Colombian* expanded with axioms from WordNet (see Table 6.3.1). The new QLF to be proven contains only the predicate *die*. Step 2 in Table 6.4.1 shows the ALF after the expansion of *murder* with its corresponding axiom: murder(e1, x1, x2) ↔ kill(e1, x1, x2) & intentionally(e1) & with(e1, x3) & premeditation(x3). Then Step 3 is derived using: kill(e, x1, x2) ↔ cause(e1, x1, e2) & die(e2, x2). As explained earlier, the subject of *kill* is propagated as subject of *cause* and the object of *kill*, which is *Lucelly\_Garcia*, as the subject to *die*.

Also, we replicate the TIME-STAMP predicate to modify both e2 and e3. The QLF is successfully proven as it becomes empty.
6.4.2. Example 2

Consider the TREC-9’s question:

**Q481:** Who shot Billy the Kid?

The Q/A system identifies a few paragraphs that contain all the keywords from the question and the answer type. Two such paragraphs are:

- **P1:** The scene called for Phillips’ character to be saved from a lynching when Billy the Kid (Emilio Estevez) shot the rope in half just as he was about to be hanged.

- **P2:** In 1881, outlaw William H. Bonney Jr., alias Billy the Kid, was shot and killed by Sheriff Pat Garrett in Fort Sumner, N.M.

The answer is provided by paragraph P2 and is depicted by the system as follows. Using LFT, the question has a representation of the form: \( Q: \text{PERSON}(x_1) \land \text{shoot}(e_1, x_1, x_2) \land \text{Billy the Kid}(x_2) \) where \( x_1 \) is a variable that is to be unified with an entity of type PERSON from paragraphs. The paragraphs have the following LFT:

- **P1:** \( \text{Billy the Kid}(x_1') \land \text{shoot}(e_1', x_1', x_2') \land \text{rope}(x_2') \)
- **P2:** \( \text{Billy the Kid}(x_1') \land \text{shoot}(e_1', x_2', x_1') \land \text{Sheriff Pat Garrett}(x_2') \)
The logic prover attempts to prove the question starting from the paragraph. The logic proof for P1 fails because Billy_the_Kid is the agent of shooting, not the object as requested by the question. The logic proof for P2 succeeds and the agent of shooting Sheriff_Pal_Garrett unifies with PERSON from the question. The prover yields $e_1 = e_1'$, $x_1 = x_2'$ and $x_2 = x_1'$. Note that our logic form representation based on slot-allocation played a crucial role in this proof.

6.5. Q/A Results

Table 6.5 gives a set of Question Answer pairs that benefit from the slot allocation feature of logic form and from WordNet glosses. The table also shows the rank of the correct answer given by Falcon and us, respectively. Whether or not WordNet chains were used is indicated in the last column. We illustrate in Table 6.6 and Table 6.7 the bindings of question arguments to answer arguments.

In Table 6.8 we provide statistics on questions whose correct answers can be explained following lexico-semantic paths in WordNet. To better evaluate the impact of the logic prover the word sense disambiguation task has been performed manually for the questions, answers and for a set of targeted glosses. The Pairs of Concepts column illustrates the number of pairs of concepts from the question, respectively answer paragraph (ground concepts). The paths used for each Question-Answer pair are given in the last column. From the table we can see that the gloss relation is extensively used in chains and this emphasizes once again the important role of glosses as a rich source of knowledge. Some questions require more than one chain in order to be explained, e.g. Q045, and others require only usage of synonymy relation among words from same synset, e.g. question Q336 where the chain found - establish is inside same synset {establish, set up, found, launch}.

The paths column shows the number of paths retrieved: these are paths that originate in the ground concepts, some of which intersect and form lexical chains. The chains column shows the number of lexical chains established. The next column shows the number of concepts encountered along those paths. The Chains used column
Table 6.4. Experiments with TREC questions and comparison with Falcon

<table>
<thead>
<tr>
<th>Question</th>
<th>Question and Answer</th>
<th>Falcon rank</th>
<th>Our rank</th>
<th>WN chains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q006</td>
<td>Q: Why did David Koresh ask the FBI for a word processor? A: Mr Koresh sent a request for a word processor to enable him to record his revelations.</td>
<td>-</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>Q045</td>
<td>Q: When did Lucelly Garcia, a former ambassador of Columbia to Honduras, die? A: Several gunmen on a highway leading to the Colombian city of Ibagué murdered Colombian Ambassador to Honduras Lucelly Garcia (today) (February 14 1994).</td>
<td>-</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>Q074</td>
<td>Q: Who leads the star ship Enterprise in Star Trek? A: The star ship Enterprise in Star Trek is led by Captain Kirk.</td>
<td>2</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>Q198</td>
<td>Q: How did Socrates die? A: Socrates chose to drink poison.</td>
<td>-</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>Q280</td>
<td>Q: What’s the tallest building in New York City? A: The World Trade Center, at 1,368 feet, is New York City’s tallest building</td>
<td>2</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>Q331</td>
<td>Q: How hot is the core of the earth? A: The temperature of Earth’s inner core may be as high as 9,000 degrees Fahrenheit.</td>
<td>2</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>Q336</td>
<td>Q: When was Microsoft established? A: Microsoft Corp was founded in the US in 1975</td>
<td>-</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>Q362</td>
<td>Q: What is the capital of Burkina Faso? A: A centerpiece of the pope’s trip will be a ceremony Monday in Ouagadougou, capital of Burkina Faso</td>
<td>3</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>Q381</td>
<td>Q: Who assassinated President McKinley? A: Leon Czolgosz assassinated President McKinley</td>
<td>5</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>Q424</td>
<td>Q: What do you call a group of geese? A: I imagine a flock of geese</td>
<td>-</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>Q471</td>
<td>Q: What year did Hitler die? A: Adolf Hitler committed suicide in May 1945</td>
<td>-</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>Q481</td>
<td>Q: Who shot Billy the Kid? A: Billy the Kid, was shot and killed by Sheriff Pat Garrett in Fort Sumner, N.M.</td>
<td>3</td>
<td>1</td>
<td>No</td>
</tr>
</tbody>
</table>
shows the number of chains selected to perform unification. In each case, one chain led to the correct answer.

6.6. Conclusions

This chapter presented the transformation of WordNet glosses from logic forms to axioms and how these axioms can be helpful in practical applications such as Question Answering. Results with a reduced set of questions from the TREC Q/A test set were presented. WordNet glosses in LF can provide logical explanations of answers in Question Answering systems. The use of first order logic to question answering was employed in [22] for TREC question. The main drawback of the unlabeled logic representation of [22] is that it disconsiderers syntactic roles that play a crucial role in more complex questions such as those treated in this chapter. We provide better answers for questions with all keywords in the first ranked answers for which we take advantage of syntactic roles. Better answers were also provided for questions that have answers with only a subset of their keywords present and for which we employ world knowledge available in WordNet glosses. When using WordNet the idea is to transform glosses into axioms (one or more for each synset) that are automatically

<table>
<thead>
<tr>
<th>Question</th>
<th>Question and Answer</th>
<th>Falcon rank</th>
<th>Our rank</th>
<th>WN chains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q486</td>
<td>Q: How many states have a &quot;lemon law&quot; for new automobiles? A: Forty-four states and</td>
<td>3</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>the District of Columbia have adopted so-called lemon laws</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q498</td>
<td>Q: Who portrayed Fatman in the television show, &quot;Jake and the Fatman&quot;? A: In &quot;Jake</td>
<td>3</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>and the Fatman&quot; William Conrad stars as J.L. “Fatman” McCabe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q640</td>
<td>Q: Who is the voice of Miss Piggy? A: He co-directed the former with Frank Oz, the</td>
<td>2</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>voice of Miss Piggy.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Question</td>
<td>Question and Answer</td>
<td>Arg binding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Q006     | Q: David_Koresh(x1) & ask(e1, x1, x2, x3) & FBI(x2) & for(e1, x3) & word_processor(x3)  
A: Koresh(x1') & send(e1', x1', x2') & request(x2') & for(x2', x3') & word_processor(x3') & to(e1', e2') & enable(e2', x1') & to(e2') & record(e3', x1', x4') & off(x4', x1') & revelations(x4')   | x1=x1'  
x2=x3'  
x3=x3'  
e1=e3'                                                   |
| Q045     | Q: TIME(x1) & Lucelly_Garcia(x2) & former(x2) & ambassador(x2) & off(x2, x3) Columbia(x3) & to(x2, x4) & Honduras(x4) & die(e1, x2) & on(e1, x1)  
A: Several(x1') & gunman(x1') & on(x1', x2') & highway(x2') & lead(e1, x2', x4', x3') & to(e1', x3') & Colombian(x3') & city(x3') & off(x3', x5') & Ibague(x5') & murder(e2', x1', x6')  
Colombian(x6') & Ambassador(x6') & to(x6', x7') & Honduras(x7') & Lucelly_Garcia(x6') & on(e2', x8') & February_14_1994(x8').   | x1=x5'  
x2=x6'  
x3=x3'  
x4=x7'  
e1=e2'                                                   |
| Q074     | Q: PERSON(x1) & lead(e1, x1, x4) & on(x4, x2, x3) & star_ship(x2) & Enterprise(x3) & in(x4, x5) & Star_Trek(x5)  
A: mn(x1', x2', x3') & star_ship(x2') & Enterprise(x3') & in(x1', x4') & Star_Trek(x4') & lead(e1', x5', x1') & by(e1', x5') & Captain_Kirk(x5')   | x1=x5'  
x2=x2'  
x3=x3'  
x4=x1'  
x5=x4'  
e1=e1'                                                   |
| Q198     | Q: Socrates(x1) & die(e1, x1)  
A: Socrates(x1') & chose(e1', x1', x2') & to(e1', e2') & drink(e2', x1', x2') & poison(x2')   | x1=x1'  
e1=e3'                                                   |
| Q280     | Q: tallest(x1) & building(x1) & in(x1, x2) & New_York_City(x2)  
A: World_Trade_Center(x1') & New_York_City(x2') & of(x1', x2')  
tallest(x1') & building(x1')   | x1=x1'  
x2=x2'                                                   |
| Q331     | Q: QUANTITY (x1) & of(x1, x2) & core(x2) & of(x2, x3) & earth(x3)  
A: temperature(x1') & of(x1', x3') & Earth(x2') & of(x3', x2') & inner(x3') & core(x3') & 9000(x4') & degrees(x5') & Fahrenheit(x6') & mn(x7', x4', x5', x6')   | x1=x4'  
x2=x3'  
x3=x2'                                                   |
| Q336     | Q: TIME(x1) & in(e1, x1) & Microsoft(x2) & establish(e1, x2)  
A: Microsoft_Corp(x1') & found(e1', x2', x1') & in(e1', x3') & US(x3') & in(e1', x4') & 1975(x4')   | x1=x4'  
x2=x1'  
e1=e1'                                                   |
| Q362     | Q: CITY(x1) & capital(x1) & of(x1, x2) & Burkina_Faso(x2)  
A: Ouagadougou(x1') & capital(x1') & of(x1', x2') & Burkina_Faso(x2')   | x1=x1'  
x2=x2'                                                   |
Table 6.7. Argument binding for some Question-Answer pairs (cont’d)

<table>
<thead>
<tr>
<th>Question</th>
<th>Question and Answer</th>
<th>Arg binding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q381</td>
<td>Q: PERSON(x1) &amp; assassinate(e1, x1, x2) &amp; mn(x2, x3, x4) &amp; President(x3) &amp; Mc Kinley(x4) A: Leon_Czolgosz(x1') &amp; assassinate(e1, x1', x2') &amp; mn(x2', x3', x4') &amp; President(x3') &amp; Mc Kinley(x4')</td>
<td>x1=x1' x2=x2' x3=x3' x4=x4' e1=e1'</td>
</tr>
<tr>
<td>Q424</td>
<td>Q: group(x1) &amp; of(x1, x2) &amp; geese(x2) A: flock(x1', x2') &amp; of(x1', x2') &amp; geese(x2')</td>
<td>x1=x1' x2=x2'</td>
</tr>
<tr>
<td>Q471</td>
<td>Q: TIME(x1) &amp; Hitler(x2) &amp; die(e1, x2) &amp; in(e1, x1) A: Adolf_Hitler(x1') &amp; committ(e1', x1', x2') &amp; suicide(x2') &amp; in(e1', x3') &amp; May 1945(x3')</td>
<td>x1=x2' x2=x1' e1=e2'</td>
</tr>
<tr>
<td>Q481</td>
<td>Q: PERSON(x1) &amp; shoot(e1, x1, x2) &amp; Billy_The_Kid(x2) A: Billy_The_Kid(x1') &amp; shoot(e1', x2', x1') &amp; kill(e2', x2', x1') &amp; by(x1', x2') &amp; by(x2', x2') &amp; Sheriff_Pat_Garrett(x2') &amp; in(x1', x3') &amp; in(x2', x3') &amp; Fort_Summer(x3') &amp; N.M. (x3')</td>
<td>x1=x2' x2=x1' e1=e1'</td>
</tr>
<tr>
<td>Q486</td>
<td>Q: QUANTITY(x1) &amp; state(x1) &amp; have(e1, x1, x2) &amp; lemon_law(x2) &amp; for(x2, x3) &amp; new(x3) &amp; automobile(x3) A: Forty-four(x1') &amp; state(x1') &amp; and(x2', x1', x3') &amp; District_of_Columbia(x3') &amp; adopt(e1', x2', x4') &amp; lemon_law(x4')</td>
<td>x1=x2' x2=x4' e1=e1'</td>
</tr>
<tr>
<td>Q498</td>
<td>Q: PERSON(x1) &amp; portray(e1, x1, x2) &amp; Fatman(x2) &amp; in(e1, x3) &amp; mn(x3, x4, x5) &amp; television_show(x4) &amp; Jake_and_the_Fatman(x5) A: in(e1', x1') &amp; Jake_and_the_Fatman(x1') &amp; William_Conrad(x2') &amp; star(e1', x2', x3', x4') &amp; as(e1', x4') &amp; Fatman(x4')</td>
<td>x1=x2' x2=x4' x3=x1' x4=x1' x5=x1' e1=e1'</td>
</tr>
<tr>
<td>Q640</td>
<td>Q: PERSON(x1) &amp; voice(x1) &amp; of(x1, x2) &amp; Miss_Piggy(x2) A: Frank_Oz(x1') &amp; voice(x1') &amp; of(x1', x2') &amp; Miss_Piggy(x2')</td>
<td>x1=x1' x2=x2'</td>
</tr>
</tbody>
</table>
Table 6.8. Statistics for questions that benefit from WordNet

<table>
<thead>
<tr>
<th>Q</th>
<th>Pairs</th>
<th>Paths</th>
<th>Chains</th>
<th>Concepts</th>
<th>Usable chains</th>
<th>Chains used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q006</td>
<td>28</td>
<td>391</td>
<td>161</td>
<td>729</td>
<td>2</td>
<td>request → hypernymy → ask</td>
</tr>
<tr>
<td>Q045</td>
<td>84</td>
<td>685</td>
<td>223</td>
<td>1025</td>
<td>3</td>
<td>murder → gloss → kill → gloss → die (Colombian) → gloss → Colombia</td>
</tr>
<tr>
<td>Q198</td>
<td>24</td>
<td>548</td>
<td>182</td>
<td>858</td>
<td>1</td>
<td>poison → gloss → kill → gloss → die</td>
</tr>
<tr>
<td>Q336</td>
<td>12</td>
<td>338</td>
<td>116</td>
<td>738</td>
<td>1</td>
<td>found → synonymy → establish</td>
</tr>
<tr>
<td>Q424</td>
<td>27</td>
<td>472</td>
<td>180</td>
<td>840</td>
<td>2</td>
<td>flock → gloss → group</td>
</tr>
<tr>
<td>Q471</td>
<td>18</td>
<td>447</td>
<td>90</td>
<td>421</td>
<td>1</td>
<td>suicide → gloss → kill → gloss → die</td>
</tr>
<tr>
<td>Q486</td>
<td>84</td>
<td>766</td>
<td>186</td>
<td>1103</td>
<td>2</td>
<td>adopt → hypernymy → have</td>
</tr>
<tr>
<td>Q498</td>
<td>21</td>
<td>157</td>
<td>12</td>
<td>52</td>
<td>2</td>
<td>portray → gloss → act ← gloss ← starvation</td>
</tr>
</tbody>
</table>

...triggered when needed. This chapter shows that WordNet glosses are a powerful source of world knowledge that can be helpful in real applications such as Question Answering.
Chapter 7

CONCLUSIONS

We presented in this dissertation a methodology to derive logic forms for WordNet glosses. WordNet glosses are textual definitions of concepts that form a rich source of world knowledge that might be useful in several applications, ranging from Information Retrieval to Formal Proofs.

In order to leverage this information to automatic systems the textual glosses need to be transformed into a computational representation. We chose as the knowledge representation formalism the logic forms which combine the power of semantic nets and natural language based representations.

7.1. Summary

We proposed few extensions to the original logic form representation. We proposed solutions for comparatives which were originally ignored and provided solutions for a few special cases that were not specified: postmodifiers, relative adverbs and possessive pronouns.

Then, we focused on the problem of deriving logic forms from textual English definitions. The approach we considered was to take advantage of the structural information found in a parse tree. To obtain the parse tree of a gloss, we needed to part of speech tag and parse the gloss. As the freely available tools for performing these tasks are not perfect, we proposed a solution to improve on the reported performance. We managed to get an almost 3% increase in the accuracy measure for part of speech tagging and obtained better results than state of the art by using an assembly of parsers and applying combination techniques, with simple majority voting providing the best results.
The next step was to map each parse tree into a set of transformation rules that produce the logic forms using a large bottom up traversal of the parse tree. One challenging issue was represented by the number of transformation rules needed to accurately derive logic forms for all trees. Because the number of rules approaches nearly 10,000 we proposed to statistically select rules that yield high precision logic forms over a large number of glosses (this number was empirically established and details are provided in Chapter 4). A set of 70 such rules were selected. To cover all glosses, a set of heuristics were proposed that produce less accurate logic forms for all uncovered glosses.

The special case of coordinated compound nouns is treated separately due to its bracketing characteristics that are not captured by the statistical parser model that we use.

Finally, we presented an application of glosses in logic forms to the problem of Question Answering where we successfully retrieved better answers for hard-to-answer questions that need syntactic information in order to distinguish the correct answers from the wrong ones. We form lexical chains in WordNet glosses and provided logical proofs along plausible chains. The chains that lead to correct proofs are selected for justifying the chosen answer.

7.2. Goals Revisited

In this section we revisit the set goals in the introduction, discuss the achievement of each and look into future work related to them.

7.2.1. Concept Predicate Representation

Chapter 4 showed two major trends in knowledge representation: semantic nets and natural language based representations. The tradeoff among the two is expressiveness and reasoning methods versus easy-to-derive and user friendly features. The concept based representation that we use and extend combines features from both semantic nets and natural language based representations. We showed that the main
advantage of concept-based representation versus morpheme based representation of TACITUS consists of eliminating the predicate ambiguity resulting in reduced search spaces when it comes to proving and reasoning. For the case of WordNet glosses we proposed solutions for postmodifiers, comparatives and relative adverbs. The solutions adopted treat those cases similar to prepositional predicates with the semantic of introducing relations among two arguments (the compared entities for the case of comparatives, for example). As an extension of Chapter 4, Chapter 5 presented an original solution to the problem of bracketing coordinated compound nouns. A set of heuristics are provided that use part of speech, lexical and semantic features to provide a correct bracketing for coordinated base noun phrases. A significant improvement of 35.07% over a baseline solution is achieved.

7.2.2. High Performance Solution for Logic Form Derivation

After the theoretical aspects of logic forms for WordNet glosses have been depicted in Chapter 2 the technical issues on how to derive such kind of representation were handled. The approach used is to follow the output of a parser and detect a set of high performance rules and a set of high recall heuristics that led to a performance of 89% on deriving logic forms for WordNet glosses. The selection of rules was based on probabilistic analysis of the WordNet glosses in parsing trees representation, where for each pair (node-tag, children tag) the number of occurrences was computed and then the most frequent rules were selected. Given the parse tree of a given gloss the selected rules are triggered by using a bottom up strategy of traversing the tree. When the top most node of the tree is reached the logic form of the gloss is obtained. A partial solution may be obtained when arguments of predicates are missing, in which case a set of heuristics are triggered for each type of missing argument.

The main disadvantage of using heuristics is the fact that they are not completely reliable, which may result in partially correct logic forms. The user may choose to stop the derivation process before the heuristics are triggered and, thus, detect the unfilled arguments and assign them using other methods.
7.2.3. Boosting Performance of Tagging and Parsing

Before the derivation process is started, glosses need to be tokenized, part of speech tagged and parsed. We presented in Chapter 3 solutions for each of these tasks. For the tokenization process, we extended the Penn Treebank tokenization guidelines with a few specifics of WordNet glosses. To boost up the performance of part of speech tagging we used a two-level voting scheme that combines the output of MXPOST and Brill’s tagger at the first level, and the output of Brill’s tagger and WordNet coarse syntactic categories at the second level. A boost in performance of almost 3% was obtained over the best state of the art tagger. The increase in performance is obtained at the expense of human intervention on 7.2% of words. For parsing we used two combination techniques such as constituent voting and parser switching to get better results. For constituent voting, the best results are obtained for the case of simple majority voting when a constituent is picked as part of the output bag of constituents only if it appears in at least half of the voters plus one. The performance measures for an assembly of five voters obtained by training Collins’ model on five partitions of WSJ Penn treebank were: P=94.39%, R=93.99%, F=94.18%. For the case of parser switching, the best results were obtained for nearest neighbor techniques which select the tree that has most common constituents with all other candidate trees. For the case of five candidates, the performance measures were: P=93.46%, R=93.86%, F=93.65%.

7.2.4. Impact of WordNet Glosses on Question Answering

Chapter 6 shows how WordNet glosses in logic forms can be helpful in real world applications such as Question Answering. There are two types of questions for which we managed to provide better ranking answers than state of the art systems: (1) all the keywords from question can be found in the answer, and the syntactic relations in the question are preserved in the answer (2) some keyword concepts are missing, but lexico-semantic chains can be established between them and concepts in the answer.
We presented a set questions for which all the keywords from the question are found in the answer. For these questions, the system found the correct answer in the top 5 ranked answers (initial rank) just by matching keywords. The logic prover boosts the performance by eliminating the wrong answers and bringing the correct answer to the first place.

On a set of questions from the the second category, we successfully retrieved chains between unmatched concepts in the question. To better evaluate the impact of the logic prover the word sense disambiguation task has been done manually for the questions, answers and for a set of targeted glosses. The Pairs of Concepts column in Table 6.8 illustrates the number of pairs of concepts from the question and answer paragraph (ground concepts). The paths column shows the number of paths retrieved: these are paths that originate in the ground concepts, some of which intersect and form lexical chains. The chains column shows the number of lexical chains established. Next column shows the number of concepts encountered along those paths. The Chains used column shows the number of chains selected to perform unification. In each case, one chain led to the correct answer.

7.3. Future Work

The worked presented in this dissertation may be continued following few tracks that are indicated nextly.

7.3.1. Extend Analysis To Other Parts of Speech

We do expect changes in rule distribution when one needs to generate LFs for glosses attached to concepts belonging to other parts of speech. We tested the implemented LFTs on a set of 400 verb glosses and we noticed a degradation on the accuracy of about 10-15%. This is especially due to a greater level of linguistic parallelism in the verb glosses that was not caught in the implemented rules for noun glosses.
Our future plan is to extend the LFT to the other three parts of speech: verbs, adjectives and adverbs. A separate analysis similar to the one developed for nouns is to be developed.

7.3.2. Issues Related to Automatically Transform WordNet into a Knowledge Base

As we showed in the application of glosses in logic forms to Question Answering a pair (synset, definition in logic form) can be viewed as an axiom. One definition can be mapped into more than one axiom since the presence of parallelism in the definition imposes this. How to handle the issues of transforming WordNet into a first order knowledge base constitutes a main trend of future work as a natural extension to the ideas presented here.

7.3.3. Lexical Chains with More Types of Relations

Another future trend of our work would be to detect lexical chains using more WordNet relations than only hypernymy and axioms. The pertainym relation (links adjectives to nouns and adverbs to adjectives), entailment and causation are other relations that are best candidates for near future studies. Mainly, we are interested in relations that provide us with sound inference procedures so that the explanations offered based on them are plausible. Increasing the number of relations leads to an increased number of candidate paths and consequently the selection criterias need to be refined.

A further extension would be to integrate reasoning methods that combine two or more WordNet relations as presented in [23]. Those inferences are independent of the concept that they apply which assures the generality of their domain of application. An example of inference rules that combine two relations is (hypernym/IS-A, hypernym/IS-A) and (hypernym, entail). There are sixteen combinations of inference rules that use the two WordNet relations hypernymy and entailing. The main issue is that some inferences are not always plausible and, thus, a filtering process needs to be done before integrating them into a generally applicable prover.
7.3.4. Implement Prover

The experiments in chapter 7 were done with a partial implementation of a prover. A full implementation is to be developed and an analysis of the performance of such prover to be reported. The proof is implemented as a search mechanism guided by two principles: (1) shortest paths are preferred over longer ones (2) paths with cheaper relations are preferred over more expensive ones, e.g. paths with hypernymy relations are preferred over paths with axioms. Even for simple questions where the predicates in the question can be retrieved from the answer, the proving process is similar to a search. For question 481 where the logic form of the question is person:n#1(x1') & shoot:v#1(e1', x1', x2') & Billy_the_Kid:n#1(x2') and the answer logic form is person:n#1(x1) & Billy_the_Kid:n#1(x1) & shoot:v#1(e1, x2, x1) & Pat_Garrett(x2) & person:n#1(x2), when one tries to match the predicate person:n#1 from the question with one from the answer there are two possibilities: x1' unifies with x1 or x1' unifies with x2. Only the second unification satisfies the extra constraint of being an agent of shoot, thus when the first unification fails we backtrack and try the other one.

7.3.5. Impact of Prover on Question Answering Performance

We plan to study the impact of the prover in the performance of Question Answering systems. We will run the prover on the output of a state-of-the-art Question Answering system and compare the results with the original ones. We expect improvements on answer precision based on the partial results presented in the previous chapter. Inferences along lexico-semantic chains may provide explanations for why the answer to a question was chosen as the right one.

7.4. Closing Words

The concept-based representation is a very powerful knowledge representation mechanism. We focus in this thesis on how to derive logic forms for WordNet glosses which is a first step towards automatically building knowledge bases. We addressed
the technical issues of POS tagging and parsing glosses, the issues involved in logic form derivation from parse trees and we applied the glosses in logic forms to Question Answering.

The power of the logic form representation consists on combining first order logic (the most understood logic formalism) with generally applicable ontologies (WordNet). We envisage that the logic form will open new horizons on the understanding of reasoning processes and will bring further AI applications that are based on knowledge processing.
Annex A - Most frequent rules for each important phrase

S → NP VP
S → VP
S → S VP
S → NP VP NP
S → ADVP VP
S → S CC S
S → NP VP SBAR

SBAR → WHNP S
SBAR → IN S
SBAR → RB S

WHNP → NP
WHNP → IN NP

NP → NN
NP → JJ NN
NP → NN NN
NP → NN CC NN
NP → JJ NN JJ NN
NP → NN NN CC NN NN
NP → JJ NN NN CC JJ NN NN
NP → RB JJ NN
NP → VBG NN
NP → VBN NN
NP → JJ VBN NN
NP → VBN NN CC NN
NP → PRP$ NN

NP → NP PP
NP → NP CC NP
NP → NP SBAR
NP → NP VP
NP → NP PP SBAR
NP → NP PP PP
NP → NP CC NP CC NP NP → NP S
NP → NP ADJP
NP → ADJP NP
NP → NP PP PP PP
NP → VP PP
NP → NP VP PP
NP → JJ NP
NP → ADJP NP
NP → ADJP NN CC NN

VP → VBZ NP
VP → VBG NP
VP → VBN S
VP → VBZ NP VP
VP → TO VP
VP → VBZ NP PP
VP → VP CC VP
VP → VBZ ADJP NP

VP → VBZ NP NP
VP → VBG NP SBAR
VP → VBG CC VBG PP
VP → VB ADVP
VP → VB PP PP
VP → VB NP ADVP
VP → VBG PP PP PP

ADJP → JJ
ADJP → JJ CC JJ
ADJP → JJ JJ
ADJP → ADJP CC ADJP
ADJP → RB JJ
ADJP → RB JJ CC JJ
ADJP → VBN CC VBN

ADVP → RB
ADVP → RB CC RB
Annex B - Sample of proof output

Case 1: Relations Preservation
Checking arguments ...

New Step 1
unify q[0]: person:n#1(x1) with person:n#1(x100)
Unification done.

New Step 2
unify q[1]: shoot:v#2(e1,x100,x2) with shoot:v#2(e100,x400,x100)
Unification done.

PROVE stopped. Reason: extra argument mismatching.
shoot:v#2(e100,x100,x2)
shoot:v#2(e100,x400,x100)

Backtrack ...
unify q[0]: person:n#1(x1) with person:n#1(x300)
Unification done.

New Step 2
unify q[1]: shoot:v#2(e1,x300,x2) with shoot:v#2(e100,x400,x100)
Unification done.

PROVE stopped. Reason: extra argument mismatching.
shoot:v#2(e100,x300,x2)
shoot:v#2(e100,x400,x100)
Backtrack ...

unify q[0]: person:n#1(x1) with person:n#1(x400)
Unification done.

New Step 2

unify q[1]: shoot:v#2(e1,x400,x2) with shoot:v#2(e100,x400,x100)
Unification done.

New Step 3

unify q[2]: billy_the_kid:n#1(x2) with billy_the_kid:n#1(x100)
Unification done.

Question PROVEN!!!

question predicates

person:n#1(x400) shoot:v#2(e100,x400,x100) billy_the_kid:n#1(x100)

answer predicates

nn:n#1(x100,x200,x300) outlaw:n#1(x200) william_bonney:n#1(x300) billy_the_kid:n#1(x100)
shoot:v#2(e100,x400,x100) nn:n#1(x400,x500,x600) sheriff:n#1(x500) pat_garrett:n#1(x600)
in:n#0(e100,x700) fort_summer:n#1(x700) rel:n#0(x700,x800) new_mexico:n#1(x800)
person:n#1(x100) person:n#1(x300) person:n#1(x400) person:n#1(x500) person:n#1(x600)

Question PROVEN!!!

Backtrack ...

Backtrack ...

unify q[0]: person:n#1(x1) with person:n#1(x500)
Unification done.

New Step 2

unify q[1]: shoot:v#2(e1,x500,x2) with shoot:v#2(e100,x400,x100)
Unification done.

PROVE stopped. Reason: extra argument mismatching.
shoot:v #2(e100,x500,x2)
shoot:v #2(e100,x400,x100)

Backtrack ...
unify q[0]: person:n #1(x1) with person:n #1(x600)
Unification done.

New Step 2
unify q[1]: shoot:v #2(e1,x600,x2) with shoot:v #2(e100,x400,x100)
Unification done.

PROVE stopped. Reason: extra argument mismatching.
shoot:v #2(e100,x600,x2)
shoot:v #2(e100,x400,x100)

Backtrack ...

Backtrack ...
REFERENCES


