<table>
<thead>
<tr>
<th>Title</th>
<th>Developing and applying an integrated semantic framework for natural language understanding (PhD Dissertation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Le, Tuan Anh</td>
</tr>
<tr>
<td>Citation</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>2019-07-16</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/10220/49370">http://hdl.handle.net/10220/49370</a></td>
</tr>
<tr>
<td>Rights</td>
<td></td>
</tr>
</tbody>
</table>
DEVELOPING AND APPLYING AN INTEGRATED SEMANTIC FRAMEWORK FOR NATURAL LANGUAGE UNDERSTANDING

LE TUAN ANH
SCHOOL OF HUMANITIES
2019
DEVELOPING AND APPLYING AN INTEGRATED SEMANTIC FRAMEWORK FOR NATURAL LANGUAGE UNDERSTANDING

LE TUAN ANH

SCHOOL OF HUMANITIES

A thesis submitted to the Nanyang Technological University in partial fulfilment of the requirement for the degree of Doctor of Philosophy

2019
Statement of Originality

I certify that all work submitted for this thesis is my original work. I declare that no other person’s work has been used without due acknowledgement. Except where it is clearly stated that I have used some of this material elsewhere, this work has not been presented by me for assessment in any other institution or University. I certify that the data collected for this project are authentic and the investigations were conducted in accordance with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

18 April 2019

Le Tuan Anh
Supervisor Declaration Statement

I have reviewed the content of this thesis and to the best of my knowledge, it does not contain plagiarised materials. The presentation style is also consistent with what is expected of the degree awarded. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accordance with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

18 April 2019

Francis Charles Bond
Authorship Attribution Statement

This thesis does not contain material from any paper published in any peer-reviewed journal where I was the first or corresponding author.

18 April 2019

Le Tuan Anh
Acknowledgements

This dissertation is based on my previous research titled *Question-Answering Machine based on Deep Linguistic Parsing* (Le and Ying, 2013). The software Visual Kopasu was further developed into this second version - the Visko software suite. For both projects, my supervisor and mentor Assoc. Prof. Francis Charles Bond guided me through the ideation process, brainstorming sessions, as well as solving many nuances of the implementation process. He also inspired and motivated me to pursue research as a form of personal development. I owe a great debt of gratitude to him for providing such a great opportunity to make this research happen at Nanyang Technological University.

For both projects, I received lots of help from my former lecturer and supervisor Fan Zhenzhen at Institute of Systems Science, National University of Singapore. Her patience and guidance throughout these years has helped to strengthen my knowledge and skills in computational linguistics, and to establish a strong foundation for my research career.

I would like to express my heartfelt thanks to my previous co-supervisor and mentor, Prof. Randy John La Polla, who could not continue to supervise my research project due to a long-term leave of absence from work during the last year of this research. Through the discussions and supervision sessions at NTU, Prof. Randy has showed me the depth and breadth of linguistics as both a science and an art. His teachings about understanding each theory is an answer to a certain problem, and that languages are actual things that are being spoken and written by human, and not just soulless characters or sound waves. Together with other valuable teachings, he kept reminding me about the scientific and humanistic values in our research.

I wish to express my utmost appreciation to my other co-supervisors, Asst. Prof. František Kratochvíl, Asst. Prof. Kim Jung-Jae, and Asst. Prof. Ivan Panović who helped me through my research journey in NTU. Their expertise and thoughtful comments during our meetings and discussions on different topics have greatly contributed to the success of this research.
This research was made possible by the English Resource Grammar. I cannot give enough thanks to Dr Dan Flickinger, who has guided me many times during his visits to NTU and throughout our discussions via emails, on how to configure, customize, and to make use of the grammar for different purposes. His friendliness, patience and expertise have helped me to appreciate from subtle nuances of grammar engineering to computational linguistics as a field.

Starting as a knowledge engineer with limited background in natural language processing and computational linguistics, I was not familiar with many sub-fields and practices in linguistics. My friend and lab mate, David Moeljadi, through numerous sharing sessions, has helped me in building up my knowledge, especially in phonetics, phonology, morphology, and typology. His sharing contributed a great deal to the completion of this research.

Due to the integrative nature of this project, I had to work with various technologies and software packages. I wish to thank to my friend and mentor Asst. Prof. Sanghoun Song, who has shared how to use and integrate results from different DELPH-in software products (ACE, LKB, TSDB++, et cetera) with the NTU computational linguistics laboratory. I also wish to thank to my friend Dr Michael Wayne Goodman, the original author of the pyDelphin package, for the package development and also for the many discussion sessions given by him during his visits to NTU. The pyDelphin package has simplified many tasks from HPSG parsing with ACE to manipulating different MRS-based formats.

During the process of developing lexical mappings and data projections between different data sets, I have received a lot of help from my friend and colleague Giulia Bonansinga, especially on the source code for reconstructing the surface text and sense projection algorithm. I would like to give my genuine thanks to Giulia, for the source code that she developed, as well as the emotional support that she gave during the difficult times during my research.

It will not be enough to say thanks to my friend and lab mate, Luís Morgado da Costa, who has reviewed and gave me many important critiques and feedback on this research from the very beginning of this research to its end. Many important changes in this research were resulted from his helpful comments during my formal and informal periodic review sessions. There were many times when I was stumped by the difficulties in research, but his friendship gave me the strength to keep going.

During the course of my research, I was given the chance to go on an internship to Fuji Xerox Japan, from February to March 2016, to work on Indonesian sentiment analysis. I
learnt a lot during this internship, under the guidance of Tomoko Ohkuma and Yasuhide Miura, on how to make use of many state-of-the-art research facilities and getting familiar with industrial standard research practices. I wish to express my sincere thanks to Fuji Xerox Japan for giving me such a great opportunity and experience.

After submitting the first version of this dissertation, I continued to receive more help from others, notably from three anonymous reviewers, Dr Alexandre Rademaker at IBM Research - Brazil, Seraphina Fong, Dr Hoang Duc Chinh, Serene Chau Huynh, and Hannah Choi. They provided many useful comments, which helped improve both my scientific arguments and formal writing.

This research could not have been possible without the full financial funding from the Tier 2 grant which was provided by the Ministry of Education of Singapore That’s what you meant: a Rich Representation for Manipulation of Meaning (MOE ARC41/13). I would like to express my heartfelt thanks to Singapore for giving me the chance to pursue this research project.

My research journey at Nanyang Technological University was very pleasant. I was supported by many of the staff, especially Ann-Marie Chua, Eng Qiao En, and Amutha Shanmugam, who have patiently guided me through various aspects of student life.

My deepest thanks to my wife and computer scientist Tham Thi Van Anh. She has discussed my research with me since the beginning, critiqued and debated my research arguments, given me much helpful feedback, read and corrected my manuscripts, and endlessly completes and supports me with love and dedication.

Finally, I dedicate my research to humanity, which has helped shape me into who I am today. This research is an attempt to contribute to the advance of our collective knowledge, from a heart full of love and gratitude.

Singapore
April 2019
## Contents

Acknowledgement iv  
Abstract x  
List of Figures xii  
List of Tables xiv  
Styles and Abbreviations xv  

1 Introduction 1  
1.1 Some Simple Examples 3  
1.2 Research Statement 5  
1.3 Major Contributions 5  
1.4 Thesis Structure 7  

2 Background 9  
2.1 Morphosyntax - The Study of Forms 11  
2.1.1 Construction Grammar 15  
2.1.2 Sign-Based Construction Grammar 16  
2.2 Semantics - The Study of Functions 22  
2.2.1 Aspects of Meaning 23  
2.2.2 Formal Semantics 24  
2.2.3 Cognitive Semantics 27  
2.2.4 Frame Semantics 28  
2.2.5 Summary 31  
2.3 Semantic Representations 31  
2.3.1 Descriptive Representation 32  
2.3.2 Abstract Meaning Representation 32  
2.3.3 Minimal Recursion Semantics 34  
2.4 Computational Semantic Analysis 38  
2.5 Word-Sense Disambiguation 40  
2.5.1 Original Lesk Algorithm 41
7 Integrated Semantic Framework  124
  7.1 ISF Workflow  124
  7.2 Sense Ranking with LeLesk  132
    7.2.1 Evaluation and Discussion  133
  7.3 ISF Software Package  135
  7.4 ISF Evaluation  138
    7.4.1 Evaluation Method  138
    7.4.2 Discussion  144
    7.4.3 Future works  149

8 Visko - Software Suite for Computational Linguistic Analysis  151
  8.1 ISF Corpus Structures  151
  8.2 Visualizing Composite Semantics  154
    8.2.1 Semantic Structures Visualization  156
    8.2.2 A Single Visualizer  156
  8.3 Searching and Indexing  157
  8.4 Interactive Development  158

9 Conclusions  159
  9.1 Deliverables  160
  9.2 Future Research Plan  161
    9.2.1 Study of English Constructions  161
    9.2.2 Better Sense Ranking Methods  162
    9.2.3 Multilingual Approach  162

Appendices  163

A Software and Resources  164

B Source Code  166
Abstract

The standard approach in Natural Language Processing for semantic analysis (Word-Sense Disambiguation, Named-Entity Recognition and other related tasks) is to match tokens from shallow parsed text (tokenized, POS tagged, shallow trunking, et cetera) to a sense repository and then rank them to find the best candidates. This practice has yet to exploit the extra information that is available in structural semantics, which can be accessed using deep grammars.

This dissertation proposes the Integrated Semantic Framework, a novel method to improve computational semantic analysis by providing both structural semantics from construction grammars and lexical semantics from ontologies in a single representation. The method was implemented as a software package that produces computational semantic analysis and its performance was compared to human annotators and some others semantic analysis systems on a short story.

Currently, the implemented system only provides analyses for standard English texts. However the design is extensible to other languages and is already being developed for the Japanese language. Finally, although the implemented system is still a prototype (most rules are generated automatically, the structure matching and transforming features are still at a basic level, and a few other tasks remain on the to-improve list), the results prove that such a system can be built and can produce positive results.

This research demonstrated that it is possible to provide a more natural and sophisticated computational semantic analysis. It aims to motivate linguists to join the development of fundamental semantic theories in the field of Natural Language Processing; to interpret and provide better semantics that exist in natural languages.
# List of Figures

2.1 Taxonomy of SBCG sign .................................................. 17
2.2 Sample of “Research” frame in FrameNet .............................. 30
2.3 Simplified Lesk ............................................................ 42
2.4 Semantic relations in Wordnet ........................................... 49
2.5 An example of the graph $G_1$ ........................................... 57
2.6 An example of the directed acyclic graph $G_2$ ......................... 59
2.7 A feature structure as an Attribute-Value Matrix ..................... 60

4.1 TTL as a Data Interchange Format for linguistic data ................. 79

5.1 Fragment parse #1 of “Study” .......................................... 85
5.2 Fragment parse #2 of “Study” .......................................... 85
5.3 Fragment parse #4 of “Study” .......................................... 86
5.4 root_wn_n parse of the noun “Study” ................................. 86
5.5 A sample ISF construction .............................................. 87
5.6 ISF Construction Database corpus design ............................. 89
5.7 ISF Construction for *tea chest* ....................................... 96
5.8 MRS for *This is a tea chest.* .......................................... 97
5.9 Dependency graph of *This is a tea chest.* ........................... 97
5.10 Tranformed MRS for *This is a tea chest.* .......................... 99
5.11 Dependency graph of *This is a tea chest.* (transformed) ........... 99
5.12 DMRS of *the new dish-washer* ....................................... 100
5.13 DMRS of *the new-dish washer* ...................................... 100
5.14 Construction for *to lose one’s mind* ................................ 103

6.1 MRS and DMRS structures for *a dog barked.* ....................... 105
7.1 Integrated Semantic Framework Flowchart .................................. 125
7.2 Different Semantic Analysis workflows ................................. 139
7.3 DMRS of I am happy. ............................................................... 140

8.1 Corpus management features in Visko software suite ................. 154
8.2 Document management features in Visko software suite .......... 155
8.3 TTL Visualization in Visko .......................................................... 156
8.4 DMRS Visualisation in Visko ....................................................... 157
8.5 DMRS interactive development .................................................... 158
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Syntax, semantics and pragmatics in linguistics</td>
<td>22</td>
</tr>
<tr>
<td>2.2</td>
<td>Gloss for bank in PWGC</td>
<td>50</td>
</tr>
<tr>
<td>4.1</td>
<td>TSV scanning</td>
<td>77</td>
</tr>
<tr>
<td>4.2</td>
<td>TSV reading</td>
<td>77</td>
</tr>
<tr>
<td>5.1</td>
<td>Princeton Wordnet 3.0 synsets</td>
<td>83</td>
</tr>
<tr>
<td>5.2</td>
<td>Construction database - Lemmas with constructions</td>
<td>90</td>
</tr>
<tr>
<td>5.3</td>
<td>Construction Database - Processed Lemmas</td>
<td>91</td>
</tr>
<tr>
<td>5.4</td>
<td>Construction Database - Rules by top pred count</td>
<td>92</td>
</tr>
<tr>
<td>6.1</td>
<td>Semcor parsed using ERG.</td>
<td>112</td>
</tr>
<tr>
<td>6.2</td>
<td>Semcor analysis results</td>
<td>112</td>
</tr>
<tr>
<td>6.3</td>
<td>Not Found predicates.</td>
<td>113</td>
</tr>
<tr>
<td>6.4</td>
<td>Sense-Pred Mapping Evaluation on 6146 senses</td>
<td>114</td>
</tr>
<tr>
<td>6.5</td>
<td>Sense-Pred Mapping Validation</td>
<td>116</td>
</tr>
<tr>
<td>6.6</td>
<td>Pred-Sense Mapping Evaluation</td>
<td>122</td>
</tr>
<tr>
<td>6.7</td>
<td>Pred-Sense Evaluation result using ERG first parse for The Adventure of the Speckled Band</td>
<td>122</td>
</tr>
<tr>
<td>7.1</td>
<td>WSD with Lesk for The Adventure of the Speckled Band</td>
<td>134</td>
</tr>
<tr>
<td>7.2</td>
<td>WSD with MFS for The Adventure of the Speckled Band</td>
<td>134</td>
</tr>
<tr>
<td>7.3</td>
<td>The Adventure of the Speckled Band - Lesk - Best of each</td>
<td>143</td>
</tr>
<tr>
<td>7.4</td>
<td>The Adventure of the Speckled Band - Lesk - Best of each</td>
<td>144</td>
</tr>
<tr>
<td>7.5</td>
<td>The Adventure of the Speckled Band - Lesk - Intersection</td>
<td>145</td>
</tr>
<tr>
<td>7.6</td>
<td>The Adventure of the Speckled Band - Lesk - No Filter</td>
<td>145</td>
</tr>
<tr>
<td>7.7</td>
<td>The Adventure of the Speckled Band - MFS - Best of each</td>
<td>146</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>7.8</td>
<td>The Adventure of the Speckled Band - MFS - Intersection</td>
<td>147</td>
</tr>
<tr>
<td>7.9</td>
<td>The Adventure of the Speckled Band - MFS - No Filter</td>
<td>147</td>
</tr>
<tr>
<td>7.10</td>
<td>Summary of ISF evaluation experiments</td>
<td>148</td>
</tr>
<tr>
<td>7.11</td>
<td>WSD results for Semeval 2015, Task 13</td>
<td>150</td>
</tr>
</tbody>
</table>
Styles and abbreviations

- **Bold** is used to denote technical terms.
- **Italics** is used for direct quoted sentences and literal examples.
- “**Double-quoted** “**italics**” “**text**” is used for quoting a single constituents, such as a token or a word.
- **Sans Serif font** is used for Wordnet special features such as synet IDs, sensekey and DMRS predicate names.
- **ITALICS CAPS** represent the actual semantics of a given word or phrase.
- **[Double square bracketed text]** is used to denote a formal semantic function.
- **Monospaced font** is used for source code, computer commands or computer-readable data in raw format.

Abbreviations

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Description</th>
<th>Abbr.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRS</td>
<td>Minimal Recursion Semantics</td>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>DMRS</td>
<td>Dependency MRS</td>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>ERG</td>
<td>English Resource Grammar</td>
<td>CSV</td>
<td>Comma-Separated Values</td>
</tr>
<tr>
<td>ACE</td>
<td>the Answer Constraint Engine</td>
<td>TSV</td>
<td>Tab-Separated Values</td>
</tr>
<tr>
<td>AMR</td>
<td>Abstract Meaning Representation</td>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>NTU-MC</td>
<td>NTU Multilingual Corpus</td>
<td>ISF</td>
<td>Integrated Semantic Framework</td>
</tr>
<tr>
<td>AVM</td>
<td>Attribute-Value Matrix</td>
<td>TTL</td>
<td>Text-Tagging Library</td>
</tr>
<tr>
<td>TFS</td>
<td>Typed-Feature Structure</td>
<td>POS</td>
<td>Part-of-speech</td>
</tr>
<tr>
<td>WSD</td>
<td>Word-Sense Disambiguation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In this dissertation, I present an alternative approach for a computational semantic analysis, which is defined as “using computational tools to determine the meaning of a given utterance”. Computational semantic analysis in itself is a broad area, and it cannot be examined comprehensively in a single dissertation. Instead, my focuses in this research include studying current approaches, coming up with an alternative method, and testing both options theoretically and practically. On the one hand, the meaning of everything is too broad, and one may remain skeptical about whether it is possible to deliver a theory that covers all aspects of meaning. On the other hand, it is lazy to claim that since the problem seems impossible to solve, it should be ignored completely. Contrary to these two positions, I aim to demonstrate that: first, the semantic analysis method presented here is capable of providing useful and practical results; and second, it will grow naturally with promising results in further research.

A study by the Google Books project (Taycher, 2010) estimated that by 2010, there were at least 129 million books in the world, and 30 million of which had already been digitized (Darnton, 2013). Facilitated by the rapid development of information technology, digital
documents have been introduced as a new medium for archiving knowledge alongside the printed media. New technologies and platforms, such as wikis, forums, blogging services, and social networks, have been making it easier and faster for people to create and share their knowledge. Billions of web pages have been built, and this number is still on the rise (Kunder, 2015). Searching, aggregating and analysing this massive amount of knowledge are inefficient if done manually. These processes can be highly improved with the aid of computational systems.

Better computational semantic analysis will advance the accuracy of the following applications:

1. Computer-aided language learning platforms: In language learning, an application can be used to perform grammar checks to help students during their practice sessions to fix trivial errors. The workload for the teachers can be reduced tremendously.

2. Semantic-based machine translation: In semantic-based machine translation, the source text is parsed to obtain semantic representation. This semantic representation is then translated into the semantic structures of the target language to have the output sentence generated.

3. Document summarisation: If computers could summarise long documents into shorter ones, reading and reviewing documents would be done more efficiently.

4. Information retrieval: When computers understand natural language inputs and perform semantic-based searches instead of keyword matching, the information retrieval process will provide more relevant search results.

5. True digital assistants: Computer systems with more sophisticated language comprehension and reasoning ability can help improve users’ productivity.
1.1 Some Simple Examples

In simple terms, “semantic analysis” is a process to find out the meaning (semantics) of something.

Consider the following sentence: *It rains.*

From the syntactic point of view, there are three constituents. First, the word “*It*”, a dummy subject, is there solely to meet grammatical requirements. Next is the verb “*rains*” which is conjugated in the third person singular present tense. The last element is a full stop, a punctuation mark to formally indicate the end the sentence. These are forms, the objects, and the clues to our investigation. The assumption is that they carry meanings, and the meanings can be determined.

(1) \textit{It \ rains}.

\texttt{SUBJ VBZ PUNC}

“It rains.”

What does (1) mean? Without extra contextual information, the common interpretation is that there is an event of raining that is happening. In (2), this event is denoted by the predicate \texttt{\_rain(e)}. This event evokes the concept of \textit{RAIN} in the mind of the hearer. This concept is not exactly the same as the concept of \textit{RAIN} in the mind of the speaker, but in an ideal case they should be close enough for the interlocutors to understand each other. The aspect in semantic analysis that deals with simple word forms is often referred to as \textbf{lexical semantics}, or the study of the meaning of words.
It rains.

Another aspect of semantic analysis is structural semantics. In structural semantics, the focus is the relations or complex structures that arise from composite structures.\(^1\)

In example (3), there is a drinking event \(e\) (predicate \(\text{drink}()\)) and two participants \(x1\) and \(x2\). The pronoun \(I\) \((x1)\) is the agent of the action, while \(\text{COFFEE}\) \((x2)\) is the patient. Finally, the modifier “\(every\ day\)” describes the frequency of the action \(\text{DRINK}\).

Semantic analysis is an open and difficult problem. In the following chapters of this dissertation, I explain how it is possible to provide a rich semantic analysis using computational systems, how such a system can be built, and how it can be applied in practical applications.

\(^1\)This is an informal explanation of structural semantics for introductory purposes. As elaborated later in this dissertation, the approach adopted here is construction grammar, in which the distinction between simple constructions (i.e. words) and compound constructions (i.e. phrases) is not clear.
1.2 Research Statement

The main purpose of this thesis is to propose an alternative semantic analysis method which can provide both lexical semantics and structural semantics using a single representation. This research does not stop at generating a theory; it extends to the application of computer software into analysing actual texts. Although the semantics is different from current systems, it is comparable to those systems on Word-Sense Disambiguation for evaluating the performance on actual use cases.

It is impractical to document all the possible usages for a language, let alone to provide formal analyses. Therefore, in this thesis, I neither aim to provide a comprehensive survey of constructions in English, nor do I aim to provide a complete list of semantic frames and relations. Instead, I focus on how to synthesize information from words and structures to further constrain the final analysis. For example, the target of a modifier (e.g. an adjective or an adverb) or the grammatical role of a word in a larger construction can be used to determine a more accurate semantic role (using word class, thematic role, et cetera) and thus more effectively constrain the list of potential sense candidates. In addition, I also propose a method to provide meanings for non-compositional phrasal constructions, such as some multi-word expressions or idioms. A preliminary implementation of this method is built into the final software to demonstrate that it can be implemented by using semantic graph demonstration.

1.3 Major Contributions

This research has several novel and unique contributions.

1. To the best of my knowledge, this is the first attempt to provide both lexical se-
mantics and structural semantics in a single integrated representation. This should be contrasted to the majority of current approaches that produce loosely combined syntactic relations, thematic relations, and token-based sense annotations.

2. The described computational framework was implemented as a software suite which is capable of providing semantic analysis for most arbitrary English sentences, up to the coverage of the underlying grammar. Other than the analyser, the software suite also contains various tools for integrating more resources as the research grows (such as a database of constructions in English, an ontology inspired by Wordnet and FrameNet for representing world knowledge and mental frames, et cetera).

3. The software suite was benchmarked on Word-Sense Disambiguation, a sub-task of semantic analysis, and compared to some state-of-the-art systems (UKB, Babelfy). Although this system has yet to implement more complex algorithms due to the constraints of software implementation, it produces comparable results to other systems. Regardless, the analysis produced by this method is different and qualitatively more accurate in some metrics.

4. Two construction mapping databases were developed using one semi-automatic method. The first database contains links between Wordnet phrasal constructions and the English Resource Grammar corresponding structures, and the second contains links from available predicates and Wordnet synsets.

5. Although the system is implemented for the English language, the framework works with any language given any grammar that produces MRS-based semantics and a lexicon in Wordnet structure. The system can be expanded further to support other ontology designs.
1.4 Thesis Structure

This dissertation consists of three main sections. The first section contains chapters 2 and 3, which provide the background knowledge and motivation for an alternative approach to semantic analysis. The second section, which includes chapters 4, 5, and 6, explains how to integrate resources and analyses from different data sets and systems in order to provide a semantic analysis through an unified representation. The last section, which includes chapters 7 and 8, discusses how the final framework is implemented and evaluated in comparison to other systems.

The structure of this dissertation is outlined as follows:

Chapter 2 - Background This chapter provides useful theories and knowledge for adopting computational semantic analysis. Readers may refer to this chapter for more information and for canonical citations when encountering unfamiliar terminology.

Chapter 3 - Research Approach This chapter discusses the differences between the semantic analysis method proposed in the present thesis and existing methods.

Chapter 4 - TTL This chapter introduces TTL – a linguistic documentation framework – the motivation for its development, its design, and its implementation. TTL is a theory-agnostic documentation formalism and it is flexible enough to manage large amounts of annotated linguistic data. Its implementation contains a set of processing tools and programming libraries which are suitable for different user groups, from field linguists to computational linguists.

Chapter 5 - Constructional Semantics There is more to semantic analysis than just lexical semantics (i.e. word meaning). This chapter explains how grammatical constructions and their corresponding semantics are recognised, and how to enhance
structural semantics which are generated from a HPSG grammar.

Chapter 6 - Concepts and Predicates The relation between concepts and predicates - the key point to integrating lexical semantics and structural semantics is discussed here. The Sense-Predicate integrating process consists of two main tasks 1) Finding suitable predicates for an annotated sense in a sentence and 2) Finding potential senses for any given predicate.

Chapter 7 - Integrated Semantic Framework The design, implementation, and evaluation of the ISF as a software suite are discussed in details. The performance of the ISF is compared to other systems (UKB, Babelfy) for the task Word-Sense Disambiguation on two gold data sets. One data set was developed at NTU, and the other was from the International Workshop on Semantic Evaluation.

Chapter 8 - Visko This chapter presents the implementation of the Visko software suite, which integrates all developed tools in a single system with graphical user interface. It can be used by both computational and non-computational linguists. The main tasks, such as batch processing, online processing, searching, and interactive development, are also explained.

Chapter 9 - Conclusions The final chapter discusses the main deliverables of this research, its practical applications, and how it can be improved further in future.
Chapter 2

Background

People communicate to share information, thoughts and views. There are many forms of communication (verbal, gestures, graphic symbols, shared code, et cetera), and linguistic communication is one of them. Linguistic communication typically comprises two sides: the speakers and the hearers. The linguistic act, initiated by the speakers, starts with a communicative intention, continues with the formation of intended meanings on their minds, and finally manifests itself in speech form to constrain the hearers’ interpretations. The hearers listen to and try to guess what the speakers are trying to communicate. In this process, there are intended meanings and interpreted meanings\(^1\) Linguistic meanings, or semantics, as studied in this research, are the messages that are communicated by using natural languages.

Before the nineteenth century, much of linguistics was prescriptive to codify the ‘correct’ way of using language Sag, Wasow, and Bender (2003, chapter 1, pp 7). From the late 19th century, other schools of linguistics arise from earlier approaches. One noteworthy approach is structural linguistics, in which languages are treated as systems whose parts

\(^1\)Not all models assumed this. Montague semantics, for example, assumed that everyone shares the same ontology and choosing the meaning is a trivial task, while it is not.
interrelate and interact organically. Ferdinand de Saussure, who founded structuralism, viewed linguistics as a science of language systems. He distinguished language (a formal and abstract language system) from parole (concrete utterances made by individuals) De Saussure and Baskin (2011).

To Saussure, to study language is to study a system of signs. A sign is composed of two elements: the signified (concepts or ideas) and the signifier (the means to express the signified) and the association between them. Saussure pointed out that the values of the signs are formed when they are contrasted to each other within structures. As a result, one cannot study signs in isolation from a language.

Noonan (1999) discussed the limitations of Saussure’s structuralism. Saussure assumed that there is a single abstract model of language (langue). In this model, categories are defined discretely and relationally, not substantively. This leads to difficulties in dealing with language variations, language changes, and dynamic language frameworks. Moreover, structuralism also did not make a clear distinction between knowledge of a language and knowledge of how a language is used.

Nevertheless, Saussure has pointed out the two main aspects in studying a language system: the study of the forms and the study of the functions. People understand linguistic messages because they are able to recognise forms and infer the intended meanings or functions. For a computer system to understand languages, it should possess a similar capability. In the following sections, I discuss different theories and frameworks for analysing forms and functions, with a focus on the ones that can be implemented in computational systems.

At the time of this research, I did not have access to the original publication “Course in General Linguistics” which was published in 1916.
2.1 Morphosyntax - The Study of Forms

The main topic in morphosyntax is the study of how humans use linguistic elements and combine them to form more complex structures. The first question for syntacticians is what the basic elements in syntax are. The second question is how more complex structures are constructed from these basic elements. Human languages are not a simple list of sentences, but a complex system which can be used to construct utterances. Let us consider the following examples:

(4) a. The dog *sleeps.*
    b. The dog *eats.*
    c. *dog the eat.*

Examples such as (4a) and (4b) demonstrate the generalisation characteristics of syntax in human languages. We clearly do not remember individual sentences, but instead construct them from simple elements using rule systems.\(^3\)

In addition, not every element in syntax can be combined. There are cases such as (4c), which is clearly incorrect when being judged by native English speakers. There must be restrictions or constraints as to why people do not use such utterances.

Therefore, a theory of syntax cannot simply define syntax as a list of all possible sentences. Instead, it should describe the construction rules and capture the generalisations in natural human languages.

Earlier approaches such as Context-Free Grammar (CFG) introduced in Sag, Wasow, and Bender (2003, chapter 2) assumed that the rudimentary elements of syntax are words,

---

\(^3\)Rule system here does not refer to phrase structure rules as in generative grammars. It is used in a broader sense to describe the combinatorial mechanisms which form complex structures in languages.
phrases, and the combinatorial mechanism of phrase structure rules. Words and phrases are divided into categories such as NOUN (N), VERB (V), ADJECTIVE (ADJ), et cetera for words and NOUN-PHRASE (NP), VERB-PHRASE (VP), ADJECTIVE PHRASE (AP), et cetera for phrases.

Phrases can be created by combining words and phrases together using phrase structure rules. For instance, we can analyse the syntactic structure of (4a) using the following syntax rules:

(5) A simple CFG grammar

- Word categories
  - DET=The
  - N=dog,food
  - V=sleeps,devours

- Phrase structure rules
  - S \rightarrow NP VP
  - VP \rightarrow V
  - NP \rightarrow N
  - VP \rightarrow VP NP
  - NP \rightarrow DET N

Using the grammar in (5) to analyse the sentence *the dog sleeps*, we have the syntactic tree as in (6).

(6)
As discussed in Sag, Wasow, and Bender (2003, chapter 2, pp36-40), one of the core problems with CFGs, such as the one described above, is over-generalisation. Using such rules will license sentences that are not well-formed in natural language. For example the following sentences are licensed by the grammar in (5) but rejected by native speakers of English:


b. *The dog devour food.

Sag, Wasow, and Bender (2003) pointed out that the traditional category of verb is too coarse-grained and verbs should be categorised into more subgroups of verbs. For instance, we can introduce more subclasses of verbs as INTRANSITIVE (IV), TRANSITIVE (TV), and DITRANSITIVE (DTV) and more rules into the grammar as in (8).

(8) Additional rules

a. VP → IV

b. VP → TV NP

c. VP → DTV NP NP
That being said, introducing more rules into the grammar does not always solve the problem. Considering the subject-verb agreement in English, the plurality marking on the verb and the subject must be consistent, such as in the (9):

(9)  a. The dog sleeps.
    b. The dogs sleep.
    c. *The dog sleep.
    d. *The dogs sleeps.

To account for this agreement by adding more rules, we will end up splitting every noun category and verb category into two subtypes of X-SG and X-PL (with X is V or N). However, this will double the amount of rules in the grammar, and we have yet to consider the transitive or ditransitive verbs. We will introduce a lot of unnecessary redundancy into the grammar and lose the power of the theory to capture generalizations. The agreement between the subject and the verb does not seem to depend on what type of verb it is. This leads us to finding a new way to define syntax classes and grammar rules.

Note: Transformational Grammars

Chomsky (1969) proposed a method to overcome this problem by introducing two different types of structure in syntax, namely Surface Structures and Deep Structures. The agreement problem discussed above can be solved by transforming trees to trees with the transformational grammar formalism.

However, transformations were proposed as a foundation mechanism for to analyse linguistic phenomena. There is no clear evidence to back up the claim that empty elements and tree transformations exist in natural syntax (i.e. in human grammar). As pointed out
in Sag, Wasow, and Bender (2003, Appendix B), many contemporary grammatical theories do not maintain the idea of doing transformations in syntactic analysis. Most of these theories are surface-oriented yet sufficient for performing linguistic analysis.

2.1.1 Construction Grammar

In surface-oriented structural analysis, syntactic structures of any utterance are linked directly to its surface level without any deeper structures and invisible elements. The combinatorial mechanisms which form more complex structures such as phrases and sentences are also more sophisticated than the phrase structure rules described in Section 2.1. Sag, Wasow, and Bender (2003, chapter 9, pp 299) claimed that humans actually use all kinds of information, including non-linguistic information, to process languages.

In response to the generative grammar theory by Chomsky, several modern grammar theories which follow the principles described above have been proposed, notably the construction grammars.

Construction grammar (CxG) is not a concrete linguistic theory but a family of approaches. There are different groups of linguists who are doing construction grammars in different ways. The approaches include, but are not limited to, Berkeley Construction Grammar (BCG) by Fillmore and Kay (1997), Radical Construction Grammar (RCG) by Croft (2001), and Sign-Based Construction Grammar (SBCG) by Boas and Sag (2012).

According to Croft (2007), construction grammars typically follow an essential principle that grammatical constructions are the basic units from which sentences are to be constructed. This view on construction grammars is also shared by Sag, Wasow, and Bender (2003, Section B.2.2, pp 534): Constructions are the basic linguistic elements and are “not only descriptions of words and phrases but also idioms and other types of collocations,

---

4the relations between the constituents of utterances only appear at surface levels.
characterizable at various levels of generality”. In contrast to Noam Chomsky’s theory of generative grammar, there are no empty categories or transformations in CxG as the basic combinatorial mechanism in these approaches is constraint satisfaction. Different information across multiple aspects (syntax, semantics, et cetera) is considered together to derive the generalisation of linguistic patterns.

Although many theories were proposed on how to build a construction grammar, not many actual implementations are available. To the best of my knowledge, the formalism that provides the most implementations is Sign-Based Construction Grammar (or its immediate predecessor HPSG), for example English Resource Grammar (ERG) or JACY Japanese Grammar (JACY). Solberg (2012) stated that the ERG is capable of parsing 88.38% of WikiWoods 1.0, a corpus of 55 million sentences of unrestricted text that was built from Wikipedia: this shows how much of the language is captured.\(^5\)

### 2.1.2 Sign-Based Construction Grammar

The idea of considering linguistic elements as signs (pairs of forms and functions) used in SBCG is not a novel one. It was first introduced by Ferdinand de Saussure since 1906. The difference, as explained in Boas and Sag (2012), is that “while the Saussaurian sign is an association between sound (signifiant) and meaning (signifie), the signs of SBCG embody more components”. As SBCG is a monostratal theory, which does not assume autonomous syntactic and semantic modules, every aspect of the linguistic analysis will be embodied within the structure of sign. SBCG features include, but are not limited to, phonological structures, orthography, morphological forms, syntactic features, semantics, and contextual factors, including information structure. As a result when describing a sign in the practice of SBCG, grammarians should consider providing the information in (10):

\(^5\)Source: [http://moin.delph-in.net/WikiWoods](http://moin.delph-in.net/WikiWoods)
Depending on the type of signs taken into account, the structure above will be extended by grammarians to fulfill their needs for describing certain linguistic phenomena. For example, the lexical sign, which is a type of sign that describes words and lexemes, needs to represent argument structures of the lexical entries. This information can be added into the lex-sign as in (11)

(11) \[
\begin{bmatrix}
\text{lex-sign} \\
\ldots \text{ARG-ST list} & \langle arg_1, arg_2, \ldots \rangle
\end{bmatrix}
\]

Figure 2.1: Taxonomy of SBCG sign

The taxonomy of SBCG signs proposed in Boas and Sag (2012) are shown in Figure (2.1). In SBCG, every linguistic component from words, phrases, to expressions, is a sign, and each sign has a corresponding feature structure typically following (10). It contains
phonology, form, syntactic information, semantic information, and contextual information. However, detailed features and values of actual structures depend on the linguistic analysis of the sign. The structure of signs is not hard-coded within the formalism of SBCG, but proposed by grammarians who study the language and provide a description of the language. The formalism of SBCG, meanwhile, focuses on providing a flexible framework for documenting various linguistic analyses. The major components of the framework include **feature structures**, for describing a sign’s structures, and **unification rules**, for building constructions.

Let us consider the subject-verb agreement problem in (9) in the sentence “dogs sleep.” and how this problem is solved within the framework of SBCG. In addition, I assume that we have the atomic constructions (i.e. lexemes and words) with the appropriate features. For example, (12) is one of the signs with the word form “dogs”.  

---

6 I constructed this example without the determiner to simplify the analysis, since covering many different linguistic phenomena is not the main focus here. What I am focusing on is demonstrating how atomic constructions are described and how more complex constructions are built in the framework of SBCG.

7 There are many different constructions and signs which have the orthography as the surface text form as “dogs” (i.e. different usages of dogs). (12) describes only one usage of the word dogs as “The animal dogs (canine)”.
In (13) we define a constraint on what can be combined with this verb and fill in the slot \textsc{subj-constraint} to form more complex structures (a sentence, in this case). Let the \textit{subj-constraint} be \textsc{NUM} of \textsc{sleep-1} matches \textsc{NUM} of \textsc{SUBJ} and syntactic category of \textsc{SUBJ} is noun (oversimplified constraints for demonstration purposes), we now can form more complex signs such as in (14).
These examples illustrate the basic process of analysing text using SBCG with constraint-based approach. Although SBCG proposes feature templates for signs, it separates clearly between the formalism and linguistic analysis. For example, the hierarchy of the lexicon and the syntactic categories such as VERB, NOUN, et cetera are not assumed as universal categories but will be proposed by grammarians who carefully study typological characteristics of the object languages. As a result, the competency of a grammar described in SBCG formalism in general depends on how well grammarians collected data and derive generalisations from those datasets.

From the examples above, we can also see that with so many features and construction rules, it is extremely difficult to verify the rules manually without a computational model.
Documenting a grammar computationally with SBCG therefore cannot be completed without implementation and automatic verification steps. The standard development process of SBCG consists of many iterations of the following steps:

1. Collecting text for reference corpus
2. Building the lexicon
3. Documenting grammar by studying a corpus
4. Large scale verification of the grammar on a corpus (ideally the verification data set and the data set used to study the language should be different)
5. Updating the lexicon and grammar and repeating

As this research does not focus on developing a SBCG-based grammar but make use of an implemented grammar, I will not describe further on this theory. Readers may explore Boas and Sag (2012) for details of the formalism.

2.2 Semantics - The Study of Functions

An important aspect of linguistic analysis is how to figure out what the utterances mean and how to represent the derived meaning. The study of languages and meaning described in Saeed (2009) can be summarised as in Table 2.1:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>The formal relation of (linguistic) signs to each others</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semantics</td>
<td>Meaning abstracted away from users</td>
</tr>
<tr>
<td>Pragmatics</td>
<td>Meaning described in relation to speakers and hearers</td>
</tr>
</tbody>
</table>

Table 2.1: Syntax, semantics and pragmatics in linguistics
Semanticists do not always agree on these divisions and it is difficult to draw a clear line between pragmatics and semantics. Just by looking at what a concept is, we already find several perspectives.

- One may assume that there is an objective meaning and define it by stating the necessary and sufficient conditions for the term. The definitions that are defined that way are called intensional definitions.

- We may also define concepts as lists of all included entities (for example *DOG* is a list of dogs in the world, *HUMAN* is a list of all homo sapiens, et cetera). These are extensional definitions.

- Concepts can also be defined as lists of some representative examples but not all in extension, (*DOG* means a list of some typical dogs, et cetera) These definitions are ostensional definitions.

In this section, I discuss various aspects of meaning, and then compare and contrast different approaches in the study of meaning.

### 2.2.1 Aspects of Meaning

I restrict **meaning**, the main subject of this work, to linguistic meaning. In this context, the study of meaning is the study of the relation between linguistic signs and the things being denoted. As presented in the coming sections, differences in hypotheses about the nature of what is being denoted result in different semantic theories. For now, let us consider the potential links between linguistic signs and their denotation.

A linguistic sign may represent anything, from a word, a phrase, to an idiom. The relation between linguistic signs and their denotations can be studied for different types of
signs. The branch of semantics concerned with the meaning of atomic linguistic elements (such as words, morphemes, et cetera) is called **lexical semantics**. Another branch of semantics which studies the meaning of larger structures and the relationship between the smaller structures and their parents structures (part-whole relationship) is called **structural semantics**. For a semantic theory to be completed, both these aspects of semantics should be accounted for.

Saeed (2009) identifies three challenges in doing semantics. They include:

1. **The circularity** in meaning definitions. In other words, how do we define the meanings of words? If we use the words to define the meanings of words, we also need to define the meaning of the words used in the definition. This looped definition goes on forever, hence the name “circularity”.

2. Ensuring exact definitions of meanings. Saeed (2009) gave an example in which one person understands that a whale is a fish and another person understands it as a mammal. Are they referring to the same thing when they use the noun *whale*?

3. Determining the meaning in **context**. How do we understand the pragmatic aspect of meaning, while the same utterance in different contexts may have a different meaning?

Therefore, the semantic theories, formalism, and frameworks studied in this section are discussed from the following angles: 1) what are the underlying theoretical hypotheses, 2) how well they cover lexical and structural semantics, and 3) how they overcome the three semantic challenges.

### 2.2.2 Formal Semantics

There are two major approaches to semantics and linguistics in general: the formal approach and the functional approach. The formal approach is formal semantics, also called
objectivist semantics. In this practice, meaning is independent from human mental processes such as attention, memory, or reasoning. Linguistic symbols, such as words, have meaning because they point to concepts (the denotations) in an objective reality. From this point of view, the circularity problem does not exist, because concepts are not defined as words but as entities in the objective reality.

The exact definition problem also does not exist, as meanings exist outside of human minds in the objective reality. For example, in Montague semantics (Dowty, Wall, and Peters, 1981) the meaning of the noun *whale* is defined to be WHALE, which is mapped to the corresponding concept in the semantic world. Montague semantics uses **First Order Logic** (FOL) as semantic representation, and this representation is centred around the idea that everything is predicate. The basic elements of FOL are: predicates, variables, operators, constants, and symbols with these following basic types being used: individuals (variables), constants (named entities), verbs (n-way predicates), nouns, and adjectives (one-way predicates). When presented in written form, the following notations are often used:

**Quantifiers** $\forall$ or $\exists$

**Variables** (e.g. $x_1, x_2, x_3$) hold values (e.g. individuals)

**Individuals** TOM, MARY, et cetera

**Predicates** $\text{dog}(x)$ or $\text{love}(x, y)$

**Operators** $\neg$ or $\implies$ or $\iff$ or $\land$ or $\lor$

Montague semantics assumes that we get lexical semantics by using a mapping function $f =$ to what already exists.
As a result we have: \([whale] = WHALE\)

In addition to that, each utterance is considered a logical statement in Montague semantics. So the semantics of an utterance is the truth value of its equivalent statement regarding the objective world. Montague semantics also assumes autonomous syntax and semantic modules. For example, given the sentences such as “dog barks.” a syntactic tree will be derived as \(S = NP \backslash \text{dog} VP \backslash \text{bark}\). Next, a semantic mapping function will be used to derive

\[
[S] = [NP \backslash \text{dog}] [VP \backslash \text{bark}] = [BARK(\text{DOG})] = 1
\]

The problem with Montague semantics as demonstrated here is that, first, it ignores the subjective meanings, or speakers’ and listeners’ meanings. Next, Montague semantics assumes the mapping function between syntactic elements and semantics elements. For example, given the syntactic form of a word we can automatically derive its semantic counterpart. In natural language this is not likely the case. For example, the meaning of the word \textit{bank} can be bank as the river bank (BANK-1) or the financial bank (BANK-2). As nothing is showed on the surface form, how do we derive the semantics in this case? In this case, our linguistic model contains two syntactic-semantic mappings:

\[
[bank] = \text{BANK-1}
\]
Montague semantics did not take into account that different lexical entries may share a surface form or even a syntactic role. Similarly, semantics of structure is also assumed to be known as the mapping models (for instance, the semantics of a verb phrase is a known function which takes semantics of its noun phrase as input). Ambiguities in both lexical and structural semantics are assumed to be trivial to resolve when they are actually not.

2.2.3 Cognitive Semantics

Many semanticists have rejected the formal semantics approach described in the previous section. They take the subjective point of view in describing semantics. Meaning, by itself, is not standalone as an entity but a product of human minds. The meaning is created in human minds and is therefore always subjective. This approach of semantic analysis is adopted in cognitive semantics and functional semantics.

The circularity problem in semantic analysis does not appear in cognitive semantics. The meanings, or the denotations, are the mental concepts inside human minds and not words. Since they are created inside our minds, meanings can vary between what the speaker says and what the hearer perceives. Cognitive semantics states that speakers and hearers understand each other thanks to the common linguistic and non-linguistic knowledge shared between them. When speakers produce an utterance, they are trying to constrain the interpretations of the hearers with respect to the current context. Based on what they perceive from the speakers and the context, the hearers guess what is conveyed by the speakers by performing abductive reasoning. When the similarity in linguistic skills and knowledge between the speaker and the hearer is large enough, they understand each other.
By viewing semantic analysis as the result of the speakers’ effort to constrain the possible interpretations of utterances and the guessing/ reasoning process of the hearers, we can explain why meaning is not absolute and why humans misunderstand each other all the time. The main criticism of the cognitive semantics approach is that it always gives a subjective individual point of view and therefore can only account for individual language (idiolect). It does not capture generalisations. However this is actually not a problem. First, even an idiolect is already a generalised version of language. It is a subjective generalisation from the experiences of that particular individual in contact with the language of other individuals in the immersed linguistic environment. Second, even when we take the practice of corpus linguistics in which many natural utterances are studied to form generalised claims about linguistic facts, it is still a subjective point of view. This is because it is a generalisation of the examined corpus, which is still only a snapshot of a language, similar to the individual language.

This is demonstrated in cases of native speakers of a language. Native speakers learn their language from the community in which they were raised. In order for other individuals in that community to understand them, they have to adapt and learn to construct sentences that minimize misunderstandings, assuming that their goal is to maximize the chance to deliver their intended message. This leads to the convergence state of idiolects in a linguistic community. This is the reason why we tend to trust judgements of native speakers as they tend to make similar judgements to other speakers as other members of the community.

2.2.4 Frame Semantics

Fillmore (2006) introduces a cognitive-based semantic theory called Frame Semantics. The main ideas behind Frame Semantics is that grammatical constructions such as words, phrases, sentences, et cetera have meanings because they are linked into conceptual struc-
tures in the mind. Let us consider the following examples (adapted from Fillmore (2006))

(17)

1. Jim’s only spent two hours on land today.

2. Jim’s only spent two hours on the ground today.

3. Jim’s only spent a few years on this Earth.

When we hear the first utterance in (17) we assume that Jim spent the rest of his time on a sea voyage. The second utterance brings us the assumption that Jim has spent the rest of his time in the air or on a flight. In the last utterance, we may assume that Jim is now in heaven or travelling in space.

Fillmore (2006) proposes a theory to account for this phenomenon. It posits that when we recognise certain signs (words, phrases, et cetera) and start to construct the meanings, certain mental frames will be triggered. For instance, when we hear the noun land and ground cognitive frames in which conventional pairs between land and sea, ground and air will be brought into the context. As the whole frame is considered we can easily conduct reasoning processes to make sense of what is happening.

Fillmore has introduced two different types of frames in the context of linguistic frames: syntactic frames and case frames. Syntactic frames are used to describe membership slots of certain word classes. In later works, Fillmore introduced case frames which describe the valence of verbs. More than just numbering the arguments that a verb can take in, Fillmore also propose semantic role descriptions for each argument, such as Agents, Themes, Goals, Sources, et cetera.

Fillmore then blended the ideas of syntactic frames and case frames into one single semantic theory. What we know, today, as Frame Semantics. Moreover, many lexical
(18) A [Researcher\textsubscript{INHERITANCE}] (an individual a group or institution) attempts to answer a [Question\textsubscript{SUBFRAME}] by means of consulting literature, observation, or conducting experiments in a particular [Field\textsubscript{SUBFRAME}] pertinent to the [Question\textsubscript{SUBFRAME}]. The [Question\textsubscript{SUBFRAME}] may be underspecified in the form of a [Topic\textsubscript{SEE-ALSO}].

Figure 2.2: Sample of “Research” frame in FrameNet

Items and frames structures in English have been documented in the FrameNet project (Baker, Fillmore, and Lowe, 1998). An example of the frame research is shown in Figure 2.2 with semantics roles marked.

Frame Semantics is also used as the semantic theory to model the semantic aspect of SBCG as stated by Boas and Sag (2012, pp 80). For example, the sentence Lee says that Garfield ate Mighty Mouse will invoke the frame in (19).

(19) \[
\begin{aligned}
\text{SPEAKER} & \quad \text{Lee} \\
\text{MESSAGE} & \quad \begin{aligned}
\text{eating-fr} \\
\text{INGESTOR} & \quad \text{Garfield} \\
\text{INGESTIBLE} & \quad \text{Mighty-Mouse}
\end{aligned}
\end{aligned}
\]

However, the original formalism introduced along with Frame Semantics has been blended with Minimal Recursion Semantics (Section 2.3.3). As such, the frame information is represented in a slightly different way as in (20), whereas the frame in (19) has been separated into two different structures. This is just the matter of how to implement a representation rather than a theoretical issue.
2.2.5 Summary

From my personal view, I do not see semantics as an objective system but as an emerging phenomenon of the complex interaction of an inter-subjective, multi-agent system. As a result, I do not aim to build a complete model of languages, if that even exists. That does not mean that there is nothing useful in formal semantics and other formal semantic theories. The computational semantics used in this research, a situation semantics based on Frame Semantics, is in fact formal semantics. The system currently built should be interpreted as a competent agent which is capable of detecting and interpreting linguistic phenomena in a certain way. Ideally, it is able to learn new knowledge and improve its performance over time.

2.3 Semantic Representations

Using a morphosyntactic theory and a grammar described within the formalism of that particular theory, one can provide precise structural analysis for a given utterance. Similarly, to provide semantic analysis for an utterance, we need a semantic formalism to represent the semantics described by that semantic theory. I hereby describe two commonly used computational semantic representations: Abstract Meaning Representation (AMR) and
Minimal Recursion Semantics (MRS). In this section, I survey these two representations as well as other important computational linguistic resources.

2.3.1 Descriptive Representation

It is possible to use natural languages to describe meaning. This is the standard practice when conducting knowledge or language documentation. For example, when writing a monolingual English dictionary or encyclopedia, English itself is used to explain meanings. It is assumed that the general readers understand a certain amount of English vocabulary and grammar. Thus, the authors write the material accordingly.

This method is not suitable for computational linguistics, however, for three reasons. First, we still have to solve the problem of circular definitions as the core meanings are grounded in words. Second, there is the problem of ambiguity, as natural languages are open to interpretation. And last, language change makes it is hard to keep semantic representation consistent.

Thus, although descriptive representations are very valuable in documenting human knowledge, we need a more suitable semantic representation for computational methods.

2.3.2 Abstract Meaning Representation

Abstract Meaning Representation was designed to provide fast manual semantic annotations to help create a semantic treebank faster. As introduced in Banarescu et al.

---

8. Other than the two theories mentioned here, there are semantic theories available, such as Discourse Representation Theory (Eijck, 1990). However, to the best of my knowledge, there is no theory that was implemented as a computational grammar model. As the goal of this research is proposing a method to provide rich semantics that can be developed into a computational model, some theories are left out, mainly for managing research scope and resources.

9. Treebanking is a method used in computational linguistics. It provides structures for various aspects of linguistic analysis (semantic roles and relations for semantic treebanking, phrase structure trees or dependency trees for syntactic treebanking, et cetera).
AMR structures (hereafter referred to as AMRs) are rooted and labelled graphs which represent the semantics of utterances.

Let us consider the following sentences taken from Banarescu et al. (2013):

(21)

- The man described the mission as a disaster.
- The man’s description of the mission: disaster.
- As the man described it, the mission was a disaster.

All of the three sentences in (21) have the same semantic representation as in (22).

(22)

\[
(\text{d / describe-01} \\
: \text{arg0 ( m / man )} \\
: \text{arg1 ( m2 / mission )} \\
: \text{arg2 ( d / disaster )})
\]

AMR can be too abstract sometimes, as the case in (22). It is true that the three sentences in (21) share certain similarities, but they are actually constructed in different ways and they convey different meanings. Without linking the semantics back to their syntactic structures, it is impossible to derive the original information structure from the AMR in (22).

We can observe from (22) that AMR structures are tree-based representations. They do not have any corresponding syntactic structure and it is difficult to figure out a precise
process of how the semantic representations are derived. In fact, Banarescu et al. (2013) explained that AMR relies on human annotators to decide which structures should be used for a given utterance. To the best of our knowledge, there is no precise mechanisms (i.e. formalism) of deriving semantic structures for AMR, which makes it difficult to implement a computational model.

According to Banarescu et al. (2013), there are a few limitations in the current AMR design. First, there is no universal quantifier. Second, the process of deriving AMR structures for sentences can be vague. A teacher is represented as a person who teaches but professor is not someone who professes. Without carefully designed guidelines, the semantics may end up with inconsistencies as in (23)

\[
\begin{align*}
\text{history teacher} & : (p / \text{per-son} : \text{arg0-of} (t / \text{teach-01} : \text{arg1} (h / \text{history}))) \\
\text{history professor} & : (p / \text{professor} : \text{mod} (h / \text{history}))
\end{align*}
\]

AMR is designed to be a fast way for capturing human annotations and AMR structures can be analysed by computer applications with minimal additional effort. At the time of this research, there is no available and robust parser which can produce AMR structure for English texts. Therefore, AMR is not yet suitable to be directly employed as a semantic representation for this research.

### 2.3.3 Minimal Recursion Semantics

Copestake et al. (2005) discussed four criteria for computational semantics which are:

**Expressive Adequacy:** linguistic meanings can be expressed correctly

**Grammatical Compatibility:** meanings must be linked to other grammatical informa-
tion (e.g. Syntax)

**Computational Tractability:** meanings can be processed and checked for semantic equivalence efficiently

**Underspecifiability:** underspecification should be allowed (leaving semantic distinctions unresolved).

These four criteria have been considered throughout the design of Minimal Recursion Semantics (MRS). The underlying assumption behind MRS is that the primary units of interest for computational semantics are **elementary predications** (EP), a single relation, and its associated arguments, for example, $\text{dog}(x)$, $\text{cat}(y)$ or $\text{chase}(x,y)$.

Each MRS semantic structure contains a bag of elementary predicates (i.e. the order does not matter), a bag of handle constraints to represent semantic hierarchy, and a top handler (i.e. the chosen “top” of a semantic graph). Each predicate has a list of arguments to represent semantic relations. In the predicate $\text{chase}(x,y)$, $x$ is $\text{ARG1}$, or the first argument and $y$ is the second, or $\text{ARG2}$. As order matters in list structures, it is possible to know that $\text{dog}(x)$ is the one that chases $\text{cat}(y)$, not the other way around. I will not go into detail on how these structures are determined as they are normally generated by a computational grammar. Readers who want to learn more on this topic are highly recommended to start with Copestake et al. (2005).

For example, the first sentence in (21) can be analysed in MRS as in (24).

---

---

---
a. Dependency Graph

The man described the mission as a disaster.

b. Predicate view

_1: the_q[x6]
x6: man_n_1[]
describe_v_as[ARG1 x6, ARG2 x9, ARG3, x10]

_2: the_q[x9]
x9: mission_n_1[]

_3: a_q[x10]
x10: disaster_n_1[]

We have 7 EPs in (24b) to represent the semantic structure in (24a). x6, x9, x10 represent three entities (man, mission, and disaster) and can be quantified by _1, _2, _3 (the, the, and a). The event describe is represented by describe_v_as with three arguments. The speaker, the one who described, is the ARG1 of the event and mapped to x6. The item being described is AGR2 and linked to x9. Finally, the characteristics used to describe the item x9 is ARG3 and mapped to x10.

(25) a. Dependency Graph

The man's description of the mission: disaster
b. Predicate view

_1:_the_q[x6]
x6:_man_n_1[]
_2:.def_explicit_q[x11]
e14:poss(ARG1 x11, ARG2 x6)
x11:_description_n_of[ARG1 x15]
_3:_the_q[x15]
x15:_mission_n_1[]
e3:_colon_v_id[ARG1 x11, ARG2 x20]
_4:udef_q[x20]
x20:_disaster_n_1[]

The second sentence in (21) can be analysed as in (25). The part *Description of the mission* is represented as two predicates x11 and x15. *The man* is represented as x6 and *disaster* as x20. Together, we have *the description of the man* (poss(man,description(mission))) as equal to disaster_n[[]].

(24) and (25) show that the two captured semantics structures preserve the original syntactic structures. The semantic predicates are also linked to the surface form, and we can refer to the syntactic elements that introduced them. However, MRS has not introduced a more abstract semantic layer to capture the similarities between different sentences in (21) automatically. Until such a semantic layer is introduced, we need to analyse the MRS structures and capture the abstraction manually and case-by-case.

In some systems, MRS structures are represented using Dependency MRS (DMRS) representation (Copestake, 2009). This representation is interchangeable with MRS standard representation, and can be done easily using computer programs, such as the Python package **pyDelphin**. Sometimes, the umbrella term **xMRS** is used to mention any MRS-based
representations.

2.4 Computational Semantic Analysis

In this section, I discuss a common work flow for semantic analysis existing approaches, such as the International standard for a linguistic annotation framework as discussed in Ide and Romary (2004).

Let us examine the following sentence.

(26) I saw a girl with a telescope.

Assuming that the systems have very limited access to the context where the utterance was made. They typically follow the approach described below.

The first task is to recognise the words within the sentence. In the given example, the sentence will be broken into a list of words: ['I', 'saw', 'a', 'girl', 'with', 'a', 'telescope', '.']. This task is normally referred to as tokenization. Next, an optional step in which recognised words are annotated with parts-of-speech. Different strategies can be adopted to perform this task. The popular option is using machine learning algorithms such as Conditional Random Fields methods or an Artificial Neural Network. The words can be lemmatized before, during, or after the POS tagging process. Following is the output for (26) obtained from the Stanford tool chain\(^\text{11}\) (tokenizer, POS tagger, and parser).

(27) I saw a girl with a telescope.

PRP VBD DT NN IN DT NN .

A syntactic tree is then built, as in 28:

\(^{11}\)http://nlp.stanford.edu:8080/parser/index.jsp
From the parse tree, labelled dependencies between words can be derived:

(29) root(ROOT-0, saw-2)
    nsubj(saw-2, I-1)
    det(girl-4, a-3)
    dobj(saw-2, girl-4)
    prep(saw-2, with-5)
    det(telescope-7, a-6)
    pobj(with-5, telescope-7)

The pronoun I is recognised as the subject and the noun girl as the direct object of the verb saw. The preposition phrase with a telescope is attached to the verb. Finally, participant roles labelling and Word-Sense Disambiguation algorithms are used to provide a complete semantics analysis, as shown in Example (30).

<table>
<thead>
<tr>
<th>Participant Roles</th>
<th>Word</th>
<th>Sense</th>
</tr>
</thead>
<tbody>
<tr>
<td>See-er</td>
<td>&quot;I&quot;</td>
<td>pron_1st_sg</td>
</tr>
<tr>
<td>See-ee</td>
<td>&quot;girl&quot;</td>
<td>10129825-n</td>
</tr>
<tr>
<td>Instrument</td>
<td>&quot;telescope&quot;</td>
<td>04403638-n</td>
</tr>
</tbody>
</table>
The actual implementation is different from system to system. However, in general, most CL systems are variations of what was described in (27)–(30), with each component being pipelined\textsuperscript{12} into the next.

\section*{2.5 Word-Sense Disambiguation}

A common method for semantic analysis is Word-Sense Disambiguation. The standard approach is to scan a given text and then split it into tokens. The tokens are treated as forms, and several shallow analyses (such as part-of-speech tagging or shallow syntax parsing) are used to generate additional features. The features are used to generate sense candidates for tokens, and then ranked to deliver the final semantics. WSD is formalized as follows: given a word and its context (e.g. the surrounding words) and a lexical semantic model of the language in question (e.g. a list of word-sense pairs), identify the best sense(s) for the given word. Following are the standard steps:

1. How to select sense candidates
2. How to rank the candidates to select best senses
3. How to evaluate the results

To take an example, we may obtain the meaning of the word \textit{bank} in the sentence: \textit{I go to the bank to withdraw money}. Assuming a simplistic semantic model\textsuperscript{13} which has two different senses for the word \textit{bank} as shown in (31):

\textsuperscript{12}Pipeline is a technical term which is used to describe a chain-like system architecture. The system is constructed from independent modules which are arranged in a linear order to process data. The output of a component in the system will be used as input for the next one.

\textsuperscript{13}In the semantic model used in this research, there are, in fact, many more senses for the word \textit{bank}. In addition, this model is not just a list of word-sense pairs but also contains the relations between concepts and senses.
A WSD is expected to select the first sense of bank.

One of the popular methods for WSD is Lesk. Lesk is a family of dictionary-based WSD algorithms which determine the correct senses of any given word by calculating the overlapping the context (e.g. other words in the same sentence) and the available gloss information of each sense. Since the original algorithm was introduced in Lesk (1986), there have been many attempts to enhance the performance and the accuracy of this algorithm – thus, it is often being referred to as a family of algorithms. The major Lesk algorithms are described as follows.

2.5.1 **Original Lesk Algorithm**

The original Lesk algorithm searches for all available sense candidates of every word in the context. For each sense candidate, we came up with a list of words based on its definition. Next, for each pair of \(W_1\) and \(W_2\), we calculate the overlap (i.e. number of similar words) between the two definitions. If we have \(m_1\) senses for \(W_1\) and \(m_2\) senses for \(W_2\), we will have \(m_1 \times m_2\) sense pairs checked. The more words need to be checked in the list, the higher the number of sense pairs are to be considered.

2.5.2 **Simplified Lesk**

According to Vasilescu, Langlais, and Lapalme (2004), a variation of this algorithm known as **Simplified Lesk** not only has higher precision but is also much faster compared to the original version. In Simplified Lesk, we do not find the most overlapped sense-pairs to
identify matched senses. Instead, for each word in a context, we calculate the overlap score between the words in the gloss of each sense and the context (i.e. words) surrounding the word we are trying to disambiguate. The sense with the highest score will be assigned to the words. The pseudocode of Simplified Lesk is given in Figure 2.3.

\[
\text{for each sense } s_i \text{ of word } w_j:
\quad \text{score}_i = \text{overlap(definition of } s_i, \text{ context of } w_j) \\
\quad \text{if score}_i > \text{max_score:} \\
\quad\quad \text{max_score} = \text{score}_i \\
\quad\quad \text{best_sense} = s_i
\]

Figure 2.3: Simplified Lesk

For example, when we apply Simplified Lesk to solve the ambiguity in (31), the best sense for \textit{bank} in the sentence above is \texttt{bank}_n_1, as detailed in the steps below:

\begin{verbatim}
word = 'bank'
context = ['I', 'go', 'to', 'the', 'bank', 'to', 'withdraw', 'money']
bank_n_1 = ['a', 'financial', 'institution', 'that', 'accepts', 'deposits',
              'and', 'channels', 'the', 'money', 'into', 'lending', 'activities']
bank_n_2 = ['sloping', 'land', 'especially', 'the', 'slope', 'beside', 'a',
              'body', 'of', 'water']
score_1 = overlap(bank_n_1, context) # ['money', 'the']
score_2 = overlap(bank_n_2, context) # ['the']
# So the best sense for 'bank' is bank_n_1
\end{verbatim}

2.5.3 Extended Simplified Lesk

Banerjee and Pedersen (2003) introduced an enhancement over Simplified Lesk by extending the number of words that are used in the overlapping step of the algorithm. They real-
ized that the definitions provided by linguistic resources (Wordnet, dictionaries, et cetera) are short, and in most cases, do not contain sufficient vocabulary to make fine-grain distinctions in relatedness. This often results in low overlapping count between definitions of words and their context.

When solving a WSD problem, the ideal lexical semantic model should provide potential sense candidates along with related information on how they differ from each other. Originally, Lesk (1986) used Oxford Advanced Learner’s Dictionary senses such as cone_n1, cone_n2, pine_n1, et cetera. This basic semantic model is limited to a simple list of word-sense pairs and does not provide more information on the senses and their relations. In their version of Lesk, Banerjee and Pedersen (2003) used Wordnet as their semantic model, instead of a dictionary. With this, they were able to exploit the sense relations (hypernyms and hyponyms) to extend the word list available for the overlapping step for each sense candidate. The token list for a sense candidate in this case will be words in its definition, as well as in the definitions of its hypernyms and hyponyms. In (31), for example, bank_n_1 has one hypernym (i.e. financial-institution_n_1) and two hyponyms (i.e. agent-bank_n_1 and state-bank_n_1).\(^\text{14}\) (32) shows the actual steps of deriving token lists for bank_n_1:

\[
\begin{align*}
\text{bank}_n_1 &= ['a', 'financial', 'institution', 'that', 'accepts', 'deposits', \\
&\quad \text{and}, 'channels', 'the', 'money', 'into', 'lending', 'activities'] \\
\text{hype(bank}_n_1) &= [\text{financial-institution}_n_1] \\
\text{hypo(bank}_n_1) &= [\text{agent-bank}_n_1, \text{state-bank}_n_1] \\
\text{financial-institution}_n_1 &= ['an', 'institution', 'public', 'or', 'private']
\end{align*}
\]

\(^{14}\)In this example the sense relations are simplified for demonstration purpose. In reality, Wordnet 3.0 has more than two senses for bank_n_1.
By using the sense relations, the token list has been extended with richer vocabulary such as financial, funds, assets, et cetera, thus increasing the chance of word matching in the given context.

### 2.5.4 Enhanced Extended Simplified Lesk

From the development of Lesk algorithms in the past (from original Lesk to Simplified Lesk and extended Simplified Lesk), we observed that by using more complex semantic models with richer information (e.g. from dictionaries to Wordnet) we can obtain better disambiguation power.

Following the idea put forth by Baldwin et al. (2010) for solving Word-Sense Disambiguation in Japanese, I have exploited the Princeton Wordnet Gloss Corpus for additional available information to further extend the token list of each sense candidate. In addition to the words selected by the Extended Simplified Lesk, I use two other sources to increase the number of words available for the overlapping step:

1. tokens from the provided example sentences of each synset.
2. tokens extracted from hypernyms and hyponyms of the candidate sense as well as hypernyms and hyponyms of every tagged synset in its gloss.

### 2.5.5 WSD Evaluation with Recall, Precision, and F1

Word-Sense Disambiguation task is often evaluated using Recall, Precision and F1 score, the higher the better. Firstly, the evaluator counts the number of annotated senses generated by the system (\(count_{profile}\)) and the gold profile (\(count_{gold}\)). Secondly, True Positive, the number of correctly annotated senses predicted by a system compared to a gold profile will be calculated. This is done by evaluating the intersection between the senses set by the output profile and the gold profile. Finally the scores are calculated by using the following formulae:

\[
\text{precision} = \frac{TP}{count_{profile}}
\]

\[
\text{recall} = \frac{TP}{count_{gold}}
\]

\[
F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}
\]

For example, if a system generates seven senses (\(count_{profile}\)) for a document, when there are 10 senses (\(count_{gold}\)) in the gold profile. Out of the seven predicted senses, five senses are actually correct (i.e. in the gold profile). These are True Positive. The final scores will be:

\[
\text{precision} = \frac{5}{7} = 71.43\%
\]

\[
\text{recall} = \frac{5}{10} = 50\%
\]

\[
F1 = 2 \times \frac{71.43 \times 50}{71.43 + 50} = 58.82\%
\]
2.6 Linguistic Tools and Resources

2.6.1 Wordnet

Representing meanings of words in utterances is an important feature of linguistic resources. Traditionally, this is covered by creating dictionaries, which are collections of word-meaning pairs organized in a lexicographical order. For example, when we look up the word *dog* in the dictionary, we learn about its basic parts of speech and different usages as in (33).

(33)

*dog_n_1* a member of the genus Canis

*dog_n_2* someone who is morally reprehensible

*dog_v_1* go after with the intent to catch

...

However, this form of documenting lexical semantics has limitations. For instance, dictionaries often have few indexing orders, most of the time in one index (alphabetical order) and sometimes two indices (such as kana and kanji stroke order in Japanese dictionaries). In addition, when we look up a word, we have very limited information about its usage, the constructions in which it often appears, and its relations to other words. One may always add in more information and indices, but this information can be overwhelming and too large to be printed into a physical copy.

With the power of computer systems, alternative forms of lexicons have been introduced. One of the most well known is Wordnet (Fellbaum, 1998).
The first Wordnet, Princeton Wordnet, is an English semantic lexicon which organises meanings as a network of concepts that are linked by semantic relations (hyponymy, hypernymy, et cetera). Introduced in Miller et al. (1990), Wordnet’s design is inspired by psycholinguistic theories on how humans organise concepts in their mind. After many iterations of development, Wordnet has evolved into a massive English lexical database with more than 206,941 Word-Sense pairs in the current version.\footnote{Taken from https://wordnet.princeton.edu/wordnet/man/wnstats.7WN.html on 15th May 2015} It contains not only information on meanings of words, but also the relations among them.

Words that refer to the same concept (have the same meaning) are grouped together in a synonym set (or synset). A word may also have different usages which can be used to convey different meanings. Therefore, the relation between forms and meanings in Wordnet is many-to-many relationship. Wordnet researchers organise these relations with a taxonomy of concepts. Each concept has its own definition and an identifier string (we use an eight digit offset number and a part of speech). One or more appropriate word forms are linked to the created concept.

Example (34) shows some sample synsets and linked word forms. Here we have two synsets for representing two different concepts. 01698271-v represents the action \textit{to write} and is classified with v while 03906997-n is classified as n and refers to \textit{pen}. These two synsets are mapped to their corresponding word forms. 01698271-v is linked to \textit{write, compose, pen}, and \textit{indite} while 03906997-n is linked to \textit{pen}. When we look at this sample from the conceptual point of view, we have two concepts which can be conveyed using different words. In contrast, when we look at it from the word-form point of view, we can see that \textit{to write} and \textit{to compose} can be used to refer to the same concept while \textit{pen} can be used to refer to both concepts.

(34)
• Synonym sets

*01698271-v* : to produce a literary work

*03906997-n* : a writing implement with a point from which ink flows

• Word forms

*01698271-v* → write, compose, pen, indite

*03906997-n* → pen

When a word form is linked to a synset, they create a sense. The formal structure of a sense is a triplet 
\((\text{synsetID}, \text{lemma}, \text{sensekey})\). For example, the synset *00648224-v* which means *inquire into* (Princeton Wordnet 3.0) is linked to three different lemmas “*explore*”, “*research*” and “*search*”. Thus, there are three senses that are linked to this particular synset: 
\((00648224-v, \text{explore}, \text{explore}^{\%2:31:00::})\), \((00648224-v, \text{research}, \text{research}^{\%2:31:00::})\), and \((00648224-v, \text{search}, \text{search}^{\%2:31:00::})\).

Another important feature of Wordnet is the relations among concepts. The basic relations in the original design of Wordnet are antonymy, hyponymy and meronymy. Antonymy is a contrastive relation between two senses (but not necessarily logical negation). For example *hot* and *cold*, *rise* and *fall*, *husband* and *wife*, et cetera. Hyponymy is hierarchical relation. If we can say *B is a kind of A* for two concepts A and B, we say they have a hyponymic relation in which B is a hyponym of A and A is a hypernym of B. Lastly, meronymy is part-whole relation. Figure 2.4 demonstrates how these relations are used in Wordnet (Fellbaum, 1998). Arms and legs are parts of a whole, which is the body, so we say they have a meronymic relation with body. Organic substance is a type of substance so it is a hyponym of substance while substance is the hypernym of organic substance.
2.6.2 Princeton Wordnet Gloss Corpus

Wordnet Gloss Disambiguation project released a special version of Wordnet which uses Wordnet information on senses to disambiguate Wordnet itself. All the glosses and examples of synsets are tokenized and chunked (e.g. multiword expressions). Constituents in the tokenized sentences are annotated using synsets when available. Table 2.2 shows an example of the annotated gloss for a synset of *bank*. There are five words that have been annotated using Wordnet synset IDs.\(^{16}\) By following the linked synset IDs we can acquire their definitions. In turn, these definitions are also annotated using other synsets. As a result of this nested succession, the relations among synsets are extended as a synset can contribute to disambiguating meaning (definitions) of other synsets.

The annotated data (known as Princeton Wordnet Gloss Corpus - PWGC) is based on Wordnet 3.0 and provided in text format.\(^{17}\) Out of 59,250 types\(^{18}\) annotated in the corpus, 33,862 types are annotated manually and 26,139 types are automatically anno-

\(^{16}\)Wordnet synset ID is a unique string that acts as an identifier for each synset in Wordnet. This ID consists of an unique eight digit number and a part-of-speech (n for nouns, v for verbs, a for adjectives, and r adverbs).

\(^{17}\)http://wordnet.princeton.edu/glosstag.shtml

\(^{18}\)Types are distinct words. For example in the sentence *The big dog chases the small dog* there are 7 occurrences and 5 types (*the*, *dog*, *big*, *small*, and *chase*).
tated. The number of annotated occurrences is 458,825, including 339,969 manually and 118,856 automatically annotated tokens. The annotation accuracy can be considered high, as about three quarters of the tokens are annotated manually. Despite the large coverage of sense annotation in the gloss corpus, some of the important words in the definitions remain untagged. In Table 2.2, the words “deposits” and “activities” are content words and important to the definition, but they have yet to be sense tagged, even by the automated tagger. In my future work, I am planning to use the results from this research to improve the gloss corpus. This will contribute directly to the performance of the Lesk algorithm.

<table>
<thead>
<tr>
<th>Word</th>
<th>Synset ID</th>
<th>Annotated Gloss</th>
</tr>
</thead>
<tbody>
<tr>
<td>a financial institution</td>
<td>08054721-n an institution (public or private) that collects funds (from the public or other institutions) and invests them in financial assets</td>
<td></td>
</tr>
<tr>
<td>that accepts</td>
<td>02236124-v receive willingly something given or offered</td>
<td></td>
</tr>
<tr>
<td>deposits and channels</td>
<td>01933900-v direct the flow of money</td>
<td></td>
</tr>
<tr>
<td>the money</td>
<td>13384557-n the most common medium of exchange; functions as legal tender;</td>
<td></td>
</tr>
<tr>
<td>into lending</td>
<td>02324182-v give temporarily; let have for a limited time;</td>
<td></td>
</tr>
<tr>
<td>activities</td>
<td>:Not tagged:</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2: Gloss for bank in PWGC
2.6.3 NTU Multilingual Corpus

NTU Multilingual Corpus (NTU-MC) is a compilation of multilingual texts available within Singapore (Tan and Bond, 2011). The version introduced in Tan and Bond (2011) contains 375,000 words from 15,000 sentences in 6 languages (English, Chinese, Japanese, Korean, Indonesian, and Vietnamese). The corpus has been built semi-automatically using various tools (tokenizers, part-of-speech taggers, et cetera) as well as human verifications.

Other than the high linguistic diversity, the data is also annotated in NTU-MC. Every sentence is tokenized to words and each word is tagged with a part-of-speech. Many words are also annotated by human annotators using Wordnet synsets and the IMI system (Bond, Morgado da Costa, and Lê, 2015). In addition, the documents are aligned between languages up to sentence level. Thus, this data is suitable for many computational linguistic tasks such as word classes learning or word sense disambiguation.

2.6.4 English Resource Grammar

As introduced in Flickinger (2000), English Resource Grammar (ERG) "is a broad-coverage grammar of English in the Head-Driven Phrase Structure Grammar (HPSG) framework which can be used in a range of applications". As noted by Boas and Sag (2012), with a few straightforward modifications in notations, HPSG is compatible with SBCG. In fact, the latest version of ERG has been upgraded to follow SBCG formalism.\footnote{The version I have mentioned is revision 22197 which have been accessed on 15 May 2015 and available at \url{http://svn.delph-in.net/erg/trunk/}} Using the ERG, we are able to provide both syntactic and semantic analyses for English sentences.\footnote{The semantic analysis here is very different from the one introduced in Section 2.1.2, as it is the output of an actual implemented grammar rather than standard guidelines from the SBCG formalism.} For example we can parse the sentence \textit{Dogs chase cats}. The result can be viewed from the syntactic angle as in (35). The tree structure view in (35a) can be misleading, as it is just
a tracing of the unification process, in which different signs are combined to form more complex structure. In the actual process of parsing, the syntactic structure and semantics are built at the same time as they are two different aspects of the same linguistic signs documented in the grammar.

(35) a.

```
S
  /\  
NP  VP
  /   /
Dogs V  NP
    /   /
   chase cats
```
In addition, the result also provides semantic analysis. As shown in (35b), there is a relation of “dog” and a relation of “cat” and an event of “chasing”. In the chasing event, the first argument refers to the entities called dogs and the second argument refers to the entities called cats. The syntactic view of a sentence demonstrates how a sentence is actually formed from smaller constituents, while the semantic view gives us information about how the events and concepts in the scenarios are being perceived. An important point
about these analysed results is that they are only helpful visualizations of the completed sign which is too complex to be displayed within a single page. The syntactic tree or MRS matrix shown above is the projected information from a more complex structure rather than parts of the actual formalism.

2.7 Technical Concepts

In this section, I explain some of the technical terms that may not be familiar to readers who are not from computational linguistics.

2.7.1 Data Types

Collections

When we make a group of items, we call it a collection. There are several different data structures that are capable of representing collections (a group of individual items). I will not explain each of them in detail, but provide a quick overview of them.

Bags

When we have a group of items and the order does not matter, we call it a bag. In this thesis, I use round brackets to represent bags. For example:

\[ \text{bag}_1 = (1, 2, 3) \]
\[ \text{bag}_2 = (3, 2, 1) \]
\[ \text{bag}_3 = (3, 2, 1, 1) \]

As order does not matter, \( \text{bag}_2 \) is identical to \( \text{bag}_1 \). \( \text{bag}_3 \) is different from \( \text{bag}_2 \) as it contains two items 1 compared to one in \( \text{bag}_2 \).
Sets

When we have a bag of distinct items, we call it a set. Similar to a bag, a set is not ordered. I use curly brackets to represent sets. For example:

\[ set_1 = \{1, 2, 3\} \]
\[ set_2 = \{2, 1, 3\} \]

Because order also does not matter in sets, \( set_1 \) equals to \( set_2 \).

Lists, sequences and tuples

When we have an ordered group of items, we call it a sequence. When extra elements can be added, it is called a list. If the number of elements is fixed, it is called a tuple. This is similar to the concepts of tuples in mathematics (sometimes it is also called sequence, ordered list, but this convention is not adopted in the context of this dissertation).

Similar to mathematics convention, \( n \)-tuple denotes a tuple which consists \( n \) elements. A tuple with \( n = 2 \) is called an ordered pair. And when \( n = 3 \) it is called a triple or a triplet.

When a list contains computer characters, we call it a string (this is elaborated in the following section).

Square brackets are used to denote lists and quoted text for strings. Here are some examples:

\[ l_1 = [1, 2, 3, 4] \]
\[ l_2 = [1, 2, 4, 3] \]
\[ str = \text{“I am a string”} \]

Nested structures are possible with lists, for example:
\[ p_1 = \{\text{“tom”, “Jerry”}\} \]

\( p_1 \) is an ordered pair which contains two strings (list of characters).

Also, order is important in lists. For instance, \( l_1 \) and \( l_2 \) are not equal.

To simplify the language in this research, ordered lists will be called lists, ordered pairs will be called pairs, except in direct quotes.

**Character string**

A text string is a list of characters.

I am a text string.

0123456789012345678

When talking about a substring (i.e. a part of a string), sometimes a pair of \((c_{from}, c_{to})\) will be used. \( c_{from} \) and \( c_{to} \) are from-character index and to-character index respectively.

The indices are zero-based, in which \( c_{from} \) is inclusive and \( c_{to} \) is exclusive. In the example above, the substring \(<7:18>\) of the original string will provide a new string text string. This convention is used throughout this dissertation.

2.7.2 **Graph**

Graphs are mathematical structures that consist of nodes and links. It is a useful tool in formal modeling. Bondy and Murty (1976) provided a formal definition for graph.

A graph \( G \) is an ordered triple \((V(G), E(G), \psi_G)\) consisting of a nonempty set \( V(G) \) of vertices, a set \( E(G) \), disjoint from \( V(G) \), of edges and an incidence
Figure 2.5: An example of the graph $G_1$

**function** $\psi_G$ that associates with each edge of $G$ an unordered pair of (not necessarily distinct) vertices of $G$. If $e$ is an edge and $u$ and $v$ are vertices such that $\psi_G(e) = uv$ then $e$ is said to join $u$ and $v$; the vertices $u$ and $v$ are called the **ends** of $e$.

Following is an example of a graph $G_1$.

Let us define $G_1$ as: $G_1 = (V(G_1), E(G_1), \psi_{G_1})$

Where

- $V(G_1) = \{v_1, v_2, v_3, v_4\}$
- $E(G_1) = \{e_1, e_2, e_3, e_4, e_5\}$
- and $\psi_{G_1}$ is a map that
  
  $\psi_{G_1}(e_1) = (v_1, v_2), \psi_{G_1}(e_2) = (v_2, v_3), \psi_{G_1}(e_3) = (v_2, v_4)$
  
  $\psi_{G_1}(e_4) = (v_1, v_3), \psi_{G_1}(e_5) = (v_4, v_4)$

There are four vertices and five edges in $G_1$. $\psi_{G_1}$ states which vertices each edge joins.

It can be easier to understand what a graph is by visualising it. We can imagine a graph as nodes (vertices) in space that are linked with edges. $G_1$ can be visualised as in Figure 2.5.
When an edge joins a vertex to itself, we call it a **loop**. In $G_1$, edge $e_5$ is a loop.

When there is at least one complete cycle in a graph, we call that graph a cyclic graph. For example in $G_1$, we have two complete cycles: $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_1$ and the loop $v_4 \rightarrow v_4$. If we remove all cycles from a cyclic graph, we will have an acyclic graph.

**Directed Acyclic Graph**

As defined above, edges are unordered pairs in a graph. This means $e_1$ is a bidirectional link that joins $v_1$ and $v_2$. If we change the definition to make edges ordered pairs, we will have a **directed graph**. For example, in graph $G_2$ defined below, function $\psi_{G_2}$ returns an ordered pair instead of a bag for each edge given.

A **directed acyclic graph** is a graph where edges have directions and no cycles within.

Let us define $G_2$ as: $G_2 = (V(G_2), E(G_2), \psi_{G_2})$

Where

- $V(G_2) = \{v_1, v_2, v_3, v_4\}$
- $E(G_2) = \{e_1, e_2, e_3\}$
- and $\psi_{G_2}$ is a map that
  \[
  \psi_{G_2}(e_1) = [v_1, v_2], \psi_{G_2}(e_2) = [v_2, v_3], \psi_{G_2}(e_3) = [v_2, v_4]
  \]

Figure 2.6 visualises $G_2$.

**2.7.3 Typed Feature Structures**

The full formalism of **Typed Feature Structure** (TFS) is discussed in Carpenter (1992). In this section, I only cover the basics of Typed Feature Structures to elucidate the theories
in this research. Readers are encouraged to read the original publication for a better understanding of the formalism.

**Feature Structures**

Feature structure $F$ is an ordered pair $[Q, V]$ in which $Q$ is a set of features and $V$ is a a mapping that provides value for any valid feature. For example, the following feature $f_1$ represents a book information.

$$f_1 = [Q_1, V_1]$$

$Q_1 = \{TITLE, AUTHOR\}$

$V_1 = \{TITLE \rightarrow \text{“The speckled band”}, \ AUTHOR \rightarrow \text{“Sir Arthur Conan Doyle”}\}$

Two common notations to represent a feature structure are graph and **Attribute-Value Matrix** (AVM).

For example, $f_1$ can be represented as a directed acyclic graph as below:

It also can be represented as an AVM as in Figure 2.7.
Figure 2.7: A feature structure as an Attribute-Value Matrix

An AVM may contain nested feature structures in which one feature has another feature structure as a value. For example, the value of feature *AUTHOR* in the following AVM is another feature structure (to represent information about the author).

```
[TITLE  "The speckled band"
AUTHOR  "Sir Arthur Conan Doyle"
```

2.7.4 Type Network

A type network is a multiple inheritance hierarchy in which types inherit features from parents. For example, we can have a hierarchy like this:

```
Entity

Animates

Insects  Beasts

Human

Inanimates

Thing

Writer  Student  Book  Computer  Clock
```

It is also possible to have multiple inheritances.

Given a feature structure, we can assign a type for it and thus put it in a taxonomy. When a feature structure has a type, it is called a *Typed Feature Structure*.

Following is an example of a nested TFS. The type of the main TFS is *book* and the type of the inner TFS *AUTHOR* is *writer*. 
2.8 Summary

Outlined above are the basic theories and frameworks that concern the semantic analysis method proposed in this dissertation. Readers are advised to refer to this chapter for more information and further reading when encountering any unfamiliar terminology. Due to time constraints, many important aspects are left out intentionally. For example, the formalism of sign-based construction grammar, as interesting as it is, is related to morphosyntax rather than integrated semantic analysis, which is the main focus of this thesis. In addition, other linguistic forms and aspects such as time, prosody, sound, gestures, et cetera, are not covered here. Those topics are interesting for further research. I believe that they are useful and have the potential to contribute a lot more to linguistic analysis. They are excluded only due to the limits of available time, scope, and resources for this research.
Chapter 3

Research Approach

This chapter discusses the motivation and direction for an alternative approach to the general computational semantic analysis method that is described in Section 2.4. Given a sentence, a tokenizer is used to split it into list of tokens. Next, the tokens are fed into a syntactic parser to obtain syntactic trees and typed dependencies. Finally, the token-based semantics is derived. If human languages were as precise as artificial formal languages, this approach would be a fine option. In reality, human languages are full of ambiguities and not all problems can be solved within each step described. For example, in (26) the verb is ambiguous. It can be the past tense of the verb see or the present tense of the verb saw as in The lumberjacks saw the log. This is lexical ambiguity.

The attachment of the prepositional phrase with a telescope is another ambiguity. If the phrase is combined with the verb saw, it means the telescope is the instrument used to see the girl. It also can be attached to the noun phrase a girl to form the reading “I saw the girl who held a telescope”. This is another type of ambiguity called structural ambiguity.

It is not trivial to resolve these ambiguities. For instance, background knowledge of the
relation between the verb *saw* and its arguments may be required in order to select the correct reading. One may derive from common sense knowledge that people do not saw with a telescope but with a saw, and concludes that the sense for the verb here should be *to see*, not *cutting*. Similarly, the telescope can be interpreted as the instrument to see the girl, not being held by the girl. In general, human relies on background knowledge for semantic disambiguation.

Errors in one stage also propagate to others. When the Stanford parser is used to process the following sentences (36): *did you sleep?* and (37): *does it snow?*, the errors in POS tagging stage affect the accuracy of the dependency parsing stage.

(36)  *Did you sleep ?*  

    Did/VBD you/PRP sleep/VB ?/.  

Typed dependencies

    aux(sleep-3, Did-1)  
    nsubj(sleep-3, you-2)  
    root(ROOT-0, sleep-3)

(37)  *Does it snow ?*  

    Does/NNP it/PRP snow/VB ?/.  

Typed dependency

    dep(snow-3, Does-1)  
    nsubj(snow-3, it-2)  
    root(ROOT-0, snow-3)
In (36), as the auxiliary verb \textit{did} was tagged correctly with “VBD”, the correct typed dependency list was given. In contrast, \textit{does} in (37) was recognised as the plural form of the noun \\
\textit{doe}. As a result we have an error in the typed dependencies.

\section{Integrated Semantics}

As demonstrated above, the ambiguities in one step may need access to information in the subsequent steps in order to get resolved. The modularised linguistic analysis in which every piece of information needs to be fully specified within the processing step is the root cause of the problem. After each processing step, the errors will be accumulated and they will affect the final analyses.

Let us reconsider the error in (36) and (37). The wrong structure derived in (37) is due to the ambiguities of the word \textit{does}. The word was annotated as a noun due to the requirement to provide a POS tag for the word and given the limited information it has access to (the sentence string)

It is similar in (30). Assuming that we have chosen the interpretation in which the \textit{with a telescope} phrase attached to the verb saw, it is still very difficult to determine which sense of the verb \textit{saw} should be used, based on the structure of the sentence. A grammar will tell us that both are possible, and we need a model to select the most plausible interpretation.

The modularised approach is a result of simplifying the problem to make it feasible to engineer. For instance, Word-Sense Disambiguation narrows down meanings to token-based concepts, and ignores structural semantics. Even when phrasal structures are used, they are syntactic relations and merely employed as additional input features rather than as actual components of semantics. Later chapters will investigate this assumption in more detail. This research is motivated by a personal desire to provide a more sophisticated
computational semantic analysis with conceptual, structural, and contextual analyses.

In (37), if we do not have to determine the POS tag for the word “does” during the
POS tagging process, but wait until the structure of the sentence has been examined, we
will see that “does” as the plural of the noun “doe” cannot be used to form a complete
sentence. We can even go further to claim that if we consider the construction grammar
approach as discussed in Section 2.1.1, we do not even have to solve the ambiguity in this
case. This is because there is no possible grammatical reading in which does is a noun.
The fact that parse ranking can be improved by using sense information and vice-versa was
previously investigated in Yin et al. (2014).

In (26), once we have access to the structure of the sentence and consider the knowledge
that people see but do not cut things with telescopes, we can determine the correct sense
of the word. Again, we are using information from different sources (common knowledge
and context) to perform the disambiguation. The knowledge of morphosyntax alone does
not help in this case.

To summarise, a core issue in automatic analyses of languages is the error accumulation
in independent modularised components. To overcome this, an integrated semantic model is
developed using construction grammar and Wordnet to constrain semantic interpretations
more effectively.

3.2 Integrated Semantic Framework

In this section, I discuss how the proposed framework can be implemented computationally.
Sign-based construction grammar approach has been chosen because, to the best of my
knowledge, this is the only formalism that has a broad coverage grammar of English (ERG
- Section 2.6.4). As introduced in Section 2.1.2, sign-based construction grammar is a
monostratal grammar. This means that semantic and syntactic analyses are performed in the same process (i.e. not a pipeline of autonomous syntactic and semantic components).

When given the text string in (26), the parsing process starts with recognising forms, for example “I”, “saw”, “a”, “girl”, “with”, “a”, “telescope”. Next, the grammar locates all the lexical items in the lexicon which have an orthography that matches the text tokens above (after the inflectional processing). For example, several lexical items that have the surface forms matching the string “saw” will be selected, named “see-past-1”, “saw-1-present”, “saw-2-noun”, et cetera.¹

All of the activated lexical items (atomic constructions) are then used to construct more complex constructions. For example, the construction of the verb “see-past-1” invokes a construction template which requires a SUBJ sign, an OBJ sign, and an optional PP sign. A unification engine is then used to build a directed acyclic graph, which consists of a bag of constructions and the link to their constituents.

¹The labels of lexical entries can be arbitrary unique names given by the linguists for organising linguistic resource purpose only. They are not the POS used for phrase structure rules. The name used in the examples in this research is created by me so that the readers can easily follow my explanations. In the actual description of the grammar, the names are often longer and more difficult to read because linguists pack more information into the labels. This actually has practical reasons as it is easier to know what has been documented about certain signs just by looking at the label.
At the end of the unification phase, the semantic analysis contains both lexical semantics (i.e. conceptual predicates) and structural semantics (e.g. who does what to whom). A simplified visualisation\(^2\) as a dependency graph is shown in (39).  

\[(39)\]

\[
\begin{align*}
\text{I} & \quad \text{saw} \quad \text{a} \quad \text{girl} \quad \text{with} \quad \text{a} \quad \text{telescope} \\
\text{pron}-\text{1st-sg} & \quad \text{pst-sg} \quad \text{quan} \quad \text{noun-1} \quad \text{prep} \quad \text{quan} \quad \text{noun-1}
\end{align*}
\]

But conceptual predicates do not provide enough semantic information. Wordnet is used as an ontology to enhance lexical semantics (Section 2.6.1). A typical sentence has multiple readings and the morphosyntactic analysis provides

\(^2\)The actual result in the computer memory for each construction is much more complex compared to what is being shown here. Even a simple atomic construction for the verb *see* can have several thousands of features (orthography, form, tense, arguments, et cetera). I may choose to use either a syntactic tree, dependency graph (as in (39)), or an attribute value matrix (as in (38)) in different contexts for the convenience of visualisation.
different parses for each reading. Every parse comes with its own semantic graph, which is composed of a bag of semantic predicates. Integrating the predicates with Wordnet synonym sets is not a trivial task. A part of my research is documenting these relations to form the knowledge base for the computational model. Disambiguation techniques are still needed in order to select the correct parses and senses.

For the reading in (39), there are three concepts, an event and three relations between the concepts and the event (the see-er, the see-ee and the instrument). After correct senses have been selected, the final result is shown in (40) which contains both lexical and structural semantics.

(40)
Chapter 4

TTL - A Text Annotation Library

In this chapter, I introduce TTL, or texttaglib, a flexible text annotation format as well as a software library. This research integrates resources and analyses from many systems (multiple Wordnets, human annotators, ERG/ACE, NLTK, TSDB++, et cetera). Each resource and tool utilises a different format. They are rarely compatible with others. Moreover, there are other annotation formats currently available, such as the NLP Annotation Format (NAF)\(^1\) or Unstructured Information Management Architecture (UIMA)\(^2\). It begs the question of why yet another formalism is introduced, especially when we have (too) many other annotation formats available. Most of the current annotation formats are designed with strict formalism for pipeline systems. They are designed to be a foundation for other systems to be built upon. In practice, most of the resources and systems (for examples the ones that are used in this research) are not built on top of these formats. Some systems provide shallow annotations, such as the NTU-MC. Each utterance is split into tokens and Wordnet’s senses are used to annotated those tokens. Babelfy API outputs a list of concepts (Wordnet synsets, Babelnet senses, et cetera) that are linked to tokens.

\(^1\)https://github.com/newsreader/NAF
\(^2\)https://uima.apache.org/

69
IDs and character chunks but no token list is provided (tokenization step is omitted or hidden from end users). Semcor is annotated with a customised XML-based format, which is modified from standard XML notation. The Wordnet Glosstag corpus is released in XML with a schema designed for providing complex token-sense mapping. In fact, among all of the software listed in this dissertation, only the Integrated Semantic Framework itself generates NAF compatible format. TTL does not aim to be a fundamental format, but instead is a transparent format which helps to overcome the challenge of integrating different systems without changing the underlying mechanism of each system. For example, TTL receives the input from Stanford part-of-speech tagger and generates input for the UKB Word-Sense Disambiguation system. A human annotator generates a gold profile using IMI system (NTU-MC) via a web interface. The output of IMI and UKB are both then redirected to TTL for benchmarking purposes. All systems are agnostic of each other and continue to work as normal, using their original formats.

There are three main components in the TTL eco system. The first one is the TTL specification or formalism, which is kept minimal intentionally. The second one is a Python software package which helps to process data flows, generate inputs, and convert outputs of different systems. The third one is a simplified text annotation format (TTLIG) that supports linguist annotators who work with limited tools (e.g. a bare minimum text editor and/or no Internet connection).

The following is a brief list of supporting features:

1. Stores large number of sentences
2. Flexible sentence bracketing (such as storing token information), even no bracketing at all
3. Tag-every-where system
   (a) Document-level tagging (i.e. metadata)
(b) Sentence-level tagging
(c) Token-level tagging
(d) Conceptual-level tagging (including token-concept mappings)

4. Provides different storage formats

(a) TSV: TTL data is stored in multiple TSV/CSV (tab-separated values) files, a text-based format, which can be edited in most simple text editors or spreadsheet programs;
(b) JSON: a document can be stored in a single JSON file which is flexible and widely supported by many programming languages. JSON is a simple data interchange format and it is easier to manipulate when compared to XML;
(c) SQLite: Indexed structured data - SQLite is a widely supported database format that supports both quick and powerful queries;
(d) TTLIG (or TIG in short): A simplified interlinear dialect of TTL. This is the easiest format among all currently supported formats. It allows users to directly edit the data using a text editor. It was developed specifically for field linguists to collect and organize their data from field trips. The format supports customisable glyphs using UTF-8 fonts, time reference to multimedia data, and more.

4.1 TTL Data Structure

TTL organises textual data into collections. Each collection may contain multiple corpora. Each corpus may contain multiple documents. Each document may contain multiple sentences. And each sentence may be bracketed into multiple constituents, also referred to as tokens. Bracketing is implemented to be theory agnostic, which means there is no strict
definition of what documents, sentences, or tokens should be. All levels can be bracketed according to research interests, and can be annotated without any imposed structures. This is the crucial point in designing a single framework that handles different analyses from multiple sources. For instance, a token in TTL is not necessary a word. A token can be any meaningful unit according to the data collector. The standard information contained in tokens includes `cfrom`, `cto`, `text`, `lemma`, `POS`, and `comment`. There features are provided for convenience as they are used quite often in NLP systems. However, they are all non-mandatory except `<cfrom:cto>`, which is required to link back to the source sentence. When there are more features to record (morphemes, phonemes, tenses, mood, person, number, gender, et cetera) a tag system can be used for extended features.

Following is an example of how different tokenization strategies can differ. In the sentence *I drink drip coffee.*, we can adopt a maximal bracketing strategy and produce five tokens “I”, “drink”, “drip”, “coffee”, “.” (punctuation), or we can split them into four tokens “I”, “drink”, “drip coffee”, “.”. The sentence can also be split into two tokens: “I” and “drink drip coffee.”. There can also be no bracketing at all, as tokenization is not required in some linguistic analyses.

After the raw text has been collected and organised into collections, corpora, documents, sentences, and tokens (optional), they can be annotated using tags. Tags can be freely created for different kinds of annotation. Each tag contains a label, a source, and a type. A tag can be attached to most TTL objects, from corpus, documents, to sentences, or tokens. Consider the example below:

(41)  

\[ I \quad \text{drink} \quad \text{drip-coffee} \quad . \]

PRON::1sg  WN::01170052v  WN::07919894n  PUNC::.

I drink drip coffee.
A PRON tag was created to annotate the first token with the label 1sg. Similarly a PUNC tag was also created to annotate the full stop punctuation. Two WN (i.e. Wordnet) tags are used to annotate the second and the third token with Wordnet senses.

To annotate semantic information, TTL concepts can be created. Concepts are special tags which can be used to annotate more than one token. They are useful for annotating multiple-word expressions when adopting maximal bracketing strategy (very popular in NLP systems). For example, if the sentence is tokenized into five tokens instead of four, we can do the following instead:

(42) \( I \) \( drink \) \( drip \) \( coffee \).  
PRON::1sg WN::01170052v \( C_1 \) \( C_1 \) PUNC::.

I drink drip coffee.

with \( C_1 = WN :: 07919894n \rightarrow token_2(drip), token_3(coffee) \).

As mentioned above, tokens are not required for annotating text and a tag can also be attached to a sentence (sentence-level tags). For example:

\( I \) \( drink \) \( drip \) \( coffee \). \( \rightarrow \) \( tag_1, tag_2 \)

with \( tag_1 = FUNC<0:1>\)Theme and \( tag_2 = FUNC<2:19>\):Rheme. The final result can be represented as:

(43) \( I \) drink drip coffee.
FUNCTION<0:1>Theme FUNCTION<2:19> Rheme

This is different from splitting the sentence into two tokens and annotating each one with a tag. In (43), \( tag_1 \) is annotated directly onto a chunk of the original sentence (the first character) and \( tag_2 \) covers the third character to the nineteenth character. Characters in a sentence are indexed using zero-based numbering (i.e. the first character is zero). A chunk
is identified by an ordered pair \(<\text{cfrom:cto}>\) with \text{cfrom} being the index of the starting character and \text{cto} being the index of the stopping character (non-inclusive). In example (43), \text{char}_{19} \text{ is the punctuation ("."}) and it is not included in \text{tag}_2.

The tag types are not predefined or fixed so new tags can be created when needed (although Leipzig and Penn treebank POS tags are provided as suggestions). TTL can be used to store annotations for different levels of analysis such as morphology, syntax, semantics, pragmatics, et cetera. For example, the stems \text{speed}, \text{meter}, and the interfix \text{o-} in the token "speed-o-meter" can be annotated by creating three tags \text{stem<0:5>}, \text{interfix<5:8>} and \text{stem<8:13>}.

TTL data can be stored in different formats such as TSV (tab separated values), JSON, or SQLite. TSV is a text-based table-like human-readable format. It makes it easier to browse through items and have a quick look at the data structure. JSON is widely supported in different development environments (Python, Java, Emacs Lisp, et cetera) and is thus good for computational purposes. SQLite is a database format where it is easy to manage structured data, making it convenient for different task such as corpus linguistic analysis. With SQLite, for example, one can perform a query to list all sentences that contain the token "does" with the lemmatized form "doe" (i.e. female of deer) instead of "do" – which is a common lemmatization mistake.

4.2 TTLIG

\text{TTLIG}, or \text{TIG} for short is TTL interlinear gloss, a special simplified implementation of TTL that enables field linguists to efficiently document linguistic data on field trips. It is less powerful when compared to the standard TTL, but it can be edited very quickly. Each

\footnote{It is a common practice to name computer file extensions using three characters for backward compatible purposes.}
TIG document is stored in a single text file and can be created using any text editor. By default, it uses UTF-8 encoding, which makes it capable of handling most languages and scripts. Following are some basic usages of TIG files:

- Human-friendly, can be read as normal text files using most operating systems
- Customisable fonts, useful for documenting languages with new scripts.
- Supports different levels of documentations with customisable fields (transliteration, translation, morphology, et cetera)
- Tools to validate data consistency and conversion to other supported formats (HTML, LaTeX, et cetera)

4.2.1 TTLIG Basic Structure

A TIG file contains a file header (to tell TIG readers that this file should be read in TIG mode), optional metadata, and sentences that are blank line separated. Following is an example of a tig file.

```
# TTLIG
Language: English
Language code: eng
Lines: __manual__
Author: Le Tuan Anh
Date: July 16 2018
Title: An example TIG file for my PhD thesis

ident: s1
```
A valid TIG file only requires the first line of the header (# TTLIG). All additional header fields added after that are optional (with colon separated key-value pairs). The header section ends with a blank line. When the header section is complete, sentences and their corresponding annotations can be added.

After the header, sentences can be added with multiple lines of annotation. A sentence block must end with a blank line. Any line starting with a hash symbol (#) is a comment line and will be ignored when processed. Comment lines may appear anywhere in the document, except for the first line, which is reserved for the header.

Although TIG files can be edited with any text editor, a highlighter syntax profile was created for Notepad++ on Windows operating system to enable syntax highlighting. TIG is smart enough to detect special characters such as “ or ” when importing tokens. This is useful for processing data from certain sources (e.g. Stanford POS tagger, Semcor, Glosstag, et cetera). Spaces and other filter characters can be skipped (auto) or can be added explicitly using one chosen escape character (defaulted to ). Flexible spacing makes reading and typing TIG files easier.
4.3 Performance

Even when the TSV format is used, scanning through data happens very quickly. A collection of Indonesian texts have been used for performance benchmarking. The raw dataset is 3.2 GB and contains 15,530,000 sentences. For testing purposes, the sentences were tokenized using NLTK’s word tokenizer. Each sentence contained two sentence-level tags for its source (URL) and a timestamp (to indicate when it was created). After tokenization, the dataset contained 299,433,218 tokens. Together, the final dataset (15.5 million sentences, 31 million sentence-level tags, and 300 million tokens) was stored in TSV format, taking around 10 GB of space. A complete scan through this entire dataset to build a lexicon (how many times each token appears in the corpus and in which sentences) was performed on a laptop with Intel® Core™ i7-7600U CPU (2.80GHz × 4) and 32 GB of RAM. The task was completed in 878 seconds (14.6 minutes) and generated a lexicon-sentence mapping with 2,633,305 token classes. These numbers refer to an exhaustive scanning using a TSV text file. If an SQLite database was used, this task would have been performed much faster.

<table>
<thead>
<tr>
<th>Format</th>
<th>Sent Tokens</th>
<th>Tags</th>
<th>Classes</th>
<th>Scanning Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSV</td>
<td>15.53M</td>
<td>299.43M</td>
<td>31.06M</td>
<td>2.63M</td>
</tr>
</tbody>
</table>

Table 4.1: TSV scanning

In addition, a TSV-based document can be read completely into memory for quick and direct manipulation. As shown in Table 4.2, TTL documents in TSV format can be read into memory with the speed of about 10 MB per second. 4

<table>
<thead>
<tr>
<th>Filename</th>
<th>Size</th>
<th>Reading time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ind_wikipedia_2016.16.30K</td>
<td>5.0 MB</td>
<td>1.67 secs</td>
</tr>
<tr>
<td>ind_newscrawl_2011.5.3M</td>
<td>1.9 GB</td>
<td>176.18 secs</td>
</tr>
</tbody>
</table>

Table 4.2: TSV reading

4This feature is subject to the amount of RAM available. In general, scanning through memory is more efficient as sentence data can be freed after processed.
4.4 Applications

TTL was designed to support both computational and non-computational linguists. One does not have to commit to a certain formalism and/or specific computational tool sets to make it useful. Here are some typical cases of using TTL:

- Language documentation with access to:
  - Framework-agnostic documentation tools
  - Original language scripts using customizable fonts
  - Morphological annotation
  - Multimedia data annotation
  - Functional annotation
  - Flexible bracketing

- Computational Linguistic related tasks:
  - Universal WSD evaluator
  - Corpus management
  - Data format converter

4.4.1 Linguistic Data Interchange Format

TTL makes it possible for the Integrated Semantic Framework to work with different projects and technologies from different research groups (see Figure 4.1), who often expect different formats and requirements on the inputs and outputs. It supports a wide diversity of inputs, from TTLIG for linguists to document linguistic data quickly, to TTL-SQLite for performing fast computational processing tasks. TTL-based data are flexible and can be transformed using available API and tools to serve inputs to different systems. Finally, the
output data from the systems can be unified using a single annotation schema for various tasks (evaluation, indexing and searching, analyses comparison, et cetera).

All datasets produced from this research are published in TTL format, with MRS information from the ISF, sense annotations from NTU-MC, gold DMRS structures from the TSDB++ profile of ERG gold suites, in a single representation.
Chapter 5

Constructional Semantics

In this research, I attempted to go beyond simple predicate-based semantics. With lexical semantics, we go from a concept per word (or words) to the possibility of one meaning per predicate. Since it is possible for one lexical unit to introduce several predicates, we may have several semantic units. From here, the idea of predicate-based semantics is developed into construction-based semantics, or the recognition of semantic units at a construction level. There are cases in which the semantic units are attached into a single non-compositional structure. For example, in the utterance *hot dog*, I would argue that it is more likely to refer to the concept of *SAUSAGE*, rather than to a canine-dog which is hot.

From a syntactic point of view, it may appear that *hot dog* is a noun phrase, with the noun “*dog*” being modified by the adjective “*hot*”. This non-compositional meaning is demonstrated further in idioms or set phrases. A classic example is the phrase *kick the bucket* which means *TO DIE*. From the structure, we may analyse the sentence as shown below:
But it is not by combining the meaning of the verb “to kick” with the meaning of the definite noun phrase “the bucket” (i.e. \( \text{kick}(e, x) \land \text{bucket}(x) \)) that we arrive at the meaning \textit{TO DIE}. The wording of the phrase in a particular order and structure constrains the meaning. Thus, the sign in this case should be closer to the form:

\[
\begin{array}{c}
\text{syn} \left[ \begin{array}{c}
\text{kick} \\
\text{OBJ} \\
\text{bucket}
\end{array} \right] \\
\text{sem} \left[ \begin{array}{c}
\text{die} \\
\text{ARG0} \\
e
\end{array} \right]
\end{array}
\]

The actual sign will be much more complex, but the important point is that, in the final semantics, there is no meaning of the smaller constituents (i.e. “kick”, “the”, or “bucket”). The whole structure is used as a single sign, which introduces a single semantic unit, the predicate \textit{die}(e).

In previous research by Bond, Sameha, and Flickinger (2013), the authors noted the importance of using semantic structures to recognise constructions instead of following syntactic structures. They refined this idea in Bond, Ho, and Flickinger (2015), where they successfully identify the semantic and syntactic structures of many idioms in English\(^1\). Although this result is very interesting and suitable for improving constructional semantics, I have not been able to exploit this data for generating construction rules for the ISF. The main difficulty is the lack of ERG semantic templates and senses for paraphrased templates.

\(^1\)I have also found the released dataset for this research which is available at \url{http://compling.hss.ntu.edu.sg idioms/possessed/}
This has to be done manually and takes a lot of time. As such, I will leave it for my future work when I make an extensive survey of constructions in English.

Ideally, these constructions should be implemented at the grammatical level to achieve the best performance speed. However, customizing a grammar into the size of the English Resource Grammar (ERG) would be extremely time-consuming, and developing a new grammar from scratch could take years to reach any real-world coverage. As a result of this, and considering all the limits imposed by time and availability of resources, I propose a solution to determine the original constructions based of the xMRS structures that the ERG provides. This solution utilizes limited syntactic information, such as tokens’ \(<\text{cfrom:cto}\)> , to produce a richer semantics analysis. This approach is similar to listening to a person who only understands literal, word-by-word meanings and trying to help them build richer interpretations of full utterances. Although this might not be the ideal approach, it can still help recognize possible constructions and produce better semantics – as pointed out in Section 7.4. Moreover, this solution works as a proof-of-concept to show that it is possible to implement constructional semantics in an actual working system. The subsequent sections will clarify how to recognize constructions based on semantic information and limited syntactic clues available within DMRS structures.

5.1 Building a Construction Database

Most of the constructions that are provided in this research are mined from Princeton Wordnet 3.0. A small number of senses are extracted from the Open Multilingual Wordnet project. First of all, all senses are retrieved in the form of \((\text{synsetID, lemma})\). Currently, there are 206,941 senses from Wordnet 3.0 and 2,923 senses from OMW.

Next, all lemmas are parsed using ACE and the ERG to test how the final constructions
Table 5.1: Princeton Wordnet 3.0 synsets

will look when analyzed with the standard model provided by the ERG. When parsing with the default settings of the ERG, there are many bad constructions produced because different modes of parsing can be used (root\_strict, root\_informal, root\_frag, root\_inffrag). Since Wordnet provides the part of speech for each of its lemmas, it is possible to exploit this information to discard the wrong parses sooner. For example, when considering the noun “reading”, any parses that contains \_read\_v\_1 with PROG+ can be discarded safely (i.e. since they refer to the progressive form of the verb to read). Since not all necessary root conditions for this process were included in the official release of the ERG, I have added these custom root to the ERG.

;; verb
root\_wn\_v := sign &
[ SYNSEM [ LOCAL [ CAT [ HEAD verb ]],
  NONLOC [ REL 0-dlist & [ LIST < > ] ] ]].

;; noun
root\_wn\_n := sign &
[ SYNSEM [ LOCAL [ CAT [ HEAD noun ]],
  NONLOC [ REL 0-dlist & [ LIST < > ] ] ]].

;; adjective
root_wn_adj := sign &
              [ SYNSEM [ LOCAL [ CAT [ HEAD adj ]],
                NONLOC [ REL 0-dlist & [ LIST < > ] ] ]].

;; adverb
root_wn_adv := sign &
              [ SYNSEM [ LOCAL [ CAT [ HEAD basic_adv ]],
                NONLOC [ REL 0-dlist & [ LIST < > ] ] ]].

By default, the lemmas are parsed according to their registered part of speech in Wordnet, and only the top 20 parses are considered. When these roots fail to produce any parses, the default root root_frag for parsing sentence fragments is used as a backup. After this process, the parses are paired with the source sense to form a construction. By using these custom roots, the bad parses from root_frag are reduced significantly. For example, when parsing the verb “study”, the ERG produces four fragments. The parse in Figure 5.1 takes “study” as a noun which is represented by the predicate _study_n_of that has an undefined quantifier udef_q and is a part of an unknown relation.

The second reading of Figure 5.2 recognises that “study” is a verb with an implied zero pronoun which was added automatically by grammar rules.

The third parse is similar to the first parse in Figure 5.1 but study is a noun phrase instead of a noun. The final parse in Figure 5.3 is a verb without any added zero pronoun, as in Figure 5.2.

When root_wn_v is used to parse “study” as a verb, only parses as in Figure 5.3 are produced. There are no unknown or zero pronoun predicates. When root_wn_n is used to parse the noun “study”, the parse in Figure 5.4 is produced.

Each parse will form a different sign. Figure 5.5 is an example of what an actual rule
Figure 5.1: Fragment parse #1 of “Study”

Figure 5.2: Fragment parse #2 of “Study”
Figure 5.3: Fragment parse #4 of “Study”

Figure 5.4: root_wn_n parse of the noun “Study”
<table>
<thead>
<tr>
<th>construction</th>
<th>kick the bucket</th>
</tr>
</thead>
<tbody>
<tr>
<td>lemma</td>
<td>kick the bucket</td>
</tr>
<tr>
<td>POS</td>
<td>v</td>
</tr>
<tr>
<td>synsetid</td>
<td>00358431-v</td>
</tr>
<tr>
<td>top pred</td>
<td>_kick_v_1</td>
</tr>
<tr>
<td>preds</td>
<td>⟨_kick_v_1, _the_q, _bucket_n_1⟩</td>
</tr>
<tr>
<td>target pred</td>
<td>_kick+the+bucket_v_1</td>
</tr>
</tbody>
</table>

```
struct

TOP   h
INDEX  e

RELS  ⟨
  [LBL h], [LBL h], [LBL h]
  [ARG0 e], [ARG0 x], [ARG0 x]
  [ARG1 x], [RSTR h], [ARG0 x]
  [ARG2 x], [BODY h], [ARG0 x]
⟩
```

Figure 5.5: A sample ISF construction

When the structure required by the construction shown in Figure 5.5 is found in a sentence, the structure information is used to create a new semantic unit based on the target predicate and information from the actual sentence.

The constructions and their related information are stored in an SQLite database, using the normal ISF/SQLite format and four extra tables to store rule information:

1. **lexunit**: this table stores information about the source lemmas, such as lemma, pos, synset ID, et cetera.

2. **ruleinfo**: links between a DMRS parse and a Wordnet sense are stored here, together with a flag for tracking the status of a rule for management purposes.
3. **rulepred**: all predicates within the DMRS structure of the sign are indexed and stored here for improving rule searching later on.

4. **predinfo**: a table that stores predicate related information such as lemmas, POS and senses and available DMRSes (not to be confused with the lemmas, POS and senses from Wordnet).

### 5.1.1 Rule Pruning

After the lemmas were parsed, there were 976,168 raw parses in the ACE output file. Unfortunately, not all of these constructions are useful for creating a construction. Many of them (i.e. more than 200,000) were disconnected structures or bad parses. To solve this, an automatic sorting of the rules was created to filter these bad parses out.

First, two corpora were created and named *lexrules* and *unknown*. They function in a similar way, with the exception that the constructions with unknown-word predicates were stored in a separate corpus (i.e. *unknown*). Currently, the rules from both of these corpora are being used, but ideally all unknown words should be handled properly by the ERG. The main use of the *unknown* corpus is to be used for further development of the ERG.

Each corpus contains five documents, *Nouns, Verbs, Adjectives, Adverbs*, and *Others*. These documents are used to store the parses according to the POS of the source lemmas for quicker browsing and searching of constructions. The document *Others* is used for any lemma with POS other than *n, r, v, a, s*, such as synsets marked with POS *x*, from the OMW. These are mainly pronouns, interjections, or conjunctions.

Next, the rules were filtered by the number of predicates they carry. Because some of the lemmas were parsed as sentence fragments, there may be extra grammatical predicates such as *unknown_rel* or *pron_rel* (for zero pronoun), or quantifier predicates added automatically
Figure 5.6: ISF Construction Database corpus design
by the ERG. These predicates must be taken away before the predicate counting process is performed. If there is only one predicate, it indicates that the lemma and predicate can be mapped directly and no transformation needs to be done. Thus, all of these rules are marked as single predicates and deactivated in the final database. The remaining lemmas were parsed as multiple predicate constructions. In order to filter the available parses, part of speech checking can be used. For the nouns, the part of speech of the top predicate should be either noun or a verb that is modified by a nominalization predicate. Similarly, for adjectives, only the constructions where the head is marked as an adjective or a verb with either \texttt{PERF+} or \texttt{TENSE:Past} are considered. It is easier for verbs and adverbs as they are recognized very well by the ERG. The rest of the constructions are deactivated. Table 5.2 shows the total number of constructions in each category. In total, 18,897 constructions could have a one-to-one mapping. These mappings are done directly by the ISF pred-sense mapping module without construction transformation. There are 29,904 lemmas with some mismatching between the parts of speech for the parses’ head and the Wordnet sense parts of speech, as well as 71,133 lemmas with unknown predicates in at least one of their parses. The rest are the number of lemmas that can be parsed with matching parts of speech.

<table>
<thead>
<tr>
<th>Category</th>
<th>count</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gold</td>
<td>18897</td>
<td>One-to-one mapping available</td>
</tr>
<tr>
<td>Compound</td>
<td>16644</td>
<td>Unsorted compounds</td>
</tr>
<tr>
<td>Mismatched</td>
<td>29904</td>
<td>Some parses are mismatched</td>
</tr>
<tr>
<td>Unknown</td>
<td>71133</td>
<td>With unknown predicates</td>
</tr>
<tr>
<td>Nouns</td>
<td>40533</td>
<td></td>
</tr>
<tr>
<td>Verbs</td>
<td>5188</td>
<td></td>
</tr>
<tr>
<td>Adjectives</td>
<td>13850</td>
<td></td>
</tr>
<tr>
<td>Adverbs</td>
<td>716</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>196865</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2: Construction database - Lemmas with constructions

As shown in Table 5.3, there are 196,865 active senses and 359,530 pairs of (sense, structure).
Since searching through constructions can be very slow, the predicate string and the $\text{carg}^2$ field of the top predicate and all other predicates inside each DMRS structure were also stored as extra attributes (shown in Figure 5.5). This extra information is indexed by the SQLite database so that they can be retrieved faster.

<table>
<thead>
<tr>
<th>POS</th>
<th>By lemmas</th>
<th>By rules</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No good parse</td>
<td>Can be parsed</td>
</tr>
<tr>
<td>a</td>
<td>4377</td>
<td>25857</td>
</tr>
<tr>
<td>n</td>
<td>5312</td>
<td>142916</td>
</tr>
<tr>
<td>r</td>
<td>1286</td>
<td>4363</td>
</tr>
<tr>
<td>v</td>
<td>1733</td>
<td>23539</td>
</tr>
<tr>
<td>x</td>
<td>191</td>
<td>190</td>
</tr>
<tr>
<td>total</td>
<td>12899</td>
<td>196865</td>
</tr>
</tbody>
</table>

Table 5.3: Construction Database - Processed Lemmas

The numbers shown in Table 5.4 attempt to show the relevance of storing the values of the feature $\text{CARG}$. There are 47,130 rules with the top predicate being $\text{named\_rel}$. If $\text{CARG}$ information was not stored, each $\text{named\_rel}$ would have to be compared to these constructions to find matchable rules. One interesting note about the ERG analyses is that there are several classes of constructions which have information stored in $\text{CARGs}$ instead of predicate’s lemmas, not just named entities. For example, the name “Holmes” with be parsed to $\text{named\_rel}$ with $\text{CARG}=$"Holmes", the word “summer” will be parsed to $\text{season\_rel}$ with $\text{CARG}=$"summer".

### 5.2 Transforming DMRS Structures

With the construction database built, it is now possible to recognize their appearance in ERG outputs. Any sentence with readings in xMRS forms can be processed using the

---

$^2$CARG stands for character strings, which hold the surface values of the word forms of the source text.
### Table 5.4: Construction Database - Rules by top pred count

<table>
<thead>
<tr>
<th>top pred</th>
<th># of rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>named</td>
<td>47130</td>
</tr>
<tr>
<td>subord</td>
<td>27059</td>
</tr>
<tr>
<td>appos</td>
<td>5990</td>
</tr>
<tr>
<td>nominalization</td>
<td>5159</td>
</tr>
<tr>
<td>_system_n_of</td>
<td>2314</td>
</tr>
<tr>
<td>_tree_n_of</td>
<td>2173</td>
</tr>
<tr>
<td>_order_v_1</td>
<td>2170</td>
</tr>
<tr>
<td>_unit_n_2</td>
<td>1830</td>
</tr>
<tr>
<td>_be_v_id</td>
<td>1612</td>
</tr>
<tr>
<td>_take_v_2</td>
<td>1557</td>
</tr>
<tr>
<td>_unit_n_of</td>
<td>1275</td>
</tr>
<tr>
<td>_and_c</td>
<td>1269</td>
</tr>
<tr>
<td>loc_nonsp</td>
<td>1261</td>
</tr>
<tr>
<td>_come_v_1</td>
<td>1244</td>
</tr>
<tr>
<td>_take_v_1</td>
<td>1151</td>
</tr>
<tr>
<td>_go_v_1</td>
<td>1097</td>
</tr>
<tr>
<td>_river_n_of</td>
<td>1087</td>
</tr>
<tr>
<td>implicit_conj</td>
<td>1069</td>
</tr>
<tr>
<td>_line_n_of</td>
<td>1020</td>
</tr>
<tr>
<td>card</td>
<td>994</td>
</tr>
</tbody>
</table>

The ERG parse result of each sentence parse can now be passed through this function. First, the parse must go through named compound simplification (Algorithm 5.2) to reduce the number of predicates to be matched in a later process. With this function, each named compound are collapsed into a single named predicate. A named compound is checked by looking at its arguments from both sides, namely ARG1 and ARG2 (Algorithm 5.1). If its arguments are either named compounds (recursive checking) or named predicates, then it will be marked as a named compound. With each detected named compound, a new named predicate is created to replace the whole structure. The new predicate will contain the `<cfrom:cto>` of the original compound predicate, as well as a new CARG value based on
the internal named predicates. The incoming links and other semantic values of the head are also transferred to the new named predicate. These changes are enough to preserve the DMRS graph because, except for their head, named compounds are an isolated subgraph.

Algorithm 5.1: Checking for named compound

```
input: Pred p
output: Boolean
begin
  if p.pred == 'compound_rel'
    if is_named_comound(p['ARG1']) or p['ARG1'].pred == 'named_rel'
      if is_named_compound(p['ARG2']) or p['ARG2'].pred == 'named_rel'
        return True
    return False
  end
end
```

Algorithm 5.2: Named compound simplification

```
input: DMRS source_dmrs
output: DMRS out_dmrs
begin
  out_dmrs ← source_dmrs
  compounds ← find_compound(source_dmrs)
  for compound in compounds
    if is_named(compound)
      new_pred ← Predicate('named_rel')
      new_pred.cfrom ← compound.cfrom
      new_pred.cto ← compound.cto
      new_pred.carg ← join(compound)
      out_dmrs.replace(compound, new_pred)
  end
end
```
Algorithm 5.3: DMRS Transformation

1. **input**: DMRS source_dmrs
2. **output**: DMRS out_dmrs
3. **begin**
4.  out_dmrs ← simplify_named_compound(source_dmrs)
5.  rules ← search(source_dmrs.preds)
6.  **for** rule in rules
7.     **for** node in out_dmrs
8.         sub ← subgraph(node, rule.structure)
9.         if sub exists
10.        transform(rule, out_dmrs)
11.        break
12. **return** out_dmrs
4. **end**

To exemplify this DMRS transformation, let us consider the sentence: *He is Sherlock Holmes*. The original DMRS semantic representation for this sentence would be as follows:

Upon inspection, the ISF can detect an isolated compound that contains only named_rel. It will simplify the compound into a single predicate denoting this same named entity. In other words, the original DMRS subgraph for the named entity compound as below:

Would be simplified into a subgraph with the form:

Substituting the simplified subgraph into the original DMRS representation, the final
DMRS representation for the sentence above would be as follows:

```
pron pronoun_q _be_v_id proper_q named(Sherlock Holmes)
```

Even though this named compound simplification process can be useful in named entity recognition, this feature is disabled in the current release. The reason for this is that Wordnet does not contain many named entities. Even if enabling this feature increases the accuracy of the final evaluation, in practice, this does not yield any real world benefits while using Wordnet. Anytime a name appears in a sentence, such as “Mary” or “John”, the final analysis will always be resolved to one of the existing synsets.

For MARY, a single concept is available:

1. 11161412-n: the mother of Jesus; Christians refer to her as the Virgin Mary; she is especially honored by Roman Catholics (Princeton Wordnet 3.0).

For JOHN, one of five available concepts would be used:

1. 04446276-n: a room or building equipped with one or more toilets
2. 06442239-n: the last of the four Gospels in the New Testament
3. 10779995-n: a prostitute’s customer
4. 11085924-n: (New Testament) disciple of Jesus; traditionally said to be the author of the 4th Gospel and three epistles and the book of Revelation
5. 11086279-n: youngest son of Henry II; King of England from 1199 to 1216; succeeded to the throne on the death of his brother Richard I; lost his French possessions; in 1215 John was compelled by the barons to sign the Magna Carta (1167-1216)

To exemplify what would happen with non-named entities, let us consider the recognition of the concept TEA CHEST, which uses the construction shown in Figure 5.7.
When parsing the sentence *This is a tea chest*, the output DMRS AVM will be as shown in Figure 5.8 and a dependency graph as shown in Figure 5.9.

The four elementary predicates shown in bold in Figure 5.8 are the same values as the `preds` in Figure 5.7. The transformation process (Algorithm 5.3) scans and matches each predicate in this DMRS with available constructions using the top predicate, the predicate list, and the construction structure. Ultimately, only the final matching step (graph matching) is needed, but the first two are useful in discarding the constructions that cannot be matched, thus reducing processing time.

Ideally, the transformation function mentioned in Algorithm 5.3 should be handled by more efficient machinery, such as the Transfer Formalism, which are general-purpose MRS rewriting rules (Oepen, 2008). This was also the method that enabled the Japanese-English
deep machine translation system (Bond et al., 2011). In a machine translation system, the source language text is parsed using the source grammar to produce MRS structures. These MRS structures are then transformed into new MRS structures of the target language using the transfer grammar (in other words semantics-to-semantics grammar). Finally, the target grammar generates target language text based on the newly created target MRS.
structures. This semantics-to-semantics transfer method is not only suitable for machine translation but also for the construction transformation task being investigated here. I attempted to use this method for construction transformation but did not succeed, due to lack of technical documentation and debugging facilities of the provided software package in Oepen (2008). In order to develop a complete implementation of the ISF, I had to implement my own version of the transformation function. In my future work, I hope that the semantics transfer machinery can be integrated into the ISF for better transformation performance.

Once the construction is matched, the transformation is complete and the whole matched subgraph is replaced with the new predicate \_tea+chest\_n\_1, with the semantic features being transferred from the old head (\_chest\_n\_1).

After transformation has taken place, the final semantic structure will appear as in Figure 5.10, and the dependency graph as in Figure 5.11.

5.3 Discussion

First, it is easy to notice that the semantics in the transformed structure (Figure 5.11) is simpler than the original one in Figure 5.9. However, it is even more important to note that the semantic analysis is also more accurate. In the original analysis, we have a compound semantics of CHEST that is modified by TEA. This can be interpreted as a chest that has some relation to tea (e.g. that contains tea or with tea-related decoration, et cetera). But TEA CHEST is an established concept in the English language, which means “04397168-n: chest for storing or transporting tea” (Princeton Wordnet 3.0) with an established design. Of course it is possible to call any chest that contains tea a tea chest, in which case the original interpretation will be used. The additional reading provided by the ISF offers an
optional reading which may lead to a better analysis. We can find more of these examples such as “dining table” (any table related to dining?) or “washing machine” (any machine that has something to do with washing?).

Second, it is not just a random combination of predicates that emerged from adjacent words. The matched construction has to be in the exact structure in order for the transformation to be processed. For example, the noun phrase *dish washer* in the phrase *the new dish washer* can have multiple interpretations. The first of these interpretations which is shown in Figure 5.12, can be matched because dish-washer is a known construction in the
ISF. However, the second interpretation shown in Figure 5.13 will not be matched because “dish” is modified by “new”, which makes the final structure (“new-dish washer”). As “new-dish washer” is not a known construction in the ISF, it will be left untouched.

Third, the transformation should not be considered similar to a surface textual matching. If we perform textual matching, we will have to perform additional steps to be able to match the inflected cases. The ISF does not have any problem processing sentences such as I have two tea chests. The transformed DMRS preserves the extra semantic information that were derived from the original semantic head, which is shown in Listing 5.4. Here, the semantics of TEA CHEST is recognized correctly with NUM being plural (p1).

Listing 5.4: DMRS of I have two tea chests.

dmrs {
    10000 [pron<0:1> x ind=+ num=sg pers=1 pt=std];
    10001 [pronoun_q<0:1> x ind=+ num=sg pers=1 pt=std];
    10002 [_have_v_1_rel<2:6> e mood=indicative perf=- prog=- sf=prop tense=pres];
If we then try to match all inflected forms of individual words within a phrase, we will run into another problem: over analysis. The DMRS structure of *guard dog* is restricted by two predicates, *guard_n_1* and *dog_n_1*. Although there are different phrases that contain inflected forms of “guard” and “dog”, such as *guarding dog*, *guarded dog*, or *guard dogs*, only *guard dogs* or *guard dogs* can produce the two predicates mentioned and thus trigger the transformation process. This is done automatically by comparing the semantic information and structure of each construction to the sentence being analyzed. The semantic features such as tense, number, or argument structure are considered in the matching. If this is done on the surface matching level, one of these would have to happen: under-matching (losing inflected cases), over-matching (bad inflected cases being used), or it would be necessary to use some other structure analysis/ranking techniques.

I make no claims that the current ISF constructional analysis is perfect. In fact, it is far from being perfect. The main reason is that most of the constructions are generated automatically from the outputs of Wordnet lemmas parsed by the ERG. Without building a corpus of phenomena and providing different strategies for building and applying construc-
tions, the current ISF system still allows noise from bad parses, even when multiple filters and manual tests have been implemented. But since the task of surveying and providing computational implementation for different constructions would be in itself another large scale project, I have decided to leave this out of the scope of this research, despite the fact that this is a very interesting problem with many useful applications.

The second limitation that constrains the performance of the construction analysis module is the graph matching algorithm. The adjacent requirement was added to make sure that the construction analyzer is able to produce analyses in reasonable time (a few minutes for long sentences). In addition, the current graph matching module is not powerful enough to perform detached or template-based (graph with holes) matching, thus leaving out many interesting structures that I would have been able to investigate. When these technologies are made available and integrated into the ISF, I believe that a large number of idiomatic constructions and/or sarcastic utterances would be able to be solved computationally, at least in a certain practical level. For example, in the construction to lose one’s mind, the referent that ONE points to and the agent (ARG1 of the predicate $\_\text{lose}_v_1$) has to be identical. Figure 5.14 shows what the rule for this would look like. Currently this is achievable if the whole graph is scanned through manually using code written for each individual construction. This is one of the main focuses of my plan for future research.
Figure 5.14: Construction for *to lose one’s mind*
Chapter 6

Concepts and Predicates

Looking at the sentence *a dog barked* using semantic structural analysis, we can find three predicates\(^1\) \(_a\_q\_rel\), \(_\text{dog}\_n\_1\_rel\), and \(_\text{bark}\_v\_1\_rel\), as shown in Figure 6.1. The AVM and the dependency diagram show the structural semantics of the sentence. There is an individual \(x_3\) that is a \(_\text{dog}\_n\_1\) and an event of barking \(e_2\) in which \(x_3\) is the agent.

From the perspective of lexical semantics analysis, there are two known concepts, “dog” and “bark”, which can be identified in the sentence. This is represented by using Wordnet senses: 02084071-n (dog) and 01047745-v (bark).

In this chapter, I discuss the relation between these two major analyses, and how to integrate them into a single representation.

In order to solve this integration problem, there are two major tasks that need to be done. The first task is figuring out how to integrate these two levels when the conceptual information is available (i.e. map the synset IDs to the predicates). The second task is figuring out how we can find this conceptual information when it is not available. This problem is similar to the token-based lexical semantic analysis as discussed in Chapter 7.2,

\(^1\)In general, the term **predicate** is used to denote MRS elementary predications in this dissertation.
but using predicates instead of tokens. This second task can be divided into two sub-tasks: how to find potential sense candidates for the available predicates, and how to rank the senses to choose the best sense for each predicate. The ranking algorithm is similar to token-based methods (Lesk, MFS), so it is not covered in this chapter.

To summarize, in order to integrate lexical semantics and structural semantics, there are two tasks to be solved:

**Sense-Pred Mapping:** find corresponding predicates to known concepts;

**Pred-Sense Mapping:** find potential senses for known predicates;

The former task is for combining different available analyses (i.e. from different syss-
tems/methods) and the latter is for providing an integrated analysis.

6.1 Sense-Pred Mapping

In this section, I discuss how to map known senses to predicates. Sense-Pred mapping is useful when we need to integrate different systems and analyses. In IMI (Bond, Morgado da Costa, and Lê, 2015), human annotators go through a text and take note of concepts that they find in the text. The output of this step is a list of sentences and associated concepts (Wordnet synsets). In ERG test suites, human annotators go through sentence parses and choose the best parse for each sentence (i.e. the best structure to represent the intended meanings). The gold dataset of the ISF (speckled - The Adventure of the Speckled Band) is created by integrating the two datasets mentioned above - gold concepts and gold structures.

Assuming that the sentences and sense information have both been normalised into TTL format, we can map the sense locations into the original sentences (i.e. from which character to which character). In a well-formed DMRS structure, we can also map each predicate to their respective location in the sentence surface form (i.e. from which character to which character). So the obvious approach is to exploit this information to map the senses to predicates (see Algorithm 6.1). First, I will assume that, for each sense in the gold sense annotation, there will be one and only one associated predicate in the gold treebank. For each provided sense, we will go through the remaining predicate list and compare the cfrom and cto.

If they are matched, we will pair this predicate to the current sense and take both of them out of the consideration list. The loop will continue and terminate when either the predicate list or the sense list is empty. The pseudo-code for this mapping can be found
Algorithm 6.1: Naive Sense-Pred Mapping

1. **input**: `Concept[] concepts, Pred[] preds`
2. **output**: `((pred_1, concept_1), (pred_2, concept_2), ...)`
3. begin
   4. `matched_preds ← set()`
   5. for `concept` in `concepts`
      6. for `pred` in `preds`
         7. if `match(concept, pred)`:
         8. `matched_preds.add((concept, pred))`
         9. `preds.remove(pred)`
         10. break
   11. return `matched_preds`
4. end

The above `match()` function checks if the a concept can be matched to a predicate. The following conditions are required:

1. The concept and the predicate have an identical pair of `(cfrom, cto)` values;

2. The concept is a multi-word expression and the predicate’s lemma is the same text as the surface form of the first word in the concept

3. The concept is a multi-word expression and its tokens’ text are the same as the predicate’s lemma (split by the character `+`). If the last part of the predicate’s lemma is a preposition, then the tokens’ text will be compared to both lists of the concept’s lemma (with and without the preposition)

In total, there are 6,152 senses available in the speckled data set. Using the greedy approach sketched above, 5,140 senses were mapped to available predicates. There were 1,012 senses that could not be mapped. These unmapped senses were grouped together by synset IDs and by lemmas for further investigation on why they could not be matched.
A first glance at this group of unmapped concepts revealed that a large number of them contained words that stand next to punctuation marks. When the ERG calculates the (cfrom, cto) pair, sometimes cfrom and cto are extended to include the punctuations as well. To overcome this, every time a predicate was considered, the surface string was also examined. If it contained punctuation marks on either side, the (cfrom, cto) pair used for comparison was adjusted and the surface string would have the punctuation removed. Just by doing this, the number of unmapped concepts was reduced to 353.

Another problem found concerned double annotated senses. For example, when annotating the word knock up, the annotator had annotated it with 02185373-v (knock) and at the same time created a new concept for knock up. As the ISF did not provide a sense for individual word forms in a construction, the one that was left out was deemed as an error when it is not. I made an assumption that if a group of tokens were tagged with multiple senses, the senses that covered the most tokens will be kept and the rest will be ignored. This is shown later, in Table 6.4, by the column SS- (Removing smaller senses). With this change, the number of unmapped predicates was reduced to 223.

The three strategies employed for selecting the predicates that can be mapped were: greedy (all predicates are considered), robust (all predicates except quantifiers _q_rel are considered) and strict (only real predicates\(^2\), named_rel, pron_rel, and card_rel are considered).

There were cases when multiple predicates were mapped onto the same location (cfrom, cto). For example, the token she will yield two predicates pron_rel and pron_q_rel. The synset 77000041-n should be linked to the first one (i.e. pron_rel). For efficiency purpose, the ISF does not match all possible predicates in a sentence, picking only the first available match. However, since one concept should only be linked into one predicate, the predicates

\(^2\)Real predicates are the ones contain an explicit surface word form.
must be sorted before the matching process: real predicates were considered before gram-
matical predicates\(^3\); grammar predicates that contain CARG will be considered next; and
quantifiers will be considered last.

The improved Sense-Pred Mapping algorithm, with the changes described above, can
be found below.

Algorithm 6.2: Sense-Pred Mapping final algorithms

\[
\text{Algorithm 6.2: Sense-Pred Mapping final algorithms}
\]

```
input: Concept[] concepts, Pred[] preds, mode
output: ((pred\(_1\), concept\(_1\)), (pred\(_2\), concept\(_2\)), ...)
begin
  filter(preds, mode)
  sort(preds)
  matched_preds ← set()
  for concept in concepts
    for pred in preds
      adjust_cfrom_cto(pred)
      if match(concept, pred):
        matched_preds.add((concept, pred))
        preds.remove(pred)
        break
  return matched_preds
end
```

6.1.1 Development on Semcor

Data Preparation

ISF uses Semcor as a development dataset. The Semcor corpus\(^4\) is a subset of the Brown
corpus\(^5\) which has been annotated with Wordnet senses. Even though there is no gold
DMRS profile of the Semcor, thus making it unsuitable for benchmarking our deep parsing

---

\(^3\)Grammar predicates are the ones introduced by a grammar for internal consistency.
\(^4\)http://web.eecs.umich.edu/~mihalcea/downloads.html
\(^5\)http://www.nltk.org/nltk_data/
methods, it can still be useful for development purposes.

The first problem of using the Semcor dataset is that there is no raw text version available for public access. The dataset that is available freely (including the one that is distributed with NLTK) only contains tokenized text. This is different from the required input of ISF which is raw text. Therefore my first task was to reconstruct the original Semcor’s text.

Semcor original data is provided in XML format. However, Semcor XML format is not well-formed (not quoted properly) and thus it cannot be read using Python standard XML libraries such as ElementTree or lxml. BeautifulSoup, a markup manipulation library for Python, was used to fix the malformed XML and convert data into valid XML files. Next, tokens were collected and joined together to form sentences – unfortunately, this task was not trivial. Each sentence in Semcor is given as an ordered list of tokens which are marked either as \texttt{wf} (lemmata) or \texttt{punc} (punctuations). An example of a sentence extracted from Semcor can be found in (44), below.

(44)

\begin{verbatim}
<wf cmd=ignore pos=DT>The</wf>
<wf cmd=done rdf=group pos=NNP lemma=group wnsn=1 lexsn=1:03:00:: pn=group>
  City_Purchasing_Department</wf>
<punc>,</punc>
<wf cmd=ignore pos=DT>the</wf>
<wf cmd=done pos=NN lemma=jury wnsn=1 lexsn=1:14:00::>jury</wf>
<wf cmd=done pos=VB lemma=say wnsn=2 lexsn=2:32:01::>said</wf>
<punc>,</punc>
<punc>`</punc>
<wf cmd=done pos=VB lemma=be wnsn=1 lexsn=2:42:03::>is</wf>
\end{verbatim}
<wf cmd=done pos=JJ lemma=lacking wnsn=1 lexsn=5:00:00:inadequate:00>lacking</wf>
<wf cmd=ignore pos=IN>in</wf>
<wf cmd=done pos=JJ lemma=experienced wnsn=1 lexsn=3:00:00::>experienced</wf>
<wf cmd=done pos=JJ lemma=clerical wnsn=1 lexsn=3:01:01::>clerical</wf>
<wf cmd=done pos=NN lemma=personnel wnsn=1 lexsn=1:14:00::>personnel</wf>
<wf cmd=ignore pos=IN>as</wf>
<wf cmd=ignore pos=DT>a</wf>
<wf cmd=done pos=NN lemma=result wnsn=1 lexsn=1:19:00::>result</wf>
<wf cmd=ignore pos=IN>of</wf>
<wf cmd=done pos=NN lemma=city wnsn=2 lexsn=1:15:01::>city</wf>
<wf cmd=done pos=NN lemma=personnel wnsn=1 lexsn=1:14:00::>personnel</wf>
<wf cmd=done pos=NN lemma=policy wnsn=1 lexsn=1:09:00::>policies</wf>
<punc>'</punc>
<punc>'</punc>
<punc>.</punc>

If we naively try to rebuild the sentence by joining all tokens using a space in between, we will have the sentence shown in (45).

(45) The City_Purchasing_Department, the jury said, “is lacking in experienced clerical personnel as a result of city personnel policies”.

There are other issues that go beyond incorrect spacing such as the word can't being tokenized into can and n't, wrong characters for double quotes, or underscores that are added for multiword expressions. These errors were fixed using manual rules.
Parsing with ERG

Out of 37,176 sentences, there were only 23,526 sentences with at least one reading from ERG. This effectively puts the coverage of ERG at 63.3%, as shown in Table 6.1.

<table>
<thead>
<tr>
<th>Information</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parsed Sentence</td>
<td>23,526</td>
</tr>
<tr>
<td>Total Sentence</td>
<td>37,176</td>
</tr>
<tr>
<td>Coverage</td>
<td>63.3%</td>
</tr>
</tbody>
</table>

Table 6.1: Semcor parsed using ERG.

Evaluation on Semcor

After Semcor was processed using ISF there were 175,719 occurrences and 12,862 types of predicates found. Using exact lemma matching and re-mapped part-of-speech \((r, s, a \rightarrow a)\), it was possible to link 95.47% of the predicates to Wordnet synsets (96.12% if we count by predicate types). With extended mapping rules, the coverage increased to more than 99% as shown in Table 6.2. Out of 175,719 occurrences, there were only 1,092 predicates left unmapped.

<table>
<thead>
<tr>
<th>Information</th>
<th>Occurrences</th>
<th>Unique Predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Count</td>
<td>Percentage</td>
</tr>
<tr>
<td>Exact Match</td>
<td>167,766</td>
<td>95.47%</td>
</tr>
<tr>
<td>Extended Match</td>
<td>6,861</td>
<td>3.9%</td>
</tr>
<tr>
<td>Match (total)</td>
<td>174,627</td>
<td>99.37%</td>
</tr>
<tr>
<td>Not Found</td>
<td>1,092</td>
<td>0.63%</td>
</tr>
<tr>
<td>Total</td>
<td>175,719</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 6.2: Semcor analysis results

Table 6.3 shows the top 9 unmapped predicates from the obtained results. The majority, 369 of the unmapped predicates, are named_n_rel, which is a special predicate label to
annotate named entities in ERG, and thus do not need to be linked. The same thing goes for the predicate colon_v_id_rel. Prefixes such as (re- (124), un- (104), mis- (24), pre- (22), post- (7)) and the other predicates (else_a_1_rel or at+first_a_1_rel) do not exist in Wordnet. In total, 787 predicates are not currently linkable. In a future phase of this project, I will examine these missing senses in detail, and add them to Wordnet.

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>named_n_rel</td>
<td>369</td>
</tr>
<tr>
<td>re-_a_again_rel</td>
<td>124</td>
</tr>
<tr>
<td>un-_a_rvrs_rel</td>
<td>104</td>
</tr>
<tr>
<td>else_a_1_rel</td>
<td>76</td>
</tr>
<tr>
<td>colon_v_id_rel</td>
<td>43</td>
</tr>
<tr>
<td>mis-_a_error_rel</td>
<td>24</td>
</tr>
<tr>
<td>pre-_a_ante_rel</td>
<td>22</td>
</tr>
<tr>
<td>at+first_a_1_rel</td>
<td>18</td>
</tr>
<tr>
<td>post-_a_1_rel</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 6.3: Not Found predicates.

6.1.2 Evaluation

To determine how well the Sense-Pred Mapping algorithm performs, two metrics were adopted. For evaluation, we counted how many senses could be mapped to DMRS predicates. For validation, the senses were exported back into TTL format and compared against the original gold profile to determine how accurate their matchings were. The gold dataset used for evaluation was the annotated Sherlock Holmes story The Adventure of the Speckled Band which is released as a part of the NTU-MC. The document contains 599 sentences and 6,146 annotated concepts.
Table 6.4: Sense-Pred Mapping Evaluation on 6146 senses

<table>
<thead>
<tr>
<th>ISF</th>
<th>Strategy</th>
<th>SS-</th>
<th>Fix cfrom:cto</th>
<th>Matched</th>
<th>Not matched</th>
<th>Ignored</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>greedy</td>
<td></td>
<td></td>
<td>5415</td>
<td>731</td>
<td>0</td>
</tr>
<tr>
<td>No</td>
<td>greedy</td>
<td>X</td>
<td></td>
<td>5876</td>
<td>270</td>
<td>0</td>
</tr>
<tr>
<td>No</td>
<td>greedy</td>
<td>X</td>
<td>X</td>
<td>5334</td>
<td>595</td>
<td>217</td>
</tr>
<tr>
<td>No</td>
<td>greedy</td>
<td>X</td>
<td>X</td>
<td>5791</td>
<td>138</td>
<td>217</td>
</tr>
<tr>
<td>No</td>
<td>robust</td>
<td>X</td>
<td>X</td>
<td>5785</td>
<td>144</td>
<td>217</td>
</tr>
<tr>
<td>No</td>
<td>strict</td>
<td>X</td>
<td>X</td>
<td>5571</td>
<td>358</td>
<td>217</td>
</tr>
<tr>
<td>Yes</td>
<td>greedy</td>
<td></td>
<td></td>
<td>5342</td>
<td>804</td>
<td>0</td>
</tr>
<tr>
<td>Yes</td>
<td>greedy</td>
<td>X</td>
<td></td>
<td>5815</td>
<td>331</td>
<td>0</td>
</tr>
<tr>
<td>Yes</td>
<td>greedy</td>
<td>X</td>
<td>X</td>
<td>5298</td>
<td>631</td>
<td>217</td>
</tr>
<tr>
<td>Yes</td>
<td>greedy</td>
<td>X</td>
<td>X</td>
<td>5768</td>
<td>161</td>
<td>217</td>
</tr>
<tr>
<td>Yes</td>
<td>robust</td>
<td>X</td>
<td>X</td>
<td>5765</td>
<td>164</td>
<td>217</td>
</tr>
<tr>
<td>Yes</td>
<td>strict</td>
<td>X</td>
<td>X</td>
<td>5575</td>
<td>354</td>
<td>217</td>
</tr>
</tbody>
</table>

Mapping Evaluation

Table 6.4 summarizes how different factors affected the matching performance of the algorithm. Removing the smaller senses decreased the mismatched number from 731 to 595. After adjusting the values for \((c_{from}, c_{to})\) the number decreased further to 138. When the robust strategy was used, the mismatch increased marginally from 138 to 144. However, the processing time was faster. The same phenomenon happened when the DMRS was transformed using the ISF before the evaluation process, in which the number of mismatches stayed at 161 predicates out of 6146 available senses.

Looking deeper into the list of not matched senses, we found that with 84 instances, more than half of those senses were the copula verb “be”. There are some words that do not introduce any predicates in the ERG such as “ah” (5 instances), “for example” (1 instance), or “yes” (4 instances). There are also abnormal forms of words such as “to-night” for “tonight” which the ERG cannot parse.

Another group of identified problems comes from the disagreement between the available sense annotations and the ERG. For constructions such as “neither nor”, “both A and
In contrast, the human sense annotators also annotated “neither” and “both”. In relation to this, there are disagreements on when a construction should be recognized as a whole, such as “before ever” and _before+ever_x_h, “push back” and _push_v_back in ERG. There is a long tail of instances related to this class of errors.

When the ISF construction recognition is enabled, more multi-word expressions are captured, such as “dark-lantern”, “well-known”, or “all-comprehensive”. However, this also enables the detection of valid multi-word expressions that the human annotators ignored, such as “first person”, “round table”, “long term”, “iron bar”, or “yellow light”. Sometimes, the errors are from the sense annotators’ side. For example, in the segment ... how we conveyed her by the morning train to the care of her good aunt at Harrow ..., the ISF linked “morning train” to 80001228-n: a train (service) that runs in the mornings while the annotators treated it as two separated words “morning” and “train”.

**Mapping Validation**

Considering just the number of mapped senses is not enough to evaluate the performance of the mapping function. To evaluate how well the Pred-Sense Mapping works, the mapping results must be validated against gold data. First, the sentences and annotated senses were taken from NTUMC and converted into TTL format. Then, small senses and sentences which the ERG cannot parse were removed. After this, the senses were mapped onto two different data sets: speckled TSDB (gold MRS from ERG tree bank) and speckled ISF (gold MRS from ERG treebank that are transformed using the ISF). There are two available versions of the TSDB gold profile: Bridge- and Bridge+. Here, bridge refers to an option to activate bridging rules of the ERG to allow unifying fragmented structures that are not compatible (i.e. not grammatical). This feature allows more ungrammatical
sentences, which may introduces more bad parses, but may enable the grammar to parse more sentences at the same time (i.e. “The the dogs barks.”). Finally, the senses from the two data sets were extracted and mapped back to original sentences. Assuming a perfect mapping, we should expect a perfect score because the evaluation is using the gold dataset against itself.

<table>
<thead>
<tr>
<th>ISF</th>
<th>Profile</th>
<th>Recall</th>
<th>Precision</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Bridge-</td>
<td>99.19%</td>
<td>100.00%</td>
<td>99.60%</td>
</tr>
<tr>
<td>No</td>
<td>Bridge+</td>
<td>97.52%</td>
<td>100.00%</td>
<td>98.75%</td>
</tr>
<tr>
<td>Yes</td>
<td>Bridge-</td>
<td>98.86%</td>
<td>100.00%</td>
<td>99.43%</td>
</tr>
<tr>
<td>Yes</td>
<td>Bridge+</td>
<td>97.18%</td>
<td>100.00%</td>
<td>98.57%</td>
</tr>
</tbody>
</table>

Table 6.5: Sense-Pred Mapping Validation

From Table 6.5, when a sense can be matched, it is 100% correct. However recall is still not perfect. The reason for this is discussed in a previous section (tokens without predicates, missing predicates for copula verbs and multi-word expressions recognition is not perfect).

6.2 Pred-Sense Mapping

The second necessary task was Pred-Sense Mapping, or how to find potential senses for known predicates. To map predicates with Wordnet senses, the obvious method is using the lemmas and POS from DMRS predicates to search for Wordnet synsets using exact matching. This method is usable in many cases. However, due to differences between Wordnet taxonomy and ERG’s lexicon, this does not always provide a mapping.

When considering parts-of-speech, there are three different parts-of-speech that are linked to the corresponding a (adjective) in ERG for adjectives/adverbs. Other than the obvious pos='a' (adjective) and pos='r' (adverbs) we also have satellite adjective synsets.
Adding this mapping improved the mapping process significantly.

6.2.1 Extended Matching

In many cases, the Wordnet sense’s lemma and predicate’s lemma are identical. For example, the lemma _dog_n_1 can be easily linked to “dog”, which is is included in the following Wordnet synsets:

1. 02084071-n: “a member of the genus Canis (probably descended from the common wolf) that has been domesticated by man since prehistoric times; occurs in many breeds”
2. 02710044-n: “metal supports for logs in a fireplace”
3. 03901548-n: “a hinged catch that fits into a notch of a ratchet to move a wheel forward or prevent it from moving backward”
4. 07676602-n: “a smooth-textured sausage of minced beef or pork usually smoked; often served on a bread roll”
5. 09886220-n: “someone who is morally reprehensible”
6. 10023039-n: “informal term for a man”
7. 10114209-n: “a dull unattractive unpleasant girl or woman”

It is simple if the lemmas of the sense and the lemmas of the predicate are exactly the same. However, this is not always the case. For example, the ERG predicate for look up in the sentence I look the word up. is _look_v_up, but the Wordnet synset lemma is look up. In the ERG predicate, up is analyzed as a sense of the verb look. When the ERG and Wordnet do not agree with each other about the lemmas or parts of speech for predicates Pred-Sense mapping is also problematic. For example, the lemma for trap door in Wordnet is `trap door`, but for ERG it is `trapdoor`. Boundaries can also be different, for
example `a+couple' in ERG and `a couple of' in Wordnet both represent a couple of. Abbreviations or lemma variations are also sources of mismatch. For example, the concept saint is represented in ERG with `st' while Wordnet uses `saint'. Similarly, there is a mismatch between `suburban' in ERG and `suburban' in Wordnet, et cetera.

Interestingly, we encountered missing Wordnet concepts as well. For example, modal verbs such as would, could, should, might are not defined in Wordnet in detail – there is only one entry for the concept `modal'. As a result, a manual search of a list of modal verbs was required in order to map in this context.

Similarly, there are many words not existing in the ERG lexicon. When the ERG encounters an unknown word, the unknown word handling feature is activated and it will try to guess the best semantics available for the word. However, the format of the predicate for unknown word will be different from that for known words. For example, the word “ventilators” will raise the predicate _ventilators/NNS_u_unknown, which encodes the lemma as ventilators/NNS, the part of speech as u and the sense as unknown. Additional processing needs to be done to enable semantic analysis for these unknown predicates.

Also, Wordnet tends to use either space character or a dash to separate tokens while ERG uses a plus sign. To overcome this, all of the potential variations of a lemma were added before the query was performed.

When parsing sentences, the ERG introduces many grammatical predicates, such as quantifiers, superlatives, comparatives, negatives, modal verbs, et cetera. These grammatical predicates are linked manually to Wordnet synsets to provide a better and more consistent semantic analysis.

The pseudocode for the Pred-Sense algorithm that addresses the problems mentioned above can be found in Algorithm 6.3.
Algorithm 6.3: Pred-Sense Mapping algorithm

```plaintext
input: Pred pred
output: (concept1, concept2, ...)
begin
    concepts ← set()
    if pred in MANUAL_MAP:
        return MANUAL_MAP[pred]
    elif pred in IGNORED_PREDS or pred.sense == 'modal':
        return empty set
    elif pred.pos == 'u' and pred.sense == 'unknown':
        lemma, pos ← analyse_unknown_pred(pred.lemma)
        concepts = search_pred(lemma, pos)
    elif pred.pos in 'xpq':
        concepts = search_pred(lemma, pos=['a', 'r'])
    elif pred is grammarpred and pred.carg is not empty
        if pred['ARG0'].type == 'e':
            concepts ← search_pred(pred.lemma, pos='a')
        else:
            concepts ← search_pred(pred.lemma, pos='u')
    if pred.pos == 'a' and 'ARG1' in pred.args:
        lemmas = pred.lemmas + make_adverb(pred.lemmas)
        if pred['ARG1'].type in ('h', 'e'):
            pos ← 'r'
        else:
            pos ← 'a'
        concepts ← search_pred(lemmas, pos='r')
    if concepts is empty:
        pos ← 'r' if pos == 'a' else 'r'
        concepts ← search_pred(lemmas, pos)
    if concepts is not empty:
        return concepts
    else:
        return search_pred(pred.lemma, pred.pos)
end
```
6.2.2 Pred-Sense Disambiguation

By linking the senses directly to the DMRS predicates, it is possible to shift the paradigm from token-based semantic annotation (word-sense disambiguation) to predicate-based semantic annotation (pred-sense disambiguation). After a list of sense candidates is retrieved, we can use any Word-Sense Disambiguation method to rank and select the best sense(s) for each predicate. By separating the semantic representation from the source text, it is easier to create a complete semantic-oriented representation instead of a representation that is reflecting the syntax of the sentence. To document a linguistic phenomenon in a semantic-oriented (or functional-oriented) fashion, we start by describing the nature of the function, or the intended meaning of a particular utterance. This is done in Sign-Based Construction Grammar by describing the situation and related information such as individuals, predicates, argument structures, et cetera. Of course, the meaning must be triggered by a certain form (used to construct the sign) but the numbers of forms and of meanings do not have to be the same. In other words, in a DMRS representation, the number of elementary predicates and the number of tokens can be different.

This paradigm shift is more than just changing the number of form-sense links, but opens the door for a better semantic representation. In an ideal analysis, the semantics of different instances of the same phenomenon should always look the same. This is hard to achieve if we always constrain the semantic representation by syntax. One example of this is the comparative construction in English. When an adjective has two or fewer syllables, the comparative form is \textit{word-er} and when they have three or more syllables, another form \textit{more + adjective} is used. For example, even though “better” and “more beautiful” represent the same comparative construction, the former has only one word (token) while the latter has two. When these two phrases are sense-tagged, in most word-sense disambiguations “better” will be tagged with 00230335-a (comparative of good’) superior
to another (of the same class or set or kind) in excellence or quality or desirability or suitability; more highly skilled than another (Wordnet 3.0) while "more beautiful" will be tagged with 00099341-r used to form the comparative of some adjectives and adverbs and 00217728-a delighting the senses or exciting intellectual or emotional admiration (Wordnet 3.0)

The ISF analyses are more consistent. The two sentences *This painting is better* and *this painting is more beautiful* are analyzed as:

![Diagram of ISF analysis]

Here, the comparative constructions are more consistent, which were $more(e, e_1) \land good(e_1)$ and $more(e, e_2) \land beautiful(e_2)$ respectively.

### 6.2.3 Evaluation

To evaluate the Pred-Sense Mapping, the document *speckled* was used, in oracle mode. For each predicate available in a DMRS structure, the Pred-Sense mapping function was used to retrieve a list of candidate senses. This list of candidate senses was then compared to the gold profile. If the gold sense was found in the candidate list, it was counted as a match (hence the name oracle).
Table 6.6: Pred-Sense Mapping Evaluation

Table 6.6 shows that by using ISF transformation, the number of matched candidate senses increases. Looking into the senses that could not be found, we find similar problems to what happened in Section 6.1. Other than missing senses or lemmas (such as bell rope in Wordnet 3.0 or modal verbs) there is a long list of unknown predicates that are not matched perfectly. For example, as mentioned above, the word “ventilators” was parsed as _ventilators/NNS_u_unknown_rel. Although the algorithm is capable of handling unknown words, it does not have access to the dictionary form of the word, and thus cannot find the correct sense in Wordnet. Some adjectives are not available in the ERG, especially the derivative ones (e.g. “speckled”, which is derived from the verb “speckle”). For this reason, they are always passed as verbs by the ERG. This makes it more difficult to use part-of-speech information for sense mapping.

Table 6.7: Pred-Sense Evaluation result using ERG first parse for the Adventure of the Speckled Band

The test was also repeated on the silver speckled dataset, as shown in Table 6.7. Although this dataset does not have the bridging rules as in gold DMRS created by ERG annotators, the results are comparable to other profiles, with 100% precision and 97.81% recall.
6.3 Summary

In this chapter, I have discussed the relations between predicates and concepts, and how to provide mappings to unify them in an integrated representation. All the discussed features and algorithms were implemented in the ISF software package, which can produce Wordnet mappings for all ERG predicates whenever those mappings are available. The whole ERG lexicon, with more than 40,000 predicates, was passed through this software to generate 77,789 predicate-sense pairs. Although this mapping was generated based on a list of predicate strings, which did not include the full AVM structure of a DMRS, it is still useful for other systems that process MRS structures and do not have access to the ISF machinery.
Chapter 7

Integrated Semantic Framework

The final framework is built from the components introduced in Chapter 4, 5, and 6. It is implemented in Python 3. At the time of writing, it is capable of providing rich semantic analysis for English texts and contains a work-in-progress module for producing simple semantics for Japanese text. The system received input sentences in text format, combines analyses from ERG, ISF construction analyser, and LeLesk sense ranking module Figure 7.1 shows the complete work flow of the Integrated Semantic Framework system. This chapter elaborates on technical details of building the system, evaluating its performance, as well as comparing it to other systems in the context of the Word-Sense Disambiguation task.

7.1 ISF Workflow

After preprocessing the input text (sentence tokenization, morphological analysis, et cetera), an HPSG-based grammar is used to parse the raw text to get the semantics in DMRS format. The used grammar does not have to necessarily be an HPSG grammar, but any
Figure 7.1: Integrated Semantic Framework Flowchart
grammar or system that provides MRS analysis for texts. In the actual implementation of the Integrated Semantic Framework given a text, the ISF will perform a call to the pyDelphin library, which then spawns an ACE process to load the selected grammar (ERG) and process the input text.

For example, when the sentence *I drink drip coffee.* is passed into the English Resource Grammar the output is in MRS representation by default, as shown in Listing 7.1.

Listing 7.1: MRS for *I drink drip coffee.*

```
[ TOP: h0
  RELS: < [ pron<0:1> LBL: h1 ARG0: x8 [ x IND: + NUM: sg PERS: 1 PT: std ] ]
    [ pronoun_q<0:1> LBL: h2 ARG0: x8 RSTR: h13 ]
    [ _drink_v_1_rel<2:7> LBL: h3 ARG0: e9 [ e MOOD: indicative
      ARG1: x8
      ARG2: x12 [ x NUM: sg PERS: 3 ] ]
    [ udef_q<8:20> LBL: h4 ARG0: x12 RSTR: h14 ]
    [ compound<8:20> LBL: h5 ARG0: e10 [ e MOOD: indicative
      ARG1: x12
      ARG2: x11 [ x IND: + PT: notpro ] ]
    [ udef_q<8:12> LBL: h6 ARG0: x11 RSTR: h15 ]
    [ _drip_n_1_rel<8:12> LBL: h7 ARG0: x11 ]
    [ _coffee_n_1_rel<13:20> LBL: h5 ARG0: x12 ] >
  HCONS: < h0 qeq h3 h13 qeq h1 h14 qeq h5 h15 qeq h7 > ]
```

The ISF currently only works with DMRS representations, so it converts this MRS structure into DMRS format. This is done using the pyDelphin library. The equivalent DMRS representation is shown in Listing 7.2.
Listing 7.2: DMRS for *I drink drip coffee*

```plaintext
dmrs {
    10000 [pron<0:1> x ind=+ num=sg pers=1 pt=std];
    10001 [pronoun_q<0:1> x ind=+ num=sg pers=1 pt=std];
    10002 [_drink_v_1_rel<2:7> e mood=indicative
        perf=-- prog=-- sf=prop tense=pres];
    10003 [udef_q<8:20> x num=sg pers=3];
    10004 [compound<8:20> e mood=indicative perf=--
        prog=-- sf=prop tense=untensed];
    10005 [udef_q<8:12> x ind=+ pt=notpro];
    10006 [_drip_n_1_rel<8:12> x ind=+ pt=notpro];
    10007 [_coffee_n_1_rel<13:20> x num=sg pers=3];
    0;/H -> 10002;
    10001:RSTR/H -> 10000;
    10002:ARG1/NEQ -> 10000;
    10002:ARG2/NEQ -> 10007;
    10003:RSTR/H -> 10007;
    10004:ARG2/NEQ -> 10006;
    10004:ARG1/EQ -> 10007;
    10005:RSTR/H -> 10006;
}
```

In its graph form, the semantics looks like this:

![Graph representation of the DMRS]

Next, the ISF construction analyser is used to look for known constructions in obtained DMRS structures. In this step, a list of potential constructions will first be populated based on available predicates in the DMRS. Only the constructions that contain all the predicates in the DMRS are considered. For example, the compound construction *Irish Coffee*
(which contains _irish_a_1 and _coffee_n_1) is not considered because it contains the predicate _irish_a_1, which is not available in the DMRS structure of our working example. Each potential rule is matched against the DMRS structure by a built-in graph matching algorithm. When the topology is matched (i.e. predicate names, correct number and direction of links, et cetera), the rule can be fired to transform the original DMRS structure.

Because DRIP COFFEE is a known construction, the transformer module in the ISF will merge these two predicates into one:

\[ _\text{drip}_n_1 + _\text{coffee}_n_1 \rightarrow _\text{drip+coffee}_n_1 \]

In the original DMRS, DRIP COFFEE was represented by two separate predicates, _coffee_n_1 and _drip_n_1. However, it is represented by a single concept “drip coffee” as in the transformed DMRS. After the transformation, the DMRS structure looks like this:
Every time a known construction or a named-entity compound is found, a decision has to be made regarding whether we perform the transformation or not. For example “green tea” may refer to a tea that is green in color, and not the type of tea known as green tea.

If we do not make the decision and hold the ambiguities, the problem space will grow very fast (exponentially, $2^n$) along the way. For example, a single parse with four decisions to be made (to transform or not to transform) would generate $2^4 = 16$ new parses. This problem can be solved by using DMRS packing and unpacking\(^1\) as studied by Eveleigh (2010). The scope of this problem is another research project in itself and not to be

\(^1\)The research documentation and source code is available at [https://code.google.com/archive/p/cstitproject/downloads](https://code.google.com/archive/p/cstitproject/downloads)
discussed here. To simplify the problem, whenever there is a chance to transform the DMRS, the system will just perform it.

After the DMRSes are transformed, each predicate will be assigned a list of potential senses. As discussed in Chapter 6, Predicate-Sense mapping is not a trivial process. For example, pronoun_q is a quantifier and the sense is determined by the predicate itself instead of the senses\(^2\) in the ontology (Wordnet). On top of this, although the English Resource Grammar and Wordnet recognises POS attribute to denote usages of predicates and concepts, they are not always compatible. For instance, adjectives and adverbs are treated the same in the English Resource Grammar (i.e. they are both modifiers) while they are two different categories in Wordnet (r/adverb and a/adjective).

Some predicates need to be mapped manually (e.g. superl_rel or be_v_id_rel) as ERG and Wordnet employ different conceptualisation methods.

Concerning our working example, (7.3), after the Pred-Sense mapping process, the output will be the transformed DMRS plus this list of senses per predicate, as shown below:

- **_drink_v_1_rel**
  - 01170052-v: take in liquids
  - 01171183-v: consume alcohol
  - 01175467-v: propose a toast to
  - 00737005-v: be fascinated or spell-bound by; pay close attention to
  - 01172275-v: drink excessive amounts of alcohol; be an alcoholic

- **_drip+coffee_n_1_rel**

\(^2\)In theory, there is no restriction to conceptualize quantifiers in the ontology, this is yet to be done because there are more urgent mappings. For example, in the Open Multi-lingual Wordnet pronouns are recognized while they are not in the Princeton Wordnet.
– 07919894-n: coffee made by passing boiling water through a perforated container packed with finely ground coffee

If this step was performed before the transformation process, the result would be:

• _drink_v_1_rel
  – 01170052-v: take in liquids
  – 01171183-v: consume alcohol
  – 01175467-v: propose a toast to
  – 00737005-v: be fascinated or spell-bound by; pay close attention to
  – 01172275-v: drink excessive amounts of alcohol; be an alcoholic

• _drip_n_1_rel
  – 03241660-n: a projection from a cornice or sill designed to protect the area below from rainwater (as over a window or doorway)
  – 07383323-n: the sound of a liquid falling drop by drop
  – 07432559-n: flowing in drops; the formation and falling of drops of liquid

• _coffee_n_1_rel
  – 04972451-n: a medium brown to dark-brown color
  – 07929351-n: a seed of the coffee tree; ground to make coffee
  – 07929519-n: a beverage consisting of an infusion of ground coffee beans
  – 12662772-n: any of several small trees and shrubs native to the tropical Old World yielding coffee beans

Because of the transformation, the number of senses to be considered for _drip+coffee_n_1_rel is reduced significantly from seven different candidates to one.
Parse ranking is not a focus in this research, so the default best parse provided by ACE is used as default. Although there are many sense ranking algorithms available, implementation time is a big constraint of this research. To simplify the implementation process, I have adopted the ranking method from the LESK algorithm as it is easy to implement. The details of the implementation are discussed in Section 7.2.

With parse ranking and sense ranking being done, the ISF provides the complete semantic analysis in TTL/XML format.

### 7.2 Sense Ranking with LeLesk

The algorithm Enhanced Extended Simplified Lesk described in Section 2.5.4 was implemented as an open source software package named **LeLesk** and is available for public access at [https://github.com/letuananh/lelesk](https://github.com/letuananh/lelesk). LeLesk supports batch mode processing and allows many WSD tasks to be performed in a single execution. The input can be stored in either TTL or simple TSV (i.e. tab-separated values) files. In the case of a TSV input file, every line follows this format: word [tab] suggested-sense [tab] POS-optional [tab] Context (sentence) [tab] tokenized sentence (optional). The package is integrated into the ISF to provide sense ranking ability.

LeLesk was optimised in many rounds in order to achieve the current performance, which I believe is good enough for light production uses. The first method is to enhance data retrieval speed. By default, the PWGC is provided in XML/text format, which takes a long time to be processed. A script was developed to convert it into an SQLite database for faster access. In the first few releases of the package, the system required around 6 GB

---

3Opensource projects such as PyDelphin, Wordnet 3.0, Princeton Wordnet Gloss Corpus, lxml, Python 3, and SQLite 3 have been used to implement this software package. The source code of the implementation is licensed under the MIT license.

132
of RAM and around 7 seconds to perform WSD on a given word. This contrasts to the submitted version which takes a few seconds to start up and in general, it takes less than a second to disambiguate a sentence.

Next, the performance was improved by exploiting the fact that most of the calculations can be pre-computed except the overlapping between context and candidates’ token lists, which is a very fast operation (set intersection). The most expensive task in the algorithm is token expansion, which can be built beforehand and memoized using caching\(^4\) techniques. Thus given a token and its context, we only need to retrieve the senses’ token lists from the memory and then calculate the overlapping score. In the LeLesk package, token lists for all available synsets of the PWGC are generated and stored in an SQLite database. When a synset and a context are presented, the token list can be retrieved directly from the database instead of being rebuilt from Wordnet structures.

### 7.2.1 Evaluation and Discussion

The package LeLesk was tested by performing WSD on 599 sentences in the document *The Adventure of the Speckled Band*, available in the NTU-MC. This document has 6,352 words, which have been annotated by human annotators using Wordnet synsets. As LeLesk only provides analysis using Princeton Wordnet 3.0, from which the Gloss Corpus was developed, the new senses from Open Multilingual Wordnet were removed before the evaluation process takes place. The final number of senses which are used to evaluate is 4,543 senses. The gold annotations from NTU-MC were converted from IMI/SQLite format into TTL format. The extraction script for this task is published and is available at: [https://github.com/letuananh/omwtk](https://github.com/letuananh/omwtk).

\(^4\)Caching is a computational technique which frequently used information can be stored in computer memory for faster access.
The sentences from *The Adventure of the Speckled Band* were processed using two different morphosyntax analysers, the NLTK and Stanford tool chain. The outputs were passed into LeLesk for performing Word-Sense Disambiguation using two algorithms, Lesk and most-frequent sense (MFS - i.e. to pick the sense with the highest tag count from Wordnet). The evaluation results are shown in Table 7.1 and 7.2. On an average computer with quad-core processors and 8 GB of RAM, it takes around 2 minutes to disambiguate the whole document (127 seconds). Currently, Lesk is still outperformed by MFS for about 5% in F1 score.

<table>
<thead>
<tr>
<th>Profile</th>
<th>Recall</th>
<th>Precision</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLTK</td>
<td>41.10%</td>
<td>39.25%</td>
<td>40.15%</td>
</tr>
<tr>
<td>Stanford CoreNLP</td>
<td>41.95%</td>
<td>39.32%</td>
<td>40.59%</td>
</tr>
</tbody>
</table>

Table 7.1: WSD with Lesk for *The Adventure of the Speckled Band*

<table>
<thead>
<tr>
<th>Profile</th>
<th>Recall</th>
<th>Precision</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLTK</td>
<td>46.62%</td>
<td>44.52%</td>
<td>45.55%</td>
</tr>
<tr>
<td>Stanford CoreNLP</td>
<td>47.72%</td>
<td>44.72%</td>
<td>46.17%</td>
</tr>
</tbody>
</table>

Table 7.2: WSD with MFS for *The Adventure of the Speckled Band*

There are several reasons why a word may be assigned with a wrong sense. Firstly, the Wordnet version that was used in tagging NTU-MC has been extended with many new synsets by its annotators. Some of the synset IDs are old, but there are new lemmas added and they are not available to LeLesk. For example, in LeLesk’s Wordnet there are only three senses for *Watson* (3 people): *Thomas Augustus Watson* (the engineer), *John Broadus Watson* (the psychologist), and *James Dewey Watson* (the geneticist). None of these matches *Dr. Watson* (the character in Sherlock Holmes stories). Therefore the system always provides a wrong answer for *Watson*, as it does not have the correct concept to reference Dr. Watson in Sherlock Holmes. Similarly, there are other proper nouns that
are not available to LeLesk such as Stoke Moran or Grimesby Roylott. These need to be added manually into the LeLesk database in future releases.

Another important source of errors are multi-word expressions. For example, the sentence “Very sorry to knock you up, Watson” is tokenized into ['Very', 'sorry', 'to', 'knock', 'you', 'up', 'Watson'] and LeLesk cannot detect that the two tokens knock and up belong to a single construction. This class of error also contains (knock up, run over, draw up, dog cart et cetera).

Lastly, errors may also come from the use of unknown lemmas for some concepts. For instance, the concept 07083136-n (undertone) is not linked to the lemma low voice in Princeton Wordnet 3.0, but is used in NTU-MC. This effectively prevents the system from providing the correct sense disambiguation.

7.3 ISF Software Package

The Integrated Semantic Framework in this dissertation was implemented using Python 3 packages, and other resources such as English Resource Grammar, Minimal Recursion Semantics, ACE, and Wordnets. This package is to demonstrate how each task in the framework can be done computationally. It was designed using separation of concerns\(^5\) and implemented as different Python modules, mainly for better testability and integrability. The main components includd texttaglib for data manipulation and conversion, LeLesk for sense disambiguation related features, YAWlib for using data from different Wordnet projects, and coolisf - the main package contains modules for providing unified semantic analysis, most notably are the predicate-sense analyser module, the semantic graph analyser, the data conversion library, the construction recognition, and the transfor-

\(^5\)Separation of concerns is a principle for designing computer system in which different functions are divided into various packages. This is done in order to improve module reusability.
migration module. Finally, the package VisualKopasu or Visko serves as an optional GUI interface for corpus management, interactive development and visualization.

To the end users, the ISF can be used in two distinct modes: online mode and batch mode. In online mode, users can use an interface (web interface, terminal prompt, et cetera) to interact with the application. When users enter a sentence, the application will process the input sentence using a grammar, augment the output semantics with Wordnet senses, perform parse ranking and WSD, and then display the output in extended DMRS graph. This online mode is helpful for grammar engineering or for demonstration purposes, but it may not necessarily be useful for real-life applications.

In batch mode, multiple sentences can be written into a file (either text file with one sentence per line or in TTL format), and then passed to the ISF to be processed in a single execution. The output is provided in ISF/XML format for further analysis or evaluation. It is possible to perform either a full analysis (pass in a text document and retrieve semantic graphs in an XML output file) or using different tool-chains to compare the performance and contribution of each step along the way. For example, users can parse the sentence raw file using ACE, and then use the ISF to convert ACE output file into ISF/XML format. From there, other intermediate tasks available within ISF can be performed individually. This is useful considering the expensive nature of deep computational analysis. As ERG remains unchanged, the output of ACE will always be the same for the same input. Thus constructional analysis can be tested separately without re-parsing each sentence every time it is invoked. The same applies for the Predicate-Sense Analysis. Once the transformation process is fixed, the output remains unchanged and can be reused for faster development.

To use the ISF, first a grammar profile needs to be created. It has to be a compiled grammar using the Answer Constraint Engine (ACE\textsuperscript{6}) binary format. The grammar must

\textsuperscript{6}http://moin.delph-in.net/AceTop
declare any usage of pre-processors (modules that process sentences before they are sent to ACE) and post-processors (modules that run after the ACE output is produced). These processors are useful for working with different languages. For instance when parsing Japanese text, the ISF uses Jacy grammar, which requires its Japanese input to be segmented. This is done by the Japanese Shallow Processor using MeCab\(^7\) and a Python wrapper for Python. Another example is the ISF construction analyzer. As it has to wait for ACE to finish parsing a sentence in order to run, it is implemented as a post processor. The parser code itself is developed based on the available API of pyDelphin\(^8\) which acts as a mediator that calls ACE and returns DMRS structure.

The construction analyzer is implemented from scratch as a Python module and is available as both a post-processor and a feature in the package. Thus, users can invoke this separately without having to use the whole process of the ISF analyzer.

There are three ways to use the system:

1. Use the Visko software suite, where the users can use a graphical user interface to parse sentences, perform transformations, create or edit DMRS structures manually, search for phenomena through out corpus, et cetera.

2. Use the command line interface to parse single sentences and read the output in text mode. A batch mode is also provided which processes text files and output to files or terminal’s standard output stream.

3. Use different tools for different steps (i.e. ACE, LeLESK, Stanford POS tagger, et cetera) and output to files, then stream these files into the ISF (using TTL format) to integrate them to generate a semantic analysis.

\(^7\)http://taku910.github.io/mecab/
\(^8\)https://github.com/delph-in/pydelphin
As for LeLesk, since it is embedded into the ISF, users can choose any of the following modes to perform semantic analysis (summarized in Figure 7.2):

1. Use default batch mode of the ISF to analyze raw text. This mode provides both construction analysis and semantic analysis, so the output is sense-tagged. The results can be fed into other systems using TTL format.

2. Use LeLesk to perform WSD directly on DMRS structures.

3. Use other systems such as Core NLP or NLTK to perform tokenization and POS tagging, convert this data into TTL format, and pass the data into LeLesk to perform WSD (NLTK is used by default in LeLesk)

4. If other systems are used for performing semantic analysis, the output can be converted into TTL format for inter-system comparison.

7.4 ISF Evaluation

The performance of the ISF was evaluated by analysing the Adventure of the Speckled Band (the same as in Section 7.2) and then compared to other systems (UKB (Agirre and Soroa, 2009) and Babelfy (Moro, Cecconi, and Navigli, 2014)) mainly by the metric of Word-Sense Disambiguation.

7.4.1 Evaluation Method

It is difficult to compare different semantic analysis systems because they follow different methods and provide different representations. Systems such as UKB or Babelfy provide token-based semantics. They split text into tokens and then assign senses to them. Some
systems, like Babelfy, only output fragments that have senses, while all other tokens are discarded. LeLesk uses a similar work flow with NLTK for word tokenizer and Wordnet lemmatizer, but keeps all tokens throughout the process.

On the other hand, the ISF does not link senses to syntactic structures or tokens, but integrates them in semantic constructions. There are words which do not introduce new predicates, but constrain the relation between predicates. For example in Figure 7.3, there is no predicate for the copula “be”. The copula only introduces a constraint between the ARG1 of _happy_a_with and the ARG0 of pron to make the former modifies the latter.

The output from the systems is normalized to a common format for comparison. As different systems provide senses in different ways (i.e. linked to tokens, predicates, sentence fragments), the only thing that they share is the original text. Thus, for evaluation purposes, each sense in the output is mapped back to the original text (string format) as a bag
Figure 7.3: DMRS of *I am happy.*
of triplets \( (c_{from}, c_{to}, \text{synsetID}) \). In texttaglib this can be done by creating sentence-level tags. To evaluate a profile against the gold profile, the TTL Evaluator scans through each sentence in the gold profile. For each synset in gold annotation of the sentence, a list of locations of the same synset in the other profile will be retrieved. If a location can be found in the retrieved list, it will be counted as matching. Otherwise, it will be counted as not matching. This algorithm is described in Algorithm 7.4.

Algorithm 7.4: Algorithm for TTL Evaluation

```plaintext
input: gold, profile
output: matched_list, not_matched_list
begin
for each sent in gold:
    for each synset in sent:
        plocs ← get all locs in profile(sent.ID)
        for each gloc in synset:
            for each ploc in plocs:
                if overlap(ploc, gloc):
                    matched_list.add(gloc)
                    synset.remove(gloc)
        no_matched_list.add(gloc)
return matched_list, no_matched_list
end
```

When given a profile and a gold set of annotations, the evaluation function returned two lists: matched and not matched. The matched list contained true positives and the not-matched list contained false negatives. These two numbers were used to calculate recall, precision, and F1 scores.

The next step was to create evaluation profiles. The performance of the ISF was evaluated using “The Adventure of the Speckled Band”. NLTK and Core NLP were used to create two baseline profiles using LeLesk. These baselines were compared to six different profiles using the ISF. The DMRS structures were retrieved in several ways: from the ERG
gold repository (referred as the Dan profiles), with and without bridging rules, with and without ISF transformation, and the first parse from ACE parse ranking module.

In total, there are eight profiles: gold DMRS with bridging rules and ISF transformation, gold DMRS with ISF transformation, gold DMRS with bridging rules, just gold DMRS, first parses from ACE with ISF transformation, just first parses from ACE, and the four baselines mentioned above. These profiles are then evaluated in three different ways. The first method is best-of-each, which means a profile is judged only by the sentences that it can process (e.g. there is at least one parse returned from ERG). The second method is intersection. This method tests based on the sentences that all profiles can parse. The last method is All. In this method there is no filtering, if a profile could not process a sentence, it will be penalized for all the missing senses.

Each profile mentioned above was tested based on two different ranking methods: Lesk and Most-Frequent-Sense (MFS). In total, the ISF was evaluated using 48 different experiments.

To create the first four gold-DMRS profiles, the DMRS structures of two test suites available in the ERG (with and without bridging rules\(^9\)) were exported into ISF/XML format. These two profiles (Dan and Dan/bridge) were passed into the ISF construction analyzer to be transformed. This step provided two more profiles (Dan/ISF and Dan/bridge/ISF). The next two profiles (ERG and ERG/ISF) were created similarly, except the DMRSes came from the first parses that ACE/ERG returns instead of a tree bank. Next, these six profiles were converted into ISF/XML format and were analyzed by the ISF predicate-sense disambiguation module. Finally, the sense-annotated profiles were exported back to TTL format (senses as sentence-level tags) and were ready for comparison.

The two baselines were created using NLTK and Stanford Core NLP to analyze the

\(^9\)Bridging rules are special parsing rules for joining fragments of sentences that cannot be parsed as a whole.
sentences (tokenization and POS tagging). The outputs were then converted into TTL and passed directly into LeLesk. The output of LeLesk was in TTL format by default so no conversion was needed. Details on the different setups are summarised in Table 7.3.

<table>
<thead>
<tr>
<th>Profile</th>
<th>Gold DMRS</th>
<th>Bridging</th>
<th>ISF Transformation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan/bridge/ISF</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>To estimate ranking power</td>
</tr>
<tr>
<td>Dan/ISF</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>To estimate ranking power</td>
</tr>
<tr>
<td>Dan/bridge</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>To determine the contribution of construction analysis</td>
</tr>
<tr>
<td>Dan</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>To determine the contribution of parse ranking</td>
</tr>
<tr>
<td>ERG/ISF</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Real life performance of the ISF</td>
</tr>
<tr>
<td>ERG</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Real life performance of the ISF without construction analysis</td>
</tr>
<tr>
<td>NLTK</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Baseline #1</td>
</tr>
<tr>
<td>Stanford tagger</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Baseline #2</td>
</tr>
<tr>
<td>UKB</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>State-of-the-art #1</td>
</tr>
<tr>
<td>Babelfy</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>State-of-the-art #2</td>
</tr>
</tbody>
</table>

Table 7.3: The Adventure of the Speckled Band - Lesk - Best of each

The evaluations were done in three distinct modes:

1. In no filter mode, the whole document was used, which contains 4,598 annotated senses and 599 sentences.

2. In intersection mode, there were 3,895 senses in 562 sentences that could be processed by all profiles.

3. In best of each mode, the number of sentences and senses varied from profile to profile.

In addition, two external WSD systems, UKB and Babelfy, were also used to evaluate how well the ISF performs compared to state-of-the-art products.

143
7.4.2 Discussion

Overall, predicate-based sense analysis almost always outperformed word-based sense analysis. Babelfy scored the highest precision (48%) compared to all other methods, including UKB. However, its recall was too low, at 35.88%, which resulted in a very low final F1 score, at 41.23%. It performed at about the same level as NLTK/LESK method.

On the other hand, UKB is truly the state-of-the-art system. Its scores were high in both recall (49.83%) and precision (46.84%), which resulted in the F1 score of 48%. It outperformed most setup of the ISF. The only time that the ISF performed on the same level with UKB is when MFS ranking method was used. In general use cases, it is expected to score lower than UKB by 1-2% in F1 score.

Tables 7.4, 7.5, and 7.6 show that precision stayed consistently between 43.59% and 45.82%, compared to the 39.32% of shallow parsing methods (NLTK/Lesk, CoreNLP/Lesk). This stayed true even in no filter mode, where a profile was penalized for not being able to process a sentence. For recall, predicate-based methods were almost always better than token-based methods, except for all mode, where there were too many sentences that could not be parsed by the default ACE/ERG settings.

<table>
<thead>
<tr>
<th>Profile</th>
<th>Recall</th>
<th>Precision</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan/bridge/ISF</td>
<td>49.59%</td>
<td>45.82%</td>
<td>47.63%</td>
</tr>
<tr>
<td>Dan/ISF</td>
<td>49.32%</td>
<td>45.63%</td>
<td>47.41%</td>
</tr>
<tr>
<td>Dan/bridge</td>
<td>48.91%</td>
<td>44.05%</td>
<td>46.35%</td>
</tr>
<tr>
<td>Dan</td>
<td>48.64%</td>
<td>43.87%</td>
<td>46.14%</td>
</tr>
<tr>
<td>ERG/ISF</td>
<td>48.69%</td>
<td>44.95%</td>
<td>46.75%</td>
</tr>
<tr>
<td>ERG</td>
<td>48.10%</td>
<td>43.59%</td>
<td>45.73%</td>
</tr>
<tr>
<td>NLTK</td>
<td>41.10%</td>
<td>39.25%</td>
<td>40.15%</td>
</tr>
<tr>
<td>Stanford tagger</td>
<td>41.95%</td>
<td>39.32%</td>
<td>40.59%</td>
</tr>
<tr>
<td>UKB</td>
<td>49.83%</td>
<td>46.84%</td>
<td>48.29%</td>
</tr>
<tr>
<td>Babelfy</td>
<td>35.88%</td>
<td>48.45%</td>
<td>41.23%</td>
</tr>
</tbody>
</table>

Table 7.4: *The Adventure of the Speckled Band* - Lesk - Best of each
### Table 7.5: The Adventure of the Speckled Band - Lesk - Intersection

<table>
<thead>
<tr>
<th>Profile</th>
<th>Recall</th>
<th>Precision</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan/bridge/ISF</td>
<td>49.40%</td>
<td>45.81%</td>
<td>47.54%</td>
</tr>
<tr>
<td>Dan/ISF</td>
<td>49.37%</td>
<td>45.79%</td>
<td>47.51%</td>
</tr>
<tr>
<td>Dan/bridge</td>
<td>48.63%</td>
<td>44.03%</td>
<td>46.21%</td>
</tr>
<tr>
<td>Dan</td>
<td>48.63%</td>
<td>44.03%</td>
<td>46.21%</td>
</tr>
<tr>
<td>ERG/ISF</td>
<td>48.42%</td>
<td>44.77%</td>
<td>46.52%</td>
</tr>
<tr>
<td>ERG</td>
<td>47.88%</td>
<td>43.44%</td>
<td>45.55%</td>
</tr>
<tr>
<td>NLTK</td>
<td>41.23%</td>
<td>39.32%</td>
<td>40.26%</td>
</tr>
<tr>
<td>Stanford tagger</td>
<td>42.08%</td>
<td>39.37%</td>
<td>40.68%</td>
</tr>
<tr>
<td>UKB</td>
<td>49.58%</td>
<td>46.53%</td>
<td>48.00%</td>
</tr>
<tr>
<td>Babelfy</td>
<td>35.88%</td>
<td>48.45%</td>
<td>41.23%</td>
</tr>
</tbody>
</table>

### Table 7.6: The Adventure of the Speckled Band - Lesk - No Filter

When *MFS* was used as a ranking method, the precision of token-based profiles increased from 39.32% to 44.72%. This is slightly lower than predicate-based methods, which stayed in the range 43.87% - 45.82%. As for recalls, all profiles performed similarly in the range of 47% - 49%. Details of these comparisons can be found in Tables 7.7, 7.8, and 7.9.

Among predicate-based methods, both precision and recall increased around 0.5%-1.5% when ISF transformation was activated. By recognizing constructions, the ISF was able to provide better sense candidates for multi-word expressions. When DMRS were taken from the first parse that comes out of ACE/ERG instead of gold DMRS, there was a slight
drop in precision. For Lesk ranking method, precision dropped from 45.82% to 44.95% and recall dropped from 49.59% to 48.69%. Tables 7.7, 7.8, and 7.9 show the same trend with MFS ranking method (i.e. precision dropped from 45.82% to 45.65%, recall dropped from 49.59% to 49.45%).

There was one exception when no filter comparison was used. Because there were sentences that could not be parsed, recall dropped really low (from 49% to around 43%). However, precision remained high as usual. Comparing the F1 scores, the simplest profile ERG was almost always higher than the two baselines by about 5% to 7%. The only time this was not true is shown in Table 7.9, where the F1 score was lower than the baselines by about 1%.

<table>
<thead>
<tr>
<th>Profile</th>
<th>Recall</th>
<th>Precision</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan/bridge/ISF</td>
<td>50.36%</td>
<td>46.54%</td>
<td>48.37%</td>
</tr>
<tr>
<td>Dan/ISF</td>
<td>50.22%</td>
<td>46.46%</td>
<td>48.27%</td>
</tr>
<tr>
<td>Dan/bridge</td>
<td>49.70%</td>
<td>44.77%</td>
<td>47.11%</td>
</tr>
<tr>
<td>Dan</td>
<td>49.54%</td>
<td>44.68%</td>
<td>46.99%</td>
</tr>
<tr>
<td>ERG/ISF</td>
<td>49.45%</td>
<td>45.65%</td>
<td>47.48%</td>
</tr>
<tr>
<td>ERG</td>
<td>48.91%</td>
<td>44.32%</td>
<td>46.50%</td>
</tr>
<tr>
<td>NLTK</td>
<td>46.62%</td>
<td>44.52%</td>
<td>45.55%</td>
</tr>
<tr>
<td>Stanford tagger</td>
<td>47.72%</td>
<td>44.72%</td>
<td>46.17%</td>
</tr>
<tr>
<td>UKB</td>
<td>49.83%</td>
<td>46.84%</td>
<td>48.29%</td>
</tr>
<tr>
<td>Babelfy</td>
<td>35.88%</td>
<td>48.45%</td>
<td>41.23%</td>
</tr>
</tbody>
</table>

Table 7.7: *The Adventure of the Speckled Band* - MFS - Best of each

The results from 48 experiments above are summarized in Table 7.10, using three representative profiles: gold-DMRS based, ACE first parse based, and NLTK/Lesk. From these results, we can see that the predicate-sense candidate analysis method had higher precision when compared to word-sense candidate analysis, with about a 10% increment. At the same time, there was no compromise on recall, except for the no filter mode due to unparsed sentences.
<table>
<thead>
<tr>
<th>Profile</th>
<th>Recall</th>
<th>Precision</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan/bridge/ISF</td>
<td>50.22%</td>
<td>46.57%</td>
<td>48.33%</td>
</tr>
<tr>
<td>Dan/ISF</td>
<td>50.22%</td>
<td>46.57%</td>
<td>48.33%</td>
</tr>
<tr>
<td>Dan/bridge</td>
<td>49.50%</td>
<td>44.82%</td>
<td>47.04%</td>
</tr>
<tr>
<td>Dan</td>
<td>49.50%</td>
<td>44.82%</td>
<td>47.04%</td>
</tr>
<tr>
<td>ERG/ISF</td>
<td>49.24%</td>
<td>45.53%</td>
<td>47.31%</td>
</tr>
<tr>
<td>ERG</td>
<td>48.75%</td>
<td>44.23%</td>
<td>46.38%</td>
</tr>
<tr>
<td>NLTK</td>
<td>46.34%</td>
<td>44.20%</td>
<td>45.24%</td>
</tr>
<tr>
<td>Stanford tagger</td>
<td>47.52%</td>
<td>44.46%</td>
<td>45.94%</td>
</tr>
<tr>
<td>UKB</td>
<td>49.58%</td>
<td>46.53%</td>
<td>48.00%</td>
</tr>
<tr>
<td>Babelfy</td>
<td>35.88%</td>
<td>48.45%</td>
<td>41.23%</td>
</tr>
</tbody>
</table>

Table 7.8: *The Adventure of the Speckled Band* - MFS - Intersection

<table>
<thead>
<tr>
<th>Profile</th>
<th>Recall</th>
<th>Precision</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan/bridge/ISF</td>
<td>48.65%</td>
<td>46.54%</td>
<td>47.57%</td>
</tr>
<tr>
<td>Dan/ISF</td>
<td>45.65%</td>
<td>46.46%</td>
<td>46.05%</td>
</tr>
<tr>
<td>Dan/bridge</td>
<td>48.01%</td>
<td>44.77%</td>
<td>46.33%</td>
</tr>
<tr>
<td>Dan</td>
<td>45.04%</td>
<td>44.68%</td>
<td>44.86%</td>
</tr>
<tr>
<td>ERG/ISF</td>
<td>44.40%</td>
<td>45.65%</td>
<td>45.02%</td>
</tr>
<tr>
<td>ERG</td>
<td>43.91%</td>
<td>44.32%</td>
<td>44.12%</td>
</tr>
<tr>
<td>NLTK</td>
<td>46.62%</td>
<td>44.52%</td>
<td>45.55%</td>
</tr>
<tr>
<td>Stanford tagger</td>
<td>47.72%</td>
<td>44.72%</td>
<td>46.17%</td>
</tr>
<tr>
<td>UKB</td>
<td>49.83%</td>
<td>46.84%</td>
<td>48.29%</td>
</tr>
<tr>
<td>Babelfy</td>
<td>35.88%</td>
<td>48.45%</td>
<td>41.23%</td>
</tr>
</tbody>
</table>

Table 7.9: *The Adventure of the Speckled Band* - MFS - No Filter

Examining the actual analyses, there were a few interesting insights to be taken into consideration.

First, since the gold DMRS and the sense tags were done by two different groups, the process of integrating these two datasets revealed some disagreements between the annotators. As there are no modal verbs in Wordnet, none of them should have been sense tagged. The gold DMRS profile and the ISF did very well without sense tagging modal verbs. However, the shallow methods such as NLTK/Lesk made the same mistake as the human annotators who did the sense tags. In this case, since the mistakes were shown in
<table>
<thead>
<tr>
<th>Mode</th>
<th>Profile</th>
<th>Recall</th>
<th>Precision</th>
<th>F1</th>
<th>Recall</th>
<th>Precision</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intersection</td>
<td>Dan/bridge/ISF</td>
<td>49.40%</td>
<td>45.81%</td>
<td>47.54%</td>
<td>50.22%</td>
<td>46.57%</td>
<td>48.33%</td>
</tr>
<tr>
<td></td>
<td>ERG/ISF</td>
<td>48.42%</td>
<td>44.77%</td>
<td>46.52%</td>
<td>49.24%</td>
<td>45.53%</td>
<td>47.31%</td>
</tr>
<tr>
<td></td>
<td>NLTK</td>
<td>41.23%</td>
<td>39.32%</td>
<td>40.26%</td>
<td>46.34%</td>
<td>44.20%</td>
<td>45.24%</td>
</tr>
<tr>
<td></td>
<td>Dan/bridge/ISF</td>
<td>49.59%</td>
<td>45.82%</td>
<td>47.63%</td>
<td>50.36%</td>
<td>46.54%</td>
<td>48.37%</td>
</tr>
<tr>
<td>BoE</td>
<td>ERG/ISF</td>
<td>48.69%</td>
<td>44.95%</td>
<td>46.75%</td>
<td>49.45%</td>
<td>45.65%</td>
<td>47.48%</td>
</tr>
<tr>
<td></td>
<td>NLTK</td>
<td>41.10%</td>
<td>39.25%</td>
<td>40.15%</td>
<td>46.62%</td>
<td>44.52%</td>
<td>45.55%</td>
</tr>
<tr>
<td></td>
<td>Dan/bridge/ISF</td>
<td>47.90%</td>
<td>45.82%</td>
<td>44.33%</td>
<td>44.40%</td>
<td>45.65%</td>
<td>45.02%</td>
</tr>
<tr>
<td>No filter</td>
<td>ERG/ISF</td>
<td>43.72%</td>
<td>44.95%</td>
<td>44.33%</td>
<td>44.40%</td>
<td>45.65%</td>
<td>45.02%</td>
</tr>
<tr>
<td></td>
<td>NLTK</td>
<td>41.10%</td>
<td>39.25%</td>
<td>40.15%</td>
<td>46.62%</td>
<td>44.52%</td>
<td>45.55%</td>
</tr>
</tbody>
</table>

Table 7.10: Summary of ISF evaluation experiments

the gold dataset, this means that we were testing against imperfect data. The annotators also annotated 128 instances of the copula verb, which did not introduce any new predicates by the ERG, as mentioned above in Figure 7.3.

Secondly, there were also some disagreements when the annotators had to choose between adjectives and nouns. For instance, there were 21 numbers annotated as nouns where they should have been annotated as adjectives. For example, in the sentence *The chimney is wide, but is barred up by four large staples*, the lemma four was annotated as a noun (using 13744304-n the cardinal number that is the sum of three and one) where it should have been annotated as an adjective (using 02186665-a being one more than three) instead.

Thirdly, there were cases where additional lemmas were added during the annotation process, some lemmas such as “bell-rope” could not be matched by the ISF, which uses the gloss tag corpus to search for candidates. As the gloss tag corpus was developed based on the default release of Princeton Wordnet 3.0, it does not contain the extra lemmas that were available to the annotators.

Although additional work has been done to ensure that the ISF system can be used in practical applications, the main focus of this research was to improve semantic analysis by
providing an alternative for guessing sense candidates. By using the same sense ranking methods (Lesk, MFS) for both token-based candidate searching and predicate-based candidate searching, we can see a strong correlation between using predicate-based method and large increment in recall, precision and F1 score.

Finally, although the chosen ranking algorithms for the ISF were extremely simple (word overlapping with LESK, or sense frequency score with MFS), the ISF scored much higher than Babelfy, and is not far behind (about 2%) from UKB, which utilises graph-based ranking method. I predict that with further research into integrating graph-based ranking algorithms with predicate-based semantic analysis, the scores that are presented here can be improved even further.

7.4.3 Future works

ISF is a novel approach which aims at investigating the foundational theory that supports existing semantic analysis systems. There is much work to be done in order to integrate available resources for the same tasks from other approaches. For token-based semantics, the accuracy of tokenization and part-of-speech tagging are important. For the ISF, it is the accuracy of gold DMRS structures and parse ranking method. However, they are expensive to build. I understand the concerns regarding the generalizability of this method since it was done only on the short story “the Adventure of the Speckled Band”. I will wrap up this chapter with a preliminary result from my future research. The ISF and the UKB was used to analyse the final dataset\textsuperscript{10} from the International Workshop on Semantic Evaluation 2015, Task 13 (Moro andNavigli, 2015). Table 7.11 shows the final results. For this particular dataset, the ISF has better recalls (48.70% compared to 44.91%) but lower precision (41.91% compared to 45.36%). Overall, the ISF is on par with UKB in term of

\textsuperscript{10}This dataset was retrieved online in February 2019 at http://alt.qcri.org/semeval2015/task13/index.php?id=data-and-tools
F1 score (45.05% and 45.14%).

<table>
<thead>
<tr>
<th>Profile</th>
<th>Recall</th>
<th>Precision</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERG/ISF/LESK</td>
<td>48.70%</td>
<td>41.91%</td>
<td>45.05%</td>
</tr>
<tr>
<td>ERG/ISF/MFS</td>
<td>48.91%</td>
<td>42.10%</td>
<td>45.25%</td>
</tr>
<tr>
<td>ERG/LESK</td>
<td>48.91%</td>
<td>41.21%</td>
<td>44.73%</td>
</tr>
<tr>
<td>ERG/MFS</td>
<td>49.13%</td>
<td>41.39%</td>
<td>44.93%</td>
</tr>
<tr>
<td>UKB</td>
<td>44.91%</td>
<td>45.36%</td>
<td>45.14%</td>
</tr>
</tbody>
</table>

Table 7.11: WSD results for Semeval 2015, Task 13

Error investigation was held back intentionally as this is an important benchmark dataset, and I would like to keep it unseen to prevent potential biases in future developments of the ISF. The results are presented here solely for evaluating the performance of the ISF to other systems on truly unseen data.
Chapter 8

Visko - Software Suite for Computational Linguistic Analysis

This chapter introduces Visko, a software suite which provides tools for grammar engineering and linguistic documentation in an integrated environment. Here are some basic features that are provided:

1. Corpus organization
2. Indexing and searching
3. Interactive semantic analysis with debugging
4. Importing and Exporting data (comments, flags)
5. Data back up

8.1 ISF Corpus Structures

The TTL format that is described in Chapter 4 is adopted for presenting ISF analysis. This includes new levels of organization and interaction with the data. Following are the
available levels of an ISF corpus:

1. Collections (*.db files)
2. Corpora (bags of documents)
3. Documents (list of sentences)
4. Sentences (list of utterances)
5. Shallow (a single shallow parse with tokens, dependency tree, or related information)
6. Readings (MRSes, DMRSes)
7. Semantic representations (MRS, DMRS, AMR, et cetera)
8. DMRS nodes (with tags from different tagging methods)
9. DMRS links

First, a sentence can have multiple readings rather than a single reading. For example, the sentence *I saw a hot dog.* may mean “I saw a sausage.” or “I saw an animal of type dog that was hot.”. Instead of choosing the intended readings at an early stage, it is also possible to record multiple readings and choose it later.

Ideally, we should have a universal semantic theory that could be transformed into different representations when needed. In practice, different schools/systems adopt different theories and create different representations. It is difficult to account for all of them. As a result, reading in ISF is just a vague grouping method for the same (or close enough) interpretations from different analyses.

The ISF representation itself does not impose any specific theory or representation. For instance in the sentence *how much does he weigh?* (AMR’s The Little Prince corpus), the reading is represented in AMR formalism as:

\[(w / weigh-01\]
:ARG1 (h / he)
:ARG3 (a / amr-unknown))

At the same time, the ERG reading for the same sentence looks like this:

dmrs {
    10000 [generic_entity<0:8> x num=sg pers=3];
    10001 [undef_q<0:8> x num=sg pers=3];
    10002 [abstr_deg<0:3> i ];
    10003 [which_q<0:3> i ];
    10004 [measure<0:3> e mood=indicative perf=- prog=-
           sf=prop tense=untensed];
    10005 [much-many_a<4:8> e mood=indicative perf=- prog=-
           sf=prop tense=untensed];
    10006 [pron<14:16> x gend=m ind=+ num=sg pers=3 pt=std];
    10007 [pronoun_q<14:16> x gend=m ind=+ num=sg pers=3 pt=std];
    10008 [_weigh_v_1_rel<17:23> e mood=indicative perf=- prog=-
           sf=ques tense=pres];
    0:/H -> 10008;
    10001:RSTR/H -> 10000;
    10003:RSTR/H -> 10002;
    10004:ARG2/NEQ -> 10002;
    10004:ARG1/EQ -> 10005;
    10005:ARG1/EQ -> 10000;
    10007:RSTR/H -> 10006;
    10008:ARG2/NEQ -> 10000;
    10008:ARG1/NEQ -> 10006;
Currently, only DMRS structures are indexed. Any other semantic representations (e.g. AMR) are stored as-is in raw text strings.

Figure 8.1: Corpus management features in Visko software suite

8.2 Visualizing Composite Semantics

Visko provides an interface to visualize several semantic analyses at the same time.
1. TTL Concepts

2. MRS-based representations

   (a) DMRS Dependency graph

   (b) MRS AVM

   (c) text-based MRS & DMRS

      i. pyDelphin text-based format

      ii. XML

      iii. JSON

3. Wordnet senses

DMRS analyses can be visualized in different ways. The most common ways include: dependency graphs, AVM, and text (list of nodes and links).
8.2.1 Semantic Structures Visualization

Sentences with shallow semantic annotations are displayed as lists of tokens. If a token is annotated with a concept, it will be stylized as bold. If the linked concept is a single-token concept, the token will be colored with blue. On the other hand, if the concept is multiple-token (i.e. multi-word expressions), the token will be colored in green. Underneath the token list is a list of concepts. If a concept is not available in other levels of semantic analysis, it suggests a conflict in analyzing methods and will be highlighted with red. An example is provided in Figure 8.3.

* My name is Sherlock Holmes.

Figure 8.3: TTL Visualization in Visko

For deep semantic analyses, such as the ones provided by HPSG grammars like ERG or JACY, a dependency graph is used, as in Figure 8.4. A list of labels are displayed as nodes (predicates) and labeled arrows are used as links from nodes to nodes.

8.2.2 A Single Visualizer

Everything is brought together in a single view. The TTL tokens and concepts are listed on top, and the deep analyses at the bottom. Different representations are organized in different tabs. The bottom area is for writing analysis notes and assigning flags (Gold, error, warning, et cetera) to sentences. When a TTL concept or a predicate’s sense tag is clicked, Visko will search and display the sense information (for example the synset 09604451-n as shown in Figure 8.4.)
8.3 Searching and Indexing

Searching and indexing are tasks that go beyond raw substring matching. To this end, Visko provides multiple different searching methods:

1. search by predicate’s lemma
2. search by predicate’s string
3. search by predicate’s CARG (constant arguments)\(^1\)
4. search by sentence’s IDs
5. search by sentence’s tags and flags
6. Basic query operators like AND and OR are also provided

\(^1\)CARG or constant arguments, parameterized relations are mostly used for named entities, and a limited number of other concepts such as season names.
8.4 Interactive Development

A graphical user interface is provided to parse text on the fly. Users can choose different grammars, profiles, WSD algorithms, et cetera. This helps compare different grammar profiles/configurations and evaluate how they affect the final outcome.

When there is no good analysis, it is also possible to edit a provided DMRS or create a new DMRS directly in Visko to get the desired analysis. The format is text-based and human-friendly. Within the DMRS visual editor screen, users can generate sentences from the edited DMRS source code, or perform a construction transformation to see if available construction rules are working properly. Because the DMRS structure is verified automatically, the DMRS is guaranteed to be well-formed. Figure 8.5 shows how interactive development is made available in Visko.

Figure 8.5: DMRS interactive development
Chapter 9

Conclusions

This research introduces a novel method for integrated semantic analysis that unifies lexical semantics and structural semantics in a single representation. The proposed method is implemented as a software package that takes textual input and provides DMRS-based semantic analyses. The performance of this system was evaluated on a short Sherlock Holmes story which was annotated by human annotators and compared to other systems on Word-Sense Disambiguation. The final results show that, by exploiting constructional information and predicate-sense mappings, the accuracy and recall can be improved significantly (from 48.10% to 49.59% for recall and 43.59% to 45.82% for precision). Overall, the F1 score also improved from 45.73% to 47.63% for Lesk and 46.50% to 47.63% for MFS.

When the same text is analyzed using several methods (for example deep parsing using ERG or morphosyntax analysis using NLTK) and the results are integrated using a unified platform such as TTL, where interesting phenomena can also be observed in more detail. In predicate-based semantics like those produced by the ERG, not all tokens introduce new predicates – an example of this is the copula “be” that occasionally only introduces a new constraint between two existing predicates. In shallow parsing methods, such as NLTK
tokenizer and tagger, it is harder to decide which words are meaningful as there isn’t much information available at the time decisions need to be made. When the results are unified, these cases are more clearly revealed and can be cross-referenced to verify the original gold dataset. Some annotation mistakes were discovered in the gold dataset using this method.

The gold DMRS profile with special bridging rules also pointed out an interesting fact; the predicates provided are still useful for semantic analysis, including the sentences that are parsed incorrectly or only partially. This fact can be exploited to create hybrid semantic analysis systems. When a sentence cannot be parsed, robust parsing or similar techniques can be adopted to provide partial parses, so some output can still be provided.

### 9.1 Deliverables

Following are the software and data sets that are generated by this research:

1. **TextTagLib** - a powerful and robust linguistic documentation framework which is flexible enough to integrate linguistic data from different schools of research.
2. **LeLESK** - a Word-Sense Disambiguation software package for Python.
3. **YAWLib** - a Python module for accessing data from different Wordnet projects (Wordnet SQL, Glossed Wordnet corpus, Open Multilingual Wordnet).
4. A complete, customizable workflow for computational semantic analysis.
5. An integrated semantic representation, developed based on DMRS and unifying lexical and structural semantics.
6. **Visko** - A software suite for computational linguistic analysis, with a full featured graphical user interface (including corpus management, semantic visualizer, multi-feature search, DMRS parsing, generating, edit, et cetera).
7. The software implementation of the Integrated Semantic Framework in Python 3\textsuperscript{1}. It can be used separately or within the Graphical User Interface of Visko software suite.

8. A Construction-Sense database – a database of English constructions that was generated from Princeton Wordnet 3.0; and construction transformer – a software module to recognize and transform known constructions within DMRS structures.


10. Several open datasets\textsuperscript{2}

   (a) The Adventure of the Speckled Band (599 sentences) - with sense and DMRS structures developed by human annotators.

   (b) Construction database (lemmas - MRS structures - synset - target predicate name) with 470,444 rules.

\subsection*{9.2 Future Research Plan}

\subsubsection*{9.2.1 Study of English Constructions}

The next step in my research is to refine the construction detection and conversion so that a larger set of phenomena in English can be covered. This can be done by using a larger corpus with more linguistic resources. The phenomena which currently interest me most are the relations between verb constructions and the corresponding thematic roles as well as lexicalised idioms.

\textsuperscript{1}The source code of the Integrated Semantic Framework has been licensed under MIT license and can be obtained at https://github.com/letuananh/intsem.fx

\textsuperscript{2}The data and evaluation results can be downloaded at http://letuananh.dakside.org/intsemfx
9.2.2 Better Sense Ranking Methods

LESK algorithm is used in this research because it is easy to be customised to work with different parsing methods (NLTK, ERG). It is also suitable, within the available resources, to build the final working system. The next step is to try adopting better sense ranking methods and making them work with the ISF. I believe that this would improve the performance of the ISF even further.

9.2.3 Multilingual Approach

I am also looking forward to experimenting with languages other than English. There are many other languages which have been documented using various methods and I would like to use existing grammars (including but not limited to Japanese, Chinese, Indonesian, Abui, Vietnamese, et cetera) for my integrated semantic framework. By adapting the system with different languages, we can gain more knowledge about the complex nature of meaning in different languages. In the current state of this research, the ISF is capable of parsing Japanese text using the grammar JACY and Japanese Wordnet. When the lexical mapping between JACY predicates and Japanese Wordnet lemmas is completed, the ISF will also be able to provide full semantic analysis for Japanese. The same process can be adopted for Chinese (using the ZHONG grammar\(^3\)) or Indonesian (the INDRA grammar\(^4\)).

\(^3\)https://github.com/delph-in/zhong
\(^4\)https://github.com/davidmoeljadi/INDRA
Appendices
Appendix A

Software and Resources

Following is a list of software and resources which I used to develop this framework.

ACE  Answer Constraint Engine
     http://sweaglesw.org/linguistics/ace/

ERG  English Resource Grammar
     http://svn.delph-in.net/erg/trunk/

LOGON a collection of software, grammars, and other linguistic resources to facilitate experimentation with transfer-based machine translation (MT), or other experimentation with 'deep' parsing or generation
     http://moin.delph-in.net/LogonTop

NLTK  Natural Language Toolkit - a platform for building Python programs to work with human language data
     http://www.nltk.org/

NTU-MC  NTU Multilingual Corpus
     http://compling.hss.ntu.edu.sg/ntumc/
OMW  Open Multilingual Wordnet
   http://compling.hss.ntu.edu.sg/omw/

pyDelphin  Python libraries for DELPH-IN
   https://github.com/delph-in/pydelphin

Semcor  Semcor Wordnet 3.0
   http://web.eecs.umich.edu/~mihalcea/downloads.html#semcor

Wordnet SQL  Wordnet Database in various SQL format
   http://sourceforge.net/projects/wnsql/

... and many more other free and open source products (Python, SQLite, et cetera).
This dissertation is written in Emacs on Fedora GNU/Linux OS and is typeset in \LaTeX.
Appendix B

Source Code

Following are links for public access, source codes, and resources that were created in this project.

**Visko**  Visual Kopasu - Software suite for Computational Linguistic Analysis

https://github.com/letuananh/visualkopasu

**ISF**  Python implementation of the Integrated Semantic Framework

https://github.com/letuananh/intsem.fx

**LeLesk**  Word Sense Disambiguation package for Python

https://github.com/letuananh/lelesk

**texttaglib**  A Text Annotation Framework implemented in Python

https://github.com/letuananh/texttaglib

**YAWLib**  Yet Another Wordnet library for Python

https://github.com/letuananh/yawlib
pySemcor  A library for processing Semcor with Python

https://github.com/letuananh/pysemcor
Bibliography


Bond, Francis, Sheefa Samara Sameha, and Dan Flickinger (2013). “Making English Possessed Idioms Our Own”. In: The 20th International Conference on Head-Driven Phrase Structure Grammar (HPSG 2013). (accepted but not presented as no-one was free to attend). Berlin.


Bondy, John Adrian and Uppaluri Siva Ramachandra Murty (1976). Graph theory with applications. The Macmillan Press Ltd.


170


