



All Theses and Dissertations

---

2014-06-24

# OntoSoar: Using Language to Find Genealogy Facts

Peter Lindes

*Brigham Young University - Provo*

Follow this and additional works at: <https://scholarsarchive.byu.edu/etd>



Part of the [Linguistics Commons](#)

---

## BYU ScholarsArchive Citation

Lindes, Peter, "OntoSoar: Using Language to Find Genealogy Facts" (2014). *All Theses and Dissertations*. 4133.  
<https://scholarsarchive.byu.edu/etd/4133>

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact [scholarsarchive@byu.edu](mailto:scholarsarchive@byu.edu).

OntoSoar: Using Language to Find Genealogy Facts

Peter Lindes

A thesis submitted to the faculty of  
Brigham Young University  
in partial fulfillment of the requirements of the degree of  
Master of Arts

Deryle W. Lonsdale, Chair  
David W. Embley  
Alan K. Melby

Department of Linguistics and English Language

Brigham Young University

June 2014

Copyright © 2014 Peter Lindes

All Rights Reserved

## ABSTRACT

### OntoSoar: Using Language to Find Genealogy Facts

Peter Lindes

Department of Linguistics and English Language, BYU  
Master of Arts

There is a need to have an automated system that can read family history books or other historical texts and extract as many genealogy facts as possible from them. Embley and others have applied traditional information extraction techniques to this problem in a system called OntoES with a reasonable amount of success. In parallel much linguistic theory has been developed in the past decades, and Lonsdale and others have built computational embodiments of some of these theories using Soar. In this thesis we introduce a system called OntoSoar which combines the Link Grammar Parser using a grammar customized for family history texts with an innovative semantic analyzer inspired by construction grammars to extract genealogical facts from family history books and use them to populate a conceptual model compatible with OntoES with facts derived from the text. The system produces good results on the texts tested so far, and shows promise of being able to do even better with further development.

Keywords: information extraction, genealogy, linguistic theory, cognitive semantics, construction grammar, cognitive architectures

## ACKNOWLEDGEMENTS

Deryle Lonsdale has been my leading light for over two years now as I have attempted to learn something of linguistics and put that knowledge to work in a practical system. His teaching in the two courses I took from him and his consistent mentoring have both been outstanding. Without him I never would have gotten this far.

David Embley helped me enormously in making the transition to being a full time graduate student, and taught me a lot about conceptual modeling and information extraction. His persistent, insightful guidance throughout this thesis project has been invaluable.

Alan Melby got me thinking about agency and its importance in building systems that try to understand human language. This and his general positive attitude and encouragement have meant a lot to me.

LoriAnne Spear is amazing. She has guided me through so many twists and turns of learning how to succeed in the academic world, and without her help I never would have found my way out of the maze. Caleb McNeil, Michael Buckley, and Denise Remy have all also given much needed help.

Through these last two years my wife Bianka has never failed to support me and be my best cheerleader. She is the light of my life. Our children and grandchildren have also motivated me to keep going and be successful.

Thank you very much to all these people who are dear to me.

## TABLE OF CONTENTS

OntoSoar: Using Language to Find Genealogy Facts.....	i
ABSTRACT .....	ii
ACKNOWLEDGEMENTS .....	iii
TABLE OF CONTENTS.....	iv
LIST OF FIGURES .....	vii
LIST OF TABLES .....	viii
1. Introduction.....	1
2. Related Work .....	5
Information extraction from text .....	7
Ontology matching .....	9
Formal semantics .....	10
Cognitive semantics.....	10
Construction grammar .....	13
OntoES .....	14
Link Grammar Parser.....	15
Soar .....	16
Soar and language.....	17
3. Thesis Statement .....	18
4. Method .....	19
Examples .....	19

Target ontologies .....	25
Levels of knowledge .....	27
System architecture.....	30
Segmentation .....	31
Parsing.....	35
Building meanings.....	39
Semantic analysis.....	48
Ontology matching .....	53
Extraction of facts .....	55
Final output .....	57
5. Results .....	63
Results for the two samples .....	63
Persons .....	63
Births and Deaths .....	66
Marriages .....	67
Sons and Daughters.....	68
Accuracy measures .....	70
Analysis of errors.....	71
Results on additional samples .....	73
Run time performance .....	79

Results with different ontologies .....	79
6. Conclusions and Future Work .....	81
What has been demonstrated .....	81
Possible incremental improvements .....	83
Possible major additions .....	83
Future research directions.....	85
7. References .....	88

## LIST OF FIGURES

Figure 1: Sample 1 of Genealogy Text: from Vanderpoel (1902), p. 419..	2
Figure 2: Sample 2: from Harwood (1911), p. 84 .....	3
Figure 3: Meanings Derived from CCL Example.....	21
Figure 4: Meanings Derived from Myra Example .....	24
Figure 5: Ontology Example 1 .....	26
Figure 6: Ontology Example 2 .....	27
Figure 7: OntoSoar Block Diagram.....	30
Figure 8: Construction Example 1.....	41
Figure 9: Construction Example 1 with Meanings .....	42
Figure 10: Construction Example 2 with Meanings .....	44
Figure 11: Semantic Analysis of CCL 2.....	49
Figure 12: Meanings Derived from Myra 18-20.....	50
Figure 13: An Example of Inferencing .....	52
Figure 14: Ontology Example 3 .....	80



## LIST OF TABLES

Table 1: Person Facts for Sample 1 .....	64
Table 2: Person Facts for Sample 2 .....	64
Table 3: Births for Sample 1 .....	66
Table 4: Births for Sample 2 .....	66
Table 5: Deaths for Sample 1 .....	67
Table 6: Deaths for Sample 2 .....	67
Table 7: Marriages for Sample 1 .....	68
Table 8: Marriages for Sample 2 .....	68
Table 9: Sons and Daughters for Sample 1 .....	69
Table 10: Sons and Daughters for Sample 1 .....	69
Table 11: Accuracy Measures for Sample 1 .....	70
Table 12: Accuracy Measures for Sample 2 .....	70
Table 13: Combined Accuracy Measures .....	70
Table 14: Error Reason Codes .....	71
Table 15: Precision Data for Additional Texts .....	75

## **1. Introduction**

*Thus, intelligence is the ability to bring to bear all the knowledge that one has in service of one's goals.*

*Newell (1990), p. 90*

There is a great demand for genealogical data so that people can understand and document their family history. There is also a great supply of historical documents containing such data, but most were generated long before modern digital technology was available. This thesis addresses the problem of how we can extract this information from these historical documents in a digital form so it can be searchable.

Approaches to this general problem vary greatly depending upon the type of document involved. Census records, for example, are very structured by columns and rows with certain information always found in the same column. For this type of document the main problem is reading the handwritten data. This can be done, as it was done recently for the 1940 US census, by human indexers reading the handwritten text and typing it into a computerized form. An approach like this works well for documents of this type.

Another type of document available is a large set of family history books written before the digital age. Over 100,000 such books, many with several hundred pages each, have already been digitized by scanning them into PDF files and using OCR algorithms to extract the raw text. Of course the OCR process introduces a sizable number of errors. Dealing with OCR errors is beyond the scope of this project, although a few simple errors are corrected while preprocessing the text.

Once a book has been digitized, manual methods somewhat like those used for census records can be applied. Tools exist for showing each page on a screen so a user can go through and laboriously fill out forms for different kinds of information by clicking on the data values in the page of displayed text. However, this is an enormous task, both because of the millions of pages of text involved and because the text is not structured like a census form or any other kind of form. Extracting this kind of information, even when no handwriting is involved, is a much more complicated endeavor. Some way of automating this whole process would be of enormous benefit. This thesis presents one way of addressing this problem.

Here are two examples of text from these books. Sample 1 in Figure 1 is part of page 419 of an 830-page book (Vanderpoel, 1902).

243314. Charles Christopher Lathrop, N. Y. City, b. 1817, d. 1865, son of Mary Ely and Gerard Lathrop; m. 1856, Mary Augusta Andruss, 992 Broad St., Newark, N. J., who was b. 1825, dau. of Judge Caleb Halstead Andruss and Emma Sutherland Goble. Mrs. Lathrop died at her home, 992 Broad St., Newark, N. J., Friday morning, Nov. 4, 1898. The funeral services were held at her residence on Monday, Nov. 7, 1898, at half-past two o'clock P. M. Their children:

1. Charles Halstead, b. 1857, d. 1861.
2. William Gerard, b. 1858, d. 1861.
3. Theodore Andruss, b. 1860.
4. Emma Goble, b. 1862.

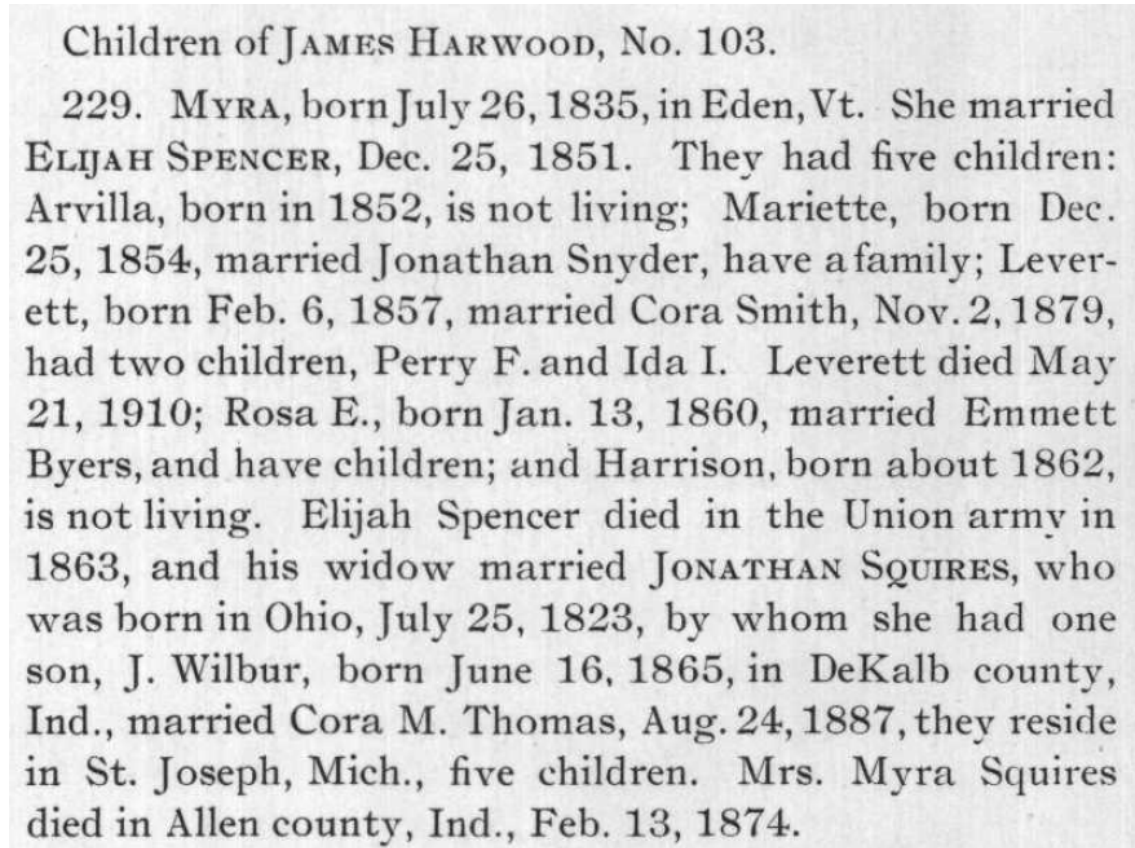
Miss Emma Goble Lathrop, official historian of the New York Chapter of the Daughters of the American Revolution, is one of the youngest members to hold office, but one whose intelligence and capability qualify her for such distinction.

Figure 1: Sample 1 of Genealogy Text: from Vanderpoel (1902), p. 419

Many parts of this book have information in a fairly structured form, as can be seen in the list of children in Figure 1. However, much of the rest of the text, except for the paragraph markings and identifying numbers for people like

243314., is only structured by English grammar rules. Not only that, but the text is often not standard English, having been greatly abbreviated, both lexically and syntactically. Also, there is much information that goes beyond simple names, dates, and places to involve information like how someone's intelligence can qualify her for a position of official historian in some organization.

Sample 2 in Figure 2 comes from a 197-page book (Harwood, 1911) which uses a much more free-flowing style in its text.



Children of JAMES HARWOOD, No. 103.

229. MYRA, born July 26, 1835, in Eden, Vt. She married ELIJAH SPENCER, Dec. 25, 1851. They had five children: Arvilla, born in 1852, is not living; Mariette, born Dec. 25, 1854, married Jonathan Snyder, have a family; Leverett, born Feb. 6, 1857, married Cora Smith, Nov. 2, 1879, had two children, Perry F. and Ida I. Leverett died May 21, 1910; Rosa E., born Jan. 13, 1860, married Emmett Byers, and have children; and Harrison, born about 1862, is not living. Elijah Spencer died in the Union army in 1863, and his widow married JONATHAN SQUIRES, who was born in Ohio, July 25, 1823, by whom she had one son, J. Wilbur, born June 16, 1865, in DeKalb county, Ind., married Cora M. Thomas, Aug. 24, 1887, they reside in St. Joseph, Mich., five children. Mrs. Myra Squires died in Allen county, Ind., Feb. 13, 1874.

Figure 2: Sample 2: from Harwood (1911), p. 84

Automatically extracting information from books like these does not have to address handwriting analysis, but does depend on higher-level knowledge.

Using the insight from Allen Newell quoted at the beginning of this chapter, and with a bit of introspection about how we ourselves get information out of text, we can see that several different kinds of knowledge can be brought to bear on this problem. The levels of knowledge that would be useful are at least: textual, syntactic, lexical, semantic, pragmatic, and world knowledge. This thesis is about building a system we call *OntoSoar* in an attempt to apply all these levels of knowledge to the problem of extracting information from genealogy books.

Once information has been extracted from the text, we need to output it to a searchable database. One part of *OntoSoar* will read in a user-defined knowledge representation structure and map the information extracted into that format. Then the resulting data in the terms of this conceptual model will be written out as the final product of processing a given text.

In the remaining chapters we first summarize related work that has been done in several fields, following this with a statement of the hypothesis we hope to prove. We then outline in detail the methods used by *OntoSoar*. In the results chapter we show how well the system works for the two sample texts given above, as well as on a set of test texts randomly selected from a large corpus of family history books. We then discuss what these results mean. We don't expect to solve the whole problem in one master's thesis, but we do hope to show the viability of an approach that can then be further built upon. We end with some conclusions and ideas for future work.

## 2. Related Work

To address the problem of finding genealogy facts in family history books we draw on extensive research over several decades in both linguistics and computer science. In linguistics we have the long tradition of generative linguistics with its concentration on evolving theories of syntax and related theories of formal semantics. Formal semantics is closely related to theories in computer science regarding conceptual models and using them to extract information from text. This in turn involves natural language processing, which draws to some degree on linguistic theory.

As we shall see, both these streams of research have a common limitation: they are trying to understand words in terms of other words without being grounded<sup>1</sup> in the real world. Our pursuit of genealogy facts, however, requires models of meaning grounded in world knowledge related to the lives of people and their family relationships. Both generative linguistics and traditional natural language processing fall short of providing grounded meaning that will allow us to build reasoning power into the system which can make inferences like: a *widow* is a woman who was married to a man who has died. Where can we get this grounding?

More recent branches of research in both linguistics and computer science have begun to address this problem. Ever since Lakoff and Johnson (1980a, summarized in 1980b) we have a stream of research in linguistics,

---

<sup>1</sup> By not being *grounded* we mean these approaches do not include any connection between words and their meaning in the outside world.

often called *cognitive linguistics*, which attempts to ground the meaning of linguistic expressions in human perception and experience. Some computer scientists have begun to use this kind of approach to build systems for understanding human language, as exemplified by Feldman (2006) and related work. Another branch of computer science has tried to build models of human cognition, called *cognitive architectures*. These theories draw heavily on experimental evidence from psychology and measurements of how the brain processes information. Anderson (2007) gives a good introduction to this field.

This thesis is based on the proposition that these various streams of research are now ready to merge into a larger river, and that we can begin to build systems such as OntoSoar by combining some of the best ideas from several of these fields.

Our solution draws on previous work in data extraction<sup>2</sup> and in using the Soar<sup>3</sup> architecture to process natural language. Both the Link Grammar Parser<sup>4</sup> and Soar are fundamental components of the OntoSoar system. Finally, the innovative semantic analyzer built here is based on a number of ideas derived from the literature on cognitive semantics and construction grammar. In this chapter we will review related work that has been done in all these areas.

---

<sup>2</sup> The focus of research of the Data Extraction Group at BYU. See Embley et al. (2011) and discussion of OntoES below.

<sup>3</sup> Soar is a cognitive architecture capable of complex reasoning. See discussion below.

<sup>4</sup> The Link Grammar Parser is an open-source parsing algorithm that is both robust and flexible (Sleator and Temperley 1991, 1993). See discussion below.

## Information extraction from text

Much research has been done on ways of extracting useful information from various kinds of texts. Various amounts of linguistic knowledge have been used in different systems.

In his ambitious work, Cimiano (2006) addresses not only populating an ontology<sup>5</sup> from text, but also using text to learn ontologies for given domains. Ontology learning is beyond the scope of this thesis. Nevertheless, Cimiano gives an excellent review of what ontologies are, how they can be represented, and how available natural language processing techniques can be used to extract information from text. However, Cimiano's focus is on the ontologies, and the language knowledge involved is rather superficial and inadequate for the needs of the current project.

With regard to natural language processing, Jurafsky and Martin (2008) have published a classic textbook on the subject. It examines in detail how various techniques from computer science, such as regular expressions, hidden Markov models, etc., can be applied to language processing. Many of these techniques can be useful in the current project to some degree, but again the emphasis is on the mathematical algorithms and not on the complexities of real natural language.

Buitelaar et al. (2009) present an approach to linguistic grounding of ontologies they call *LexInfo*. They argue that “currently available data-models are not sufficient ... without linguistic grounding or structure ... .” Although

---

<sup>5</sup> The term *ontology* is often used in the information extraction literature to mean a computerized conceptual model that can be populated with facts. See Embley et al. (2011).



this moves in the direction of attaching some language features to ontologies, it does not seriously consider the complexity of constructing meaning from natural language.

Sarawagi (2008) reviews the whole field of information extraction. It surveys “techniques from machine learning, databases, information retrieval, and computational linguistics for various aspects of the information extraction problem.” Notably absent from this list is anything addressing a deep understanding of language.

Akbik and Bross (2009) present a very interesting approach to extracting semantic relations from text using what they call “deep linguistic patterns.” They use the Link Grammar Parser and look for paths through the linkages between entity references to discover relations between these entities. Since OntoSoar also uses the Link Grammar Parser, many of the details are quite related. However, this work is not directly applicable to our problem since the relations are identified just by the words they contain as strings of text without any understanding of what those words actually mean.

Another approach sometimes called “machine reading” is discussed in depth by Hruschka (2013). He reviews in depth three systems for building knowledge bases by machine reading the web: YAGO, KnowItAll, and NELL. Each system starts with some seed knowledge and uses various techniques to make both the accumulated set of facts and the underlying ontology grow by reading large amounts of knowledge from the web. However, these systems still have fairly low accuracy in extracting individual facts and are not tuned to

the special sublanguages of English used in many family history books. In addition, all these systems are relating words to other words; there is no external grounding anywhere. Without such grounding there is no understanding of the true meaning of anything, and thus no basis for drawing inferences based on world knowledge.

Two groups at BYU have worked on problems closely related to this thesis, the Data Extraction Group (Embley et al., 2011) and the NL-Soar Research Group (Lonsdale et al., 2008). More detail will be given below on these efforts.

### **Ontology matching**

One of the features of OntoSoar is its ability to take extracted information in its internal representation of the meaning of input text and transform that information to populate an ontology provided by the user. This amounts to a special case of the general problem of ontology matching, for which there is also a large literature.

An overview and survey of this field is given by Euzenat and Shvaiko (2007). They discuss at great length the applications, techniques, and systems in this field. Bleiholder and Naumann (2008) and Mitra et al. (2004), as well as many others, discuss specific approaches in more detail. Most of this literature deals with how to map information from the Internet from one web site to another, or onto some pre-defined ontology. Fortunately for us our ontology mapping problem is much simpler since we are working within a well understood domain.

## **Formal semantics**

Much of the work on understanding the meaning of linguistic expressions has been done in the field of formal semantics. This field is summarized well by Chierchia and McConnell-Ginet (2000). Here we see predicate calculus and model-theoretic semantics used to explain the meanings of sentences. However, as we shall see when we discuss LG-Soar, this approach does not provide a model rich enough to support the reasoning needed to accomplish our task.

## **Cognitive semantics**

For OntoSoar to work, we need a way of representing, manipulating, and drawing inferences from the meaning of our input text. Soar provides a tool for doing this sort of thing, but we still need to design the data structures needed to represent meaning as well as the algorithms for processing these structures. All this together must produce a deep understanding of the text being processed, which means a deep understanding of the language used in the text.

In parallel with the progress in information extraction there has been over the last several decades a tremendous growth in linguistic theory that can explain syntactic, semantic, and other linguistic phenomena over a wide range of the world's languages. Most of this work has been done according to the *generative linguistics* paradigm first applied to language by Chomsky (1957). In recent years the generative approach has spawned Chomsky's Minimalist Program (Chomsky, 1995) for syntax and Jackendoff's theory of Conceptual Semantics (Jackendoff, 1990, 1996, 2002, and 2003). Unfortunately, these

approaches have been centered on syntax and formal semantics without either representing human language processing or being very useful for finding the meanings we need for our project. In addition, there has not been a great deal of practical application of these linguistic theories to building information extraction systems.

Starting a few decades ago another line of research called *cognitive linguistics* began. This approach does attempt to model how human beings process language and the deep semantic structures needed to understand meaning. Lakoff and Johnson (1980a) launched an approach to meaning describing how metaphor is used at every level to map our direct perceptual experience into higher level abstractions. Johnson (1987) develops one aspect of this theory with the concept of what he calls *image schemata*, data structures which can form a bridge between direct perception and symbolic representations of meaning. Lakoff (1987) explored much further how metaphor is used to build up complex meaning categories.

Johnson (1987) builds a theory of image schemata that are rooted in bodily experience and then extended by analogy and metaphor to provide structure to more abstract meanings. He argues this approach has much greater explanatory power than ordinary formal semantics:

*... on the view I am advancing, neither image schemata nor their metaphorical extensions exist only as propositions. They can be propositionally represented, but this does not capture their full reality as structures of our embodied understanding.*

*Johnson (1987) p. 103*

This is a key point for this thesis. The semantic analyzer we present here depends on using schemas similar to those described by Johnson to build an internal representation of the meaning of each sentence or sentence fragment. This rich schematic representation of meaning can then be used to reason about those meanings to produce a great deal of inferred data that would not otherwise be possible to derive. Then we can project these rich meaning structures onto a conceptual model based on formal semantics. However, without the richer intermediate representation, the number of facts that could be derived and the flexibility in projecting them would be greatly limited.

Furthermore, this rich internal structure makes it possible to build meaning structures that can represent important concepts in the same way despite a great deal of variation in the surface structure of the language used to represent them. Johnson makes this point as follows:

*Thus, the hypothesis of underlying metaphorical systems of understanding makes it possible to explain what has hitherto remained unexplained, namely, the systematic clustering of literal expressions associated with a single concept.*

*Johnson (1987) p. 106*

Shortly we will see examples of this idea at work.

More recently this line of cognitive semantics research has been turned into concrete language processing systems by a group at UC Berkeley headed by Jerome Feldman and George Lakoff. Feldman (2006) summarizes this approach, and much of its substance is amplified by Bryant (2008), Chang (2009), and several other dissertations. A central component of this research is a grammatical theory called Embodied Construction Grammar (Bergen and

Chang, 2003 and 2013). ECG has been used as one model for the semantic analyzer used in OntoSoar.

### **Construction grammar**

A number of linguists have pursued the idea of *construction grammar*, which fits well into the cognitive linguistics tradition. Hoffman and Trousdale (2013) give a good overview of this field. In Chapter 2 of this handbook Adele E. Goldberg states concerning constructionist approaches:

*Most of the approaches represented in this volume share important underlying assumptions that position the entirety of these approaches at a far remove from mainstream generative grammar.*

*Hoffman and Trousdale (2013) p. 15*

She outlines the main common tenets shared by a variety of construction grammar approaches. Briefly, these are: that a construction is a learned pairing of form and function; that semantics is associated directly with surface form without any transformational or derivational component; that constructions are related in a network that includes inheritance links; that there is a great deal of variation across languages; and that knowledge of language is usage-based, including both specific items and generalizations. Tomasello (2003) applies this usage-based approach to language acquisition.

As mentioned above, Feldman's group has applied the idea of construction grammar to a computer implementation that they call ECG. Many of the ideas from their work, especially those discussed by Bergen and Chang (2013) and Bryant (2008) have been drawn from and adapted to produce the semantic analyzer presented in this thesis.

## **OntoES**

The Data Extraction Group at Brigham Young University has been developing for some time a system called OntoES that extracts data from a variety of text types, including family history books. The basic approach used by OntoES is to start with a conceptual model or ontology (Embley et al., 1992), and augment the ontology with *recognizers*. A recognizer is a formula including a regular expression plus references to lexicon files that can be applied to a text to extract references to a certain type of entity or relationship.

The complete OntoES system consists of a number of useful tools. Conceptual models can be represented in XML in a format called OSMX, which contains the object and relationship sets of the ontology as well as various augmentations such as recognizers and facts extracted from a text to populate the ontology. There is a tool called the Workbench which is a Java program that allows a user to build ontologies graphically and examine any data they have been populated with. There is also an Annotator tool which allows a user to annotate a text with respect to a given conceptual model.

OntoSoar fits into this overall OntoES system by reading in a user ontology in OSMX form and outputting a modified OSMX file which contains the facts it found in a given input text. In addition, OntoSoar can be evaluated by using the Workbench to compare the facts found by OntoSoar with those found by a human annotator in the same text.

## Link Grammar Parser

A key part of the linguistic analysis needed for OntoSoar is syntactic parsing. We need a parser that is both robust enough to cover a wide range of English syntax, flexible enough to be adapted to the non-standard English found in the text of family history books, and available in a form that we can use.

Several general purpose parsers for English are available as open source tools. The Stanford Parser<sup>6</sup> is the best known of these. It is a statistical parser which has been trained on a large annotated corpus of news wires. It can produce phrase structure trees or typed dependencies for standard English. However, the only way to get it to work for our non-standard English would be to manually annotate a large corpus of family history text and retrain the parser on that corpus. Many other easily available parsers use the same approach and even produce output in the same format as the Stanford Parser.

A good alternative is the Link Grammar Parser (Sleator and Temperley, 1991 and 1993). Rather than a statistical parser trained on an annotated corpus, this parser uses a large dictionary of word classes and rules for linking words together in a sentence. It produces an output called a *linkage*, which is a labeled, undirected graph showing links between words. As well as being very robust, this parser can easily be adapted to the non-standard forms we need to deal with by modifying its dictionary. Therefore this is the parser chosen for use in OntoSoar.

---

<sup>6</sup> See description at <http://nlp.stanford.edu/software/lex-parser.shtml>.



## **Soar**

Our goal for OntoSoar is to understand the meaning of the text we are processing so that we can transform that meaning into facts to populate a searchable ontology. This requires a way of representing the semantics of the input in a complex meaning graph and a reasoning engine of some sort that can construct this graph, perform inferences on it to derive implicit information, and transform it into a form that can be used to populate the target ontology.

For many years there has been research into cognitive architectures, attempts at building computational models of how human beings think. Anderson (2007) gives an excellent explanation of what a cognitive architecture is and an overview of one particular exemplar called ACT-R. He does not address the question of how to use these architectures to process natural language, but does quote from Marcus (2001) with regard to what Marcus calls the “symbol manipulation hypothesis.” Marcus shows how abstract relations between variables, recursively structured representations, and mental representations of individuals and kinds are essential to how the human mind works, and also speculates on how these things might be represented by neural networks.

Another prominent cognitive architecture is called Soar (Newell, 1990; Laird, 2012). We have decided to use Soar as our system for representing meaning and performing reasoning on it. Soar is a powerful tool for building complex knowledge structures and performing reasoning on them. It has been applied to many application areas, including robotics and language processing.

## **Soar and language**

Language processing using Soar was pioneered by Richard Lewis (1993). He built a system called NL-Soar which can parse sentences using methods inspired by psycholinguistic research on how humans do sentence processing.

Lonsdale and others have moved forward in this area by applying the Soar cognitive architecture to build the LG-Soar and XNL-Soar systems (Lonsdale et al., 2008). Melby (1995a, 1995b) has also shown the necessity of agency to be able to achieve machine understanding of natural language. Soar is a good candidate to fill the role of an agent for language understandin.

The LG-Soar system is of particular interest here since it uses the Link Grammar Parser along with a semantic interpreter developed inside Soar to extract meaning from input sentences. LG-Soar has been used for information extraction applications (Tustison, 2004) and in a robotics system that can learn new linguistic constructions (Mohan et al., 2102 and 2013). OntoSoar has been derived from this approach, but with an innovative form of semantic analyzer.

The use of Soar for this project has been motivated theoretically in part by the fact that Soar is intended to model human cognition (Newell, 1990) and by the importance of agency in understanding language (Melby, 1995a and 1995b). However, in this work we make no attempt to claim cognitive plausibility for the particular approach used to apply Soar to language.

### **3. Thesis Statement**

The primary hypothesis we hope to prove with this thesis is the following:

*We can use modern lexical, syntactic, and semantic analysis tools to develop an algorithm that extracts information from genealogy texts and matches that data to a conceptual model of the family history domain provided by a user so as to populate that model with facts found in the text.*

## 4. Method

This chapter presents a description of the tools and algorithms used to make OntoSoar work. First we address what is needed for a couple of specific examples, then we look at the overall architecture of OntoSoar, and finally we discuss in some detail each major component of the system.

Before digging into the details, we must consider what kind of information we're looking for. For the purposes of this thesis we will limit ourselves to the basics of genealogical data: identifying unique individual persons along with their names, gender, birth and death dates, and direct family relationships such as marriages and parent/child relationships. We will not try to deal with places or with other life information such as employment or religion. We will also make some simplifying assumptions about family relationships, such as that a marriage is between a man and a woman, and that parent/child relationships are only for biological parents. These limitations and assumptions can be relaxed in future work.

### Examples

In the Introduction we stated that automatically extracting information from family history books depends on using higher-level semantic, syntactic, and world knowledge. In this section we present an informal analysis of some examples to get an intuitive idea of what knowledge might be needed and how it could be applied. We will consider two examples, one taken from Sample 1 (called CCL) and one from Sample 2 (called Myra).

To begin, consider the first sentence fragment in Sample 1 (CCL), which looks like this:

- (1) Charles Christopher Lathrop, N. Y. City, b. 1817, d. 1865, son of Mary Ely and Gerard Lathrop ;

This is not quite normal English. In order to meet normal rules for written grammar, we would need to paraphrase it somehow, perhaps like this:

- (2) Charles Christopher Lathrop, who lived in N. Y. City, was born in 1817, died in 1865, and was a son of Mary Ely and Gerard Lathrop.

The system to be discussed here will not do any paraphrasing of this sort, but it will need to somehow interpret text in a form like (1) to produce the same results as if it had been written in a form like (2).

As an English-speaking human being looking at the fragment in (1), what information can we extract? First we easily see that it is dealing with a person whose name is Charles Christopher Lathrop. He lived in New York City, but it is not clear in what part of his life this was true. Assuming that we can infer b. as meaning *was born in* and d. as meaning *died in*, we can derive that he was born in 1817 and died in 1865. There is a couple, presumably married, whose names are Mary Ely and Gerard Lathrop, and our primary person is their son.

We might conclude that Mary Ely is the mother and that Gerard Lathrop is the father, but how could a computer system know this? There are a couple of possibilities: use lexical knowledge or inference with pragmatic and world knowledge. A dictionary of first names could determine that Mary is almost certainly a woman's name and Gerard is very likely a man's name. The

assumption that a set of parents must include a man and a woman could help deduce one gender if the other is known.

In the absence of any name dictionaries, or if the names we're working with are not in the dictionaries, we could do some inferencing with pragmatic and world knowledge. If we know how surnames are passed down and used in the English-speaking world, the fact that Charles Christopher Lathrop and Gerard Lathrop have the same last name while Mary Ely's last name is different allows us to deduce that Gerard Lathrop is the father and Mary Ely is the mother. These parents are also additional individuals to add to our database, even though at the moment we have no more information about them.

Thus far from (1) we have identified three individuals along with birth and death dates for one of them and some family relationships. We can represent this information graphically, as in Figure 3.

Charles Christopher Lathrop, N. Y. City, b. 1817, d. 1865, son of Mary Ely and Gerard Lathrop ;

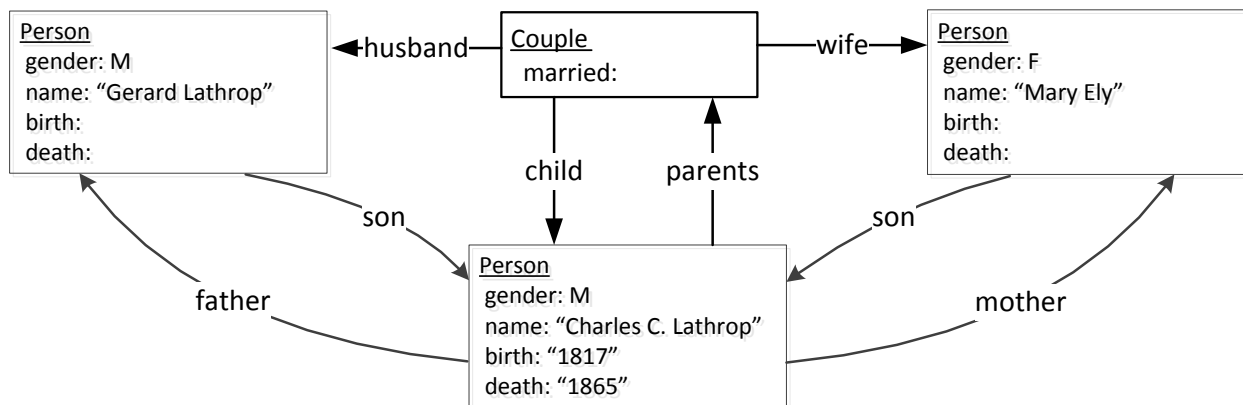


Figure 3: Meanings Derived from CCL Example

This diagram represents the meaning of fragment (1), at least as far as we have analyzed it so far. The diagram leaves open slots for additional information that is not now available but might be discovered later on.

How did we know that the names Charles Christopher Lathrop (abbreviated somewhat in Figure 3), Mary Ely, and Gerard Lathrop identify persons? Two kinds of knowledge might help. Lexical knowledge can be applied to the names themselves, either by looking up the words in dictionaries as mentioned or by using a much more sophisticated *named entity recognizer* of some sort.

There is another approach, however, using syntactic and semantic knowledge. An English syntactic parser can identify proper names, but not whether they represent a person, a place, an organization, or something else. Further syntactic knowledge can determine that a name is the subject or object of a verb like *born*, *died*, or *married*, or of some other predicate like *son of*. The semantics of these predicates plus the fact that we are working in the domain of human genealogy allow us to conclude these names refer to persons. Similar reasoning with a relation like *son of* can determine the gender of the subject.

The fragment in (3), taken from Sample 2 (Myra), gives a more complex example.

- (3) his widow married JONATHAN SQUIRES, who was born in Ohio, July 25, 1823, by whom she had one son, J. Wilbur, born June 16, 1865, in DeKalb county, Ind.

This passage is a lot more challenging. The language here is much more complex, but we consider what processing is possible.

This starts with the noun phrase *his widow*, which we cannot resolve from this fragment itself without wider discourse information. In (4) we see selections from the previous text of the paragraph:

- (4) MYRA, born July 26, 1835, in Eden, Vt. She married ELIJAH SPENCER, Dec. 25, 1851. ... Elijah Spencer died in the Union army in 1863, and his widow married ... .

Several reasoning steps will show that *his* likely refers to Elijah Spencer since he is the salient male at that point, that the noun *widow* refers to a woman whose husband has died, and that the *she* in *She married* refers back to MYRA. Since Myra married Elijah Spencer and Elijah Spencer died, his widow must therefore be Myra. The heading at the top of Sample 2 shows that we are discussing the children of James Harwood, so that Myra's maiden name must be Myra Harwood, again using the rules of surname inheritance in English.

Returning to our text in (3), we now know that it was Myra Harwood who married Jonathan Squires. Ignoring for the moment the details of Jonathan's birth presented here, we now skip to the part that says: *by whom she had one son, J. Wilbur, born June 16, 1865.* This gives us an individual named J. Wilbur who was born on June 16, 1865.

Notice, however, that the family relationships are described here by much different language than the *son of A and B* form we saw in (1). Instead we have the phrase *by whom she had one son*. To understand this we have to



identify the antecedents of the pronouns *whom* and *she*, the preposition *by*, and the verb *had* when its subject is a woman and its object is *one son*. This requires using lexical, syntactic, semantic, and pragmatic knowledge.

The result of this process for the fragment in (3), augmented by the contextual information that precedes it, will be a meaning diagram like this:

his widow married JONATHAN SQUIRES, ..., by whom she had one son, J. Wilbur, born June 16, 1865, ... .

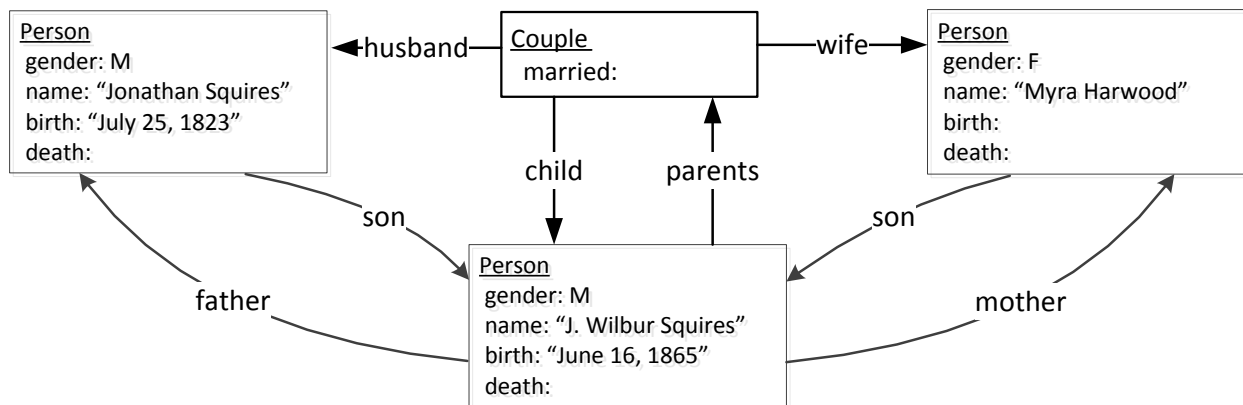


Figure 4: Meanings Derived from Myra Example

The surface linguistic form of the fragments in (1) and (3) is quite different, yet the meaning structures in Figures 3 and 4 are exactly the same except for the names and dates. This shows how human language can employ a wide range of forms to represent any given idea or set of ideas. The one used in a particular situation depends on the context of the discourse and the goals of the speaker or author. This is why natural language understanding is hard.

One key to a possible solution is the fact that we are working in a very limited, well understood domain<sup>7</sup>. Within this domain there are many simplifying assumptions we can make, such as that the subject of the verb

<sup>7</sup> See Melby (1995) for comments on domain-specific vs. general language processing.

*born* is going to be a person and not either a giraffe or a nation. The constraints of this domain make it feasible to think we could assemble enough textual, lexical, syntactic, semantic, pragmatic, and world knowledge to do a reasonably good job of extracting the information we want from the family history book texts. The OntoSoar project draws on much previous work and adds some original contributions to produce a system that looks promising for this specific problem within the much larger field of extracting information from text.

### **Target ontologies**

Up to this point we have considered a couple of specific examples of the input texts we plan to deal with, and given some intuitive ideas of how we might approach the problem of extracting useful information from them. We have not yet considered how that information can be represented in a form that would allow it to be inserted into a database where it could be searched and queried by users. We now address this question.

The *Onto* part of OntoSoar is short for OntoES, a system which has been under development by the Data Extraction Group (DEG) at BYU for several years. In part OntoES draws on a large body of literature on conceptual modeling to produce a model called OSMX capable of representing a wide variety of conceptual models and populating them with data (see Embley et al., 1992). OntoES includes tools for creating and manipulating these models graphically. A given conceptual model represented in the OSMX form, with or without being populated with facts, we will refer to as an *ontology*.

OntoSoar, then, in addition to the family history book text inputs we have been discussing, has another type of input: an OSMX ontology representing the conceptual model that the user wants the extracted information to be mapped onto. Thus OntoSoar reads in two files, a text file and an OSMX ontology file, does all the reasoning necessary to derive the meaning of the text, and then maps its internal meaning representation onto the user-provided ontology. The end result is to write out a new OSMX file in which the ontology is populated by all the facts derived from the text. This populated ontology can then be added to a searchable database to make the information available to anyone who wants to search it.

Figure 5 gives a simple example of what an ontology for genealogy information might look like as represented graphically by the OntoES tools.

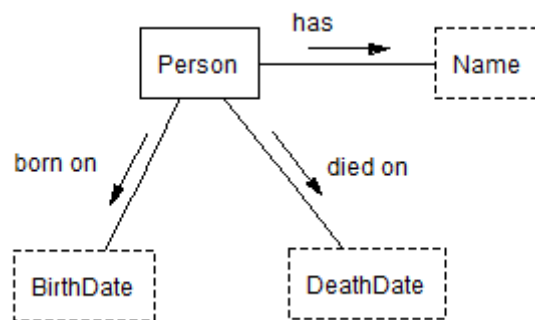


Figure 5: Ontology Example 1

This model is very simple, designed only to identify people by their names and represent their birth and death dates. A more complex example that shows one possible way of modeling family relationships is given in Figure 6.

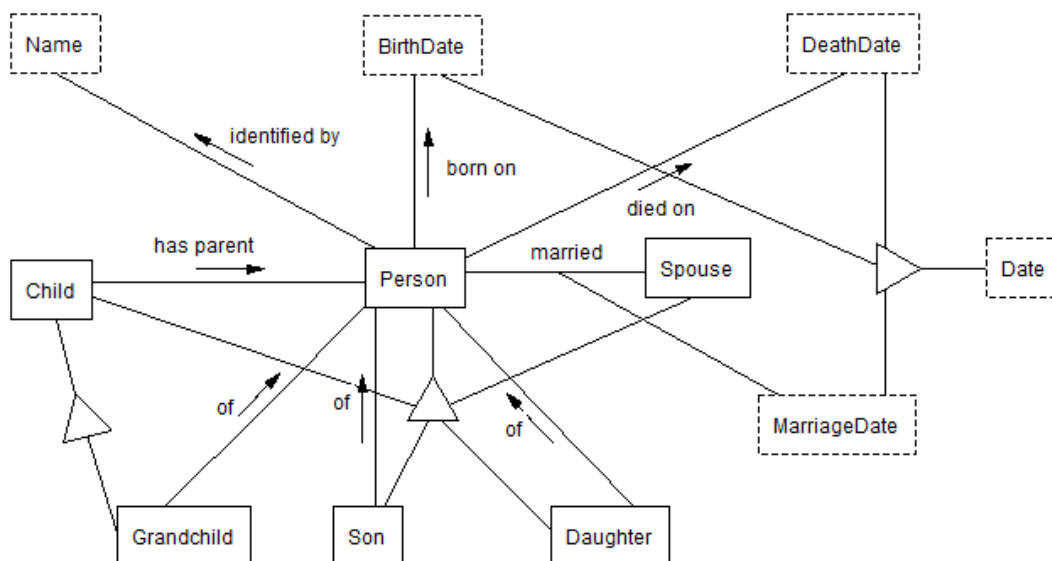


Figure 6: Ontology Example 2

This ontology, or other similar ones, has often been used in other work done with the OntoES system. It represents family relationships by using subsets of the Person class. Other representations where family relationships are first class objects in themselves are also possible. The goal of this thesis includes showing how to map to a number of possible ontologies of this sort.

### Levels of knowledge

Applying Allen Newell's insight quoted in Chapter 1 to our goals here, we see that several levels of knowledge can be brought to bear: textual, syntactic, lexical, semantic, pragmatic, and world knowledge. We now look briefly at each of these.

There are many possibilities for using knowledge at the textual level. First of all, it should not be difficult to make rules to correct some of the OCR errors that appear in the raw text we get from scanned historical documents.

For example, in our sample texts we can correct `i860` to `1860` and `Nov .`  
`2,1879` to `Nov 2, 1879`. We can also divide the text into tokens and categorize tokens as words, numbers, or punctuation. A preprocessor could identify phrases that are likely to be names of people, places, organizations, etc. either by simply looking up words in lexicons or by using one of the available named entity recognition systems. Another key element at the textual level is knowing how to break up the running text into segments that correspond roughly to sentences so that these segments can be processed reasonably by the syntactic part of the system.

Lexical knowledge consists of knowing about individual words, which could include their spelling, their pronunciation, their parts of speech, and what they mean.

Syntactic analysis usually consists of both identifying the part of speech of each word and building parse trees of the syntactic structure of each sentence. This type of analysis usually finds constituents like noun phrases, verbs with their subjects and objects, and other kinds of modifying phrases and clauses. This kind of knowledge is very useful in understanding the structure of the language in the text and how different words and phrases relate to each other structurally. However, it gives very little information on what a segment of text actually means.

Semantics involves using both syntactic and lexical knowledge to derive a representation of the meaning of a given linguistic unit. The literature on formal semantics attempts to represent meaning in terms of mathematical

models that abstract away from the mental processes of a speaker or writer and a hearer or reader. Cognitive semantics tries to understand meaning in terms of how it relates to experience in the minds of real human beings.

Often semantics, when used as a technical term, deals with the meaning of one particular sentence at a time. However, much of meaning comes from the ongoing discourse made up of many sentences and from knowledge of the world in general or the particular domain or situation being discussed independent of anything in the language itself. This level of knowledge is generally called pragmatics, and it is necessary to find what entities pronouns and other noun phrases that are not proper names refer to. For example, in (3) we need to consider the knowledge from previous sentences to know who the phrase `his widow` refers to.

The system built for this thesis uses tools at all these levels of knowledge. Incoming texts will first be processed by a textual preprocessor which will segment the text into sentence-like fragments that the syntactic parser can handle, as well as correcting as many OCR errors as possible and replacing many abbreviations for key words, such as replacing `b.` with `born` and `d.` with `died`. Syntactic analysis is done by the open source Link Grammar Parser, with its grammar modified somewhat to deal with the idiosyncrasies of the text found in genealogy books. Both semantic and pragmatic knowledge are applied by a meaning engine built using the Soar cognitive architecture, which will also map the meanings found from the text onto the conceptual model provided by a user ontology for a particular domain. This meaning engine, as

well as the programmatic glue needed to make all these elements work together smoothly, is the main contribution of this work.

### System architecture

OntoSoar is built using Java components, some Java libraries, some custom Java components, the LG parser, the Soar system, and much Soar code that implements all the semantic components. Figure 7 shows a block diagram of the system, showing the main flow of data. In addition to the blocks shown, the overall Java application manages the flow of data through the system and the interactions between Soar and the rest of the world.

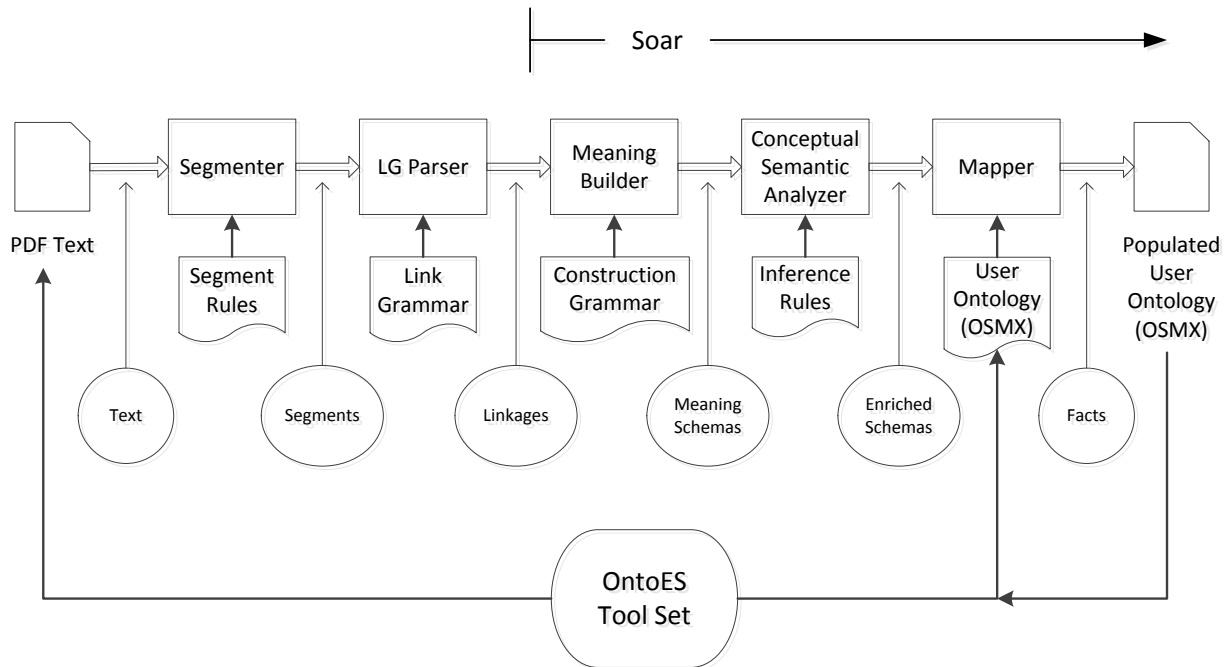


Figure 7: OntoSoar Block Diagram

Figure 7 shows a pipeline where the raw text extracted from a PDF file by an OCR engine enters at the left and the data is transformed by several components to get to the form called *Enriched Schemas* in the figure. At this

point the *Mapper* component takes a conceptual model in the form of an OSMX file, populates it with facts derived from the internal meaning structures, and outputs the populated ontology as a new OSMX file. This output file can then be viewed, evaluated, or imported into a database by tools from the OntoES tool set.

The Soar components called Meaning Builder, Conceptual Semantic Analyzer, and Mapper will be often referred to collectively as the Semantic Analyzer in what follows.

### **Segmentation**

The block called *Segmenter* in Figure 7 is actually a preprocessor that does several kinds of text processing to transform the raw OCR'd text into a form that the LG Parser can work with. The LG Parser and the rest of the pipeline process the text one *segment* at a time. A segment is roughly equivalent to a short sentence. However, the input text often has several clauses run together into a much longer sentence. The LG Parser tends to get very confused and produce bad results when it gets several clauses run together, so the main job of the Segmenter is to break the text up into sentence-like segments that can be processed well by the parser. It also makes some corrections at the individual token level to reduce OCR errors and similar anomalies.

The Segmenter starts by combining the entire input text into a single string with all groups of white space characters condensed into single spaces, then splitting this string into tokens based on those spaces. It then makes



some corrections at the token level: it makes sure there is a space after every comma (sometimes the OCR deletes a space after a comma), it changes ‘i’ to ‘1’ in a sequence of digits (another common OCR error), and replaces a string like Rosa E., with Rosa E, since the period and comma together greatly confuse the LG Parser. In addition, tokens that represent abbreviations commonly used in family history texts are replaced with the full word they represent, such as born for b., died for d., daughter for dau., etc.

Once the tokens have been cleaned up, the preprocessor proceeds to divide the text into segments by marking each token according to whether or not it should be the end of a segment. This marking is done by comparing each token against all the rules in a file of segment rules. Each rule specifies a pattern to be matched and whether to mark a token that matches that pattern as an end-of-segment marker or not. Basically the rules say that any token that ends with a period, a colon, or a semicolon should be considered a segment marker. However, there are a number of rules to recognize abbreviations that are frequently used in domain texts and not mark the end of a segment based on the period in those abbreviations.

It was found that using these rules based on punctuation did a reasonable job, but we still often had many segments that were too long and confused the parser. Solving this problem required being smarter about commas. Many commas should not end a segment, but also the texts include many commas that really do separate different clauses. So a heuristic that works reasonably well was added: whenever a comma is followed by one of a

list of specific words, break the segment at that point and replace that comma with a period. The words used to indicate a new segment should begin are: and, who, by, Mr., Mrs., Miss, he, she, they, had, have, **and** married.

Generally speaking these words indicate a new clause. However, the last three are verbs whose subjects will have been left behind in the previous segment that was broken off. This problem is solved by inserting the token *GP*, standing for *Generic Pronoun*, in front of the verb to start the new segment. Thus when the system resolves the reference for this pronoun the verb will connect with its subject again. The examples below will show how this works.

The following two figures show the results of the segmenter for each of our two sample texts, which we will refer to by short names for their principal characters as *CCL* for Sample 1 and *Myra* for Sample 2.

The segmented text for the *CCL* sample is given in (5):

- (5) 1: 243314. '.'  
2: Charles Christopher Lathrop, N. Y. City, born 1817, died 1865, son of Mary Ely and Gerard Lathrop ; ';' '  
3: GP married 1856, Mary Augusta Andruss, 992 Broad St., Newark, N. J. ', '  
4: who was born 1825, daughter of Judge Caleb Halstead Andruss and Emma Sutherland Goble. '.' '  
5: Mrs. Lathrop died at her home, 992 Broad St., Newark, N. J, Friday morning, Nov. 4, 1898. '.' '  
6: The funeral services were held at her residence on Monday, Nov. 7, 1898, at half-past two o'clock P. M. Their children: ':' '  
7: 1. '.' '  
8: Charles Halstead, born 1857, died 1861. '.' '  
9: 2. '.' '  
10: William Gerard, born 1858, died 1861. '.' '  
11: 3. '.' '  
12: Theodore Andruss, born 1860. '.' '  
13: 4. '.' '  
14: Emma Goble, born 1862. '.' '  
15: Miss Emma Goble Lathrop, official historian of the New York Chapter of the Daughters of the American Revolution, is one of the youngest members to hold office, but one whose intelligence and capability qualify her for such distinction. '.' '

An image of this part of the original PDF file is in Figure 1. Figure 2 has an image of the Myra sample, whose segmented form is given in (6).

- (6) 1: Children of JAMES HARWOOD, NO. 103. '.'  
2: 229. '.'  
3: MYRA, born July 26, 1835, in Eden, Vt. '.'  
4: She married ELIJAH SPENCER, Dec. 25, 1851. '.'  
5: They had five children: ':'  
6: Arvilla, born in 1852, is not living; ';'   
7: Mariette, born Dec. 25, 1854. ','  
8: GP married Jonathan Snyder. ','  
9: GP have a family; ';'   
10: Leverett, born Feb. 6, 1857. ','  
11: GP married Cora Smith, Nov. 2, 1879. ','  
12: GP had two children, Perry F. and Ida I. Leverett died May 21, 1910; ';'   
13: Rosa E, born Jan. 13, 1860. ','  
14: GP married Emmett Byers. ','  
15: and have children; ';'   
16: and Harrison, born about 1862, is not living. '.'  
17: Elijah Spencer died in the Union army in 1863. ','  
18: and his widow married JONATHAN SQUIRES. ','  
19: who was born in Ohio, July 25, 1823. ','  
20: by whom she had one son, J. Wilbur, born June 16, 1865, in DeKalb county, Ind.. ','  
21: GP married Cora M. Thomas, Aug. 24, 1887. ','  
22: they reside in St. Joseph, Mich., five children. ','  
23: Mrs. Myra Squires died in Allen county, Ind., Feb. 13, 1874. '.'

These printouts show three parts for each segment. First there is a segment number followed by a colon, then the actual text of the segment as it will be submitted to the parser, and finally a single punctuation mark in single quotes. This punctuation mark is the one that was originally at the end of the segment before the algorithm put a period at the end to help the parser. Later semantic analysis will need to know this original terminator because it affects pronoun resolution.

We can see that most segments now have just one or two verbs, which the parser can handle well. We also see several segments that begin with a pronoun, especially the synthetic pronoun GP. Later we will see that resolving the referents for these pronouns is important for overall system performance.

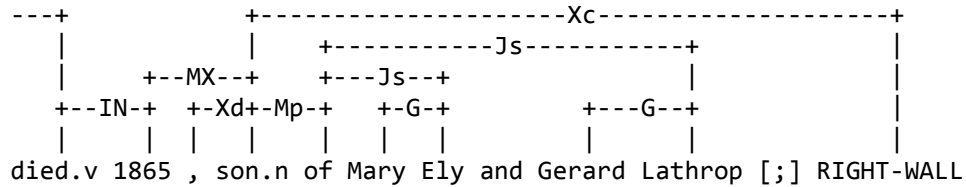
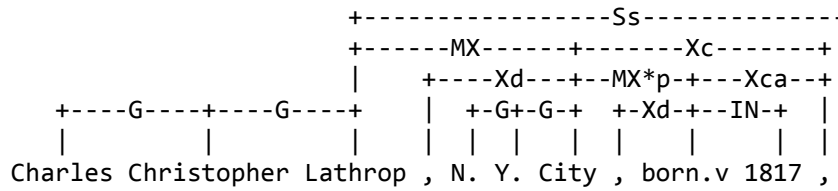
## Parsing

The Link Grammar Parser provides our syntactic analysis component, and runs as a black box to take in the text of one segment at a time and produce a parse result called a *linkage* for that segment. One of the great advantages of the LG Parser is that its grammar is accessible and easy to modify. As mentioned earlier and shown in our text samples, family history books are often written in a much abbreviated English style. Many function words are omitted completely, causing a parser that only works with standard English grammar to fail. We have modified the grammar in several small details so that it works well on our texts.

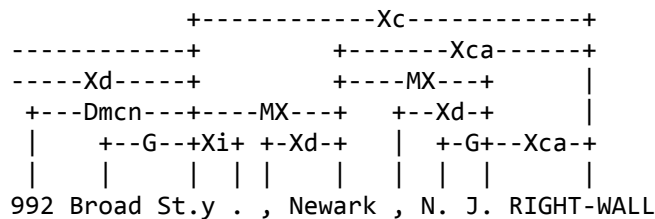
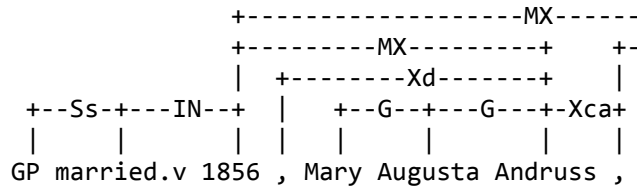
In (7) we see several examples of linkages produced by the CCL text. Some are wrapped across multiple lines in this thesis format. Each link between words is marked with a primary type in upper case and sometimes a secondary type in lower case. Some of the meanings of the main link types are: *S* subject of a verb, *O* object of a verb, *G* proper noun, *J* object of a preposition, *IN* date, *MX* appositive, and *X* punctuation.

If we look at (7a), we see that the verb *born* is attached with an *MX* link to *N. Y. City*, not to its real subject *Charles Christopher Lathrop*. The semantic processor deals with this by seeing that there is a second *MX* link which does connect to the real subject, and assuming that an appositive modifying another appositive should really modify the same thing as the first appositive. Without this we would get *N. Y. City* being born in 1817.

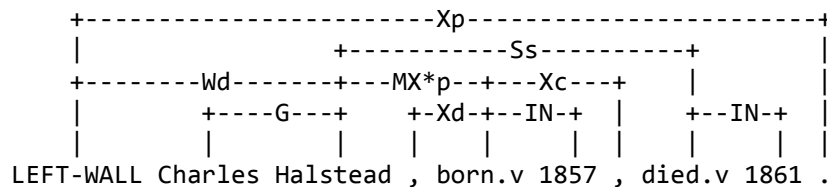
- (7) a. 2: Charles Christopher Lathrop, N. Y. City, born 1817, died 1865, son of Mary Ely and Gerard Lathrop ; ' ;'



- b. 3: GP married 1856, Mary Augusta Andruss, 992 Broad St., Newark, N. J. ','



- c. 8: Charles Halstead, born 1857, died 1861. '.'



In (8) shows some linkages from the Myra text. In (8d) we see a limitation of our segmentation algorithm. When it sees an abbreviation like I. it knows this is an abbreviation, but it has no way of knowing if that period might also indicate the end of a sentence. This is an ambiguity in English

orthography. The Segmenter assumes this is not the end of a sentence, which is the correct choice most of the time. In this particular case, however, it is the wrong answer and causes the second Leverett to get an incorrect name.

(8) a. 3: MYRA, born July 26, 1835, in Eden, Vt. '.'

```

+-----MX*x-----+
|           +-----Xca-----+ +-----Xc-----+
+--MX*p--+   +----TY----+ | | +--MX--+         |
+---Wf---+ +Xd+---IN--+TM+ +Xd+Xc+Xd+-Js+ +Xd+---Xca---+
|           | |           | | | | | | | | | | | | | |
LEFT-WALL MYRA , born.v July 26 , 1835 , in Eden , Vt. RIGHT-WALL

```

b. 4: She married ELIJAH SPENCER, Dec. 25, 1851. '.'

```

+-----Mvp-----+
+-----Os-----+ | +-----TY-----+
+--Ss---+         +---G---+ +--IN+---TM---+ +Xd+Xc+
|           |         | |           | | | | | | | | |
she married.v ELIJAH SPENCER , Dec.x [.] 25 , 1851 .

```

c. 6: Arvilla, born in 1852, is not living; ';'

```

+-----Ss-----+
+---MX*p---+-----Xc-----+ +----Ost----+
|           +Xd+---Mvp+---IN+ | +EBm+         |
|           | |           | | | | | | | | | |
Arvilla , born.v in 1852 , is.v not living.n [;]

```

d. 12: GP had two children, Perry F. and Ida I. Leverett died May 21, 1910; ';'

```

+-----Op-----+---MXp-
+--Ss+   +---Dmc---+   +-
| |     | |         | |
GP had.v two children.n ,

```

```

+-----Xc-----+
---+           +-----TY-----+ |
Xd+   +-----G-----+---G---+---Ss---+---IN---+---TM---+ +Xd+---Xca---+
| |   | |           | | | | | | | | | | | | | | | | | |
Perry F. and Ida I. Leverett died.v May.i 21 , 1910 [;] RIGHT-WALL

```

e. 17: Elijah Spencer died in the Union army in 1863. ','

```

+-----Xp-----+
|               +-----Mvp-----+
|               |   +-----Js-----+
+-----Wd-----+ |   |   +-----Ds-----+
|               |   |   |   +---AN---+   +---IN+
|               |   |   |   |   |   |   |
LEFT-WALL Elijah Spencer died.v in the Union army.n in 1863 .

```

f. 18: and his widow married JONATHAN SQUIRES. ','

```

+-----Xp-----+
|   +---Wdc---+   +-----Os-----+
+---Wc---+   +---Ds---+---Ss---+   +---G---+
|   |   |   |   |   |   |   |
LEFT-WALL and his widow.n married.v JONATHAN SQUIRES .

```

g. 19: who was born in Ohio, July 25, 1823. ','

```

+-----Mvp-----+   +-----TY-----+
+---WS---+Ss*w+---Pv---+Mvp+-Js+   +-IN+-TM+ +-Xd+Xc+
|   |   |   |   |   |   |   |   |   |
LEFT-WALL who was.v born.v in Ohio , July 25 , 1823 .

```

h. 20: by whom she had one son, J. Wilbur, born June 16, 1865, in DeKalb county, Ind.. ','

```

+-----Mvp-----+
|               +-----MXsp-----+
|               |   +-----MXs-----+   +-----Xc-----+
+---CO---+   +---Os---+   +---Xd---+   |   +-----TY-----+
+---Jw+   +-Ss-+   +-Ds-+   |   +---G---+Xca+-Xd---+---IN---+---TM+ +-Xd+Xc+
|   |   |   |   |   |   |   |   |   |   |   |
by whom she had.v one son.n , J. Wilbur , born.v June 16 , 1865 ,

```

```

-+
|
|
+-----Js-----+---MXs---+
|   +---AN---+   +-Xd+Xc+
|   |   |   |   |
in DeKalb county.n , Ind. .

```

i. 21: GP married Cora M. Thomas, Aug. 24, 1887. ','

```

+-----Mvp-----+
+-----O-----+   |   +-----TY-----+
+---Ss---+   +---G---+---G---+   +-IN+---TM---+ +-Xd+Xc+
|   |   |   |   |   |   |   |   |
GP married.v Cora M. Thomas , Aug.x [.] 24 , 1887 .

```

j. 22: they reside in St. Joseph, Mich., five children. ','

```

+-----Js-----+-----MX-----+
| +---G---+---MX---+ +-----Xd-----+
+--Sp--+--MVP--+ +Xi+ | +-Xd+Xca+ +---Dmc--+---Xc--+
| | | | | | | | | | | | | | | |
they reside.v in St.x . Joseph , Mich. , five children.n .

```

k. 23: Mrs. Myra Squires died in Allen county, Ind., Feb. 13, 1874. '.'

```

+-----IN-----
+---G---+ | +-----Js-----+---MXs---+
+-Xi+ +---G---+---S---+---MVP+ +---AN---+ +-Xd+Xc+
| | | | | | | | | | | | | | | |
Mrs.x . Myra Squires died.v in Allen county.n , Ind. ,

```

```

--+
+-----TY-----+
+---TM---+ +-Xd+Xc+
| | | | |
Feb.x [.] 13 , 1874 .

```

Segments (8e-k) show a major advantage of the current segmentation algorithm. With the original OCR'd text, a simple segmenter that only breaks on a period that's not in an abbreviation would keep all seven of these segments as one huge sentence. This is true even though if we look at Figure 2 we see a clear period at the end of what we are calling segment Myra 22 separating it from Myra 23. However, the OCR engine interpreted that period as a comma. When something like that is fed into the LG Parser it takes an enormous amount of time to run and produces a linkage with a lot of mistakes in it.

### Building meanings

Figure 7 shows that the output of the LG Parser going into Soar, where three components eventually produce a set of facts in the user ontology. These facts are output to the Java code, which puts them into a populated OSMX file.



The first of these three components is called the Meaning Builder, which we will discuss here.

Conceptually this component is based on the Embodied Construction Grammar (ECG) ideas discussed by Bergen and Chang (2003, 2013), Bryant (2008), and Chang (2009). They present both an intuitive explanation and a formalism for ECG. Although these ideas have inspired the work done here, there are two fundamental differences.

First, construction grammar in general and ECG in particular are designed to build constructions directly from an input text. However, in OntoSoar we are building constructions from the linkages produced by the LG Parser. Thus we have available not only the words themselves but also the links between them found by the parser.

Second, Bryant (2008) presents a formal grammar for ECG, and his system includes a compiler to compile a grammar written in this ECG language into an internal form. The construction grammar in OntoSoar, which we will call OCG, has been coded by hand into Soar productions. Some of these simply build static data structures when the program initializes itself and thus can be thought of as declarative knowledge, while others are productions that fire as the semantic analysis is proceeding and thus are procedural knowledge. We will see examples below. The knowledge and experience produced by the current project may enable a future effort to build a compiler to compile some form of OCG from a higher level representation into Soar code.

In the construction grammar paradigm, a *construction* is a structure that maps a part of the surface form of the input language into a meaning representation of some sort. In ECG and OCG the two ends of this mapping are called the *form pole* and the *meaning pole*. In OntoSoar the declarative part of the grammar created at system initialization contains descriptions of constructions that will be attached dynamically to parts of the input stream, as well as meaning schemas that these constructions map to.

To illustrate this concept, Figure 8 gives an example for a portion of the CCL 2 segment.

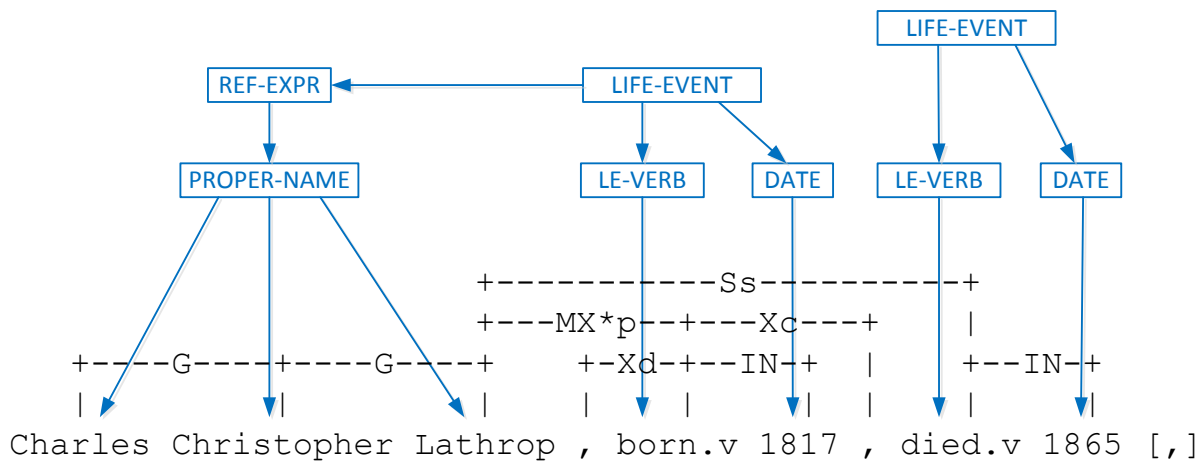


Figure 8: Construction Example 1

Here we see the linkage for this partial segment and a set of blue rectangles and arrows that represent the constructions recognized from this segment. The lower level rectangles have arrows pointing to the words that make up the form pole of each of those constructions. The drawing somewhat simplifies the complexity of the full set of constructions. Though not shown, each of these constructions is recognized based not only on the words it

contains but also on the links from each word going toward its left. Only leftward links are considered because the Soar part of OntoSoar works incrementally one word at a time, as opposed to the LG Parser which considers a whole segment at once.

Figure 8 shows the form pole of each construction. However, every construction also has its meaning pole. In Figure 9 we see the same diagram with meaning structures added.

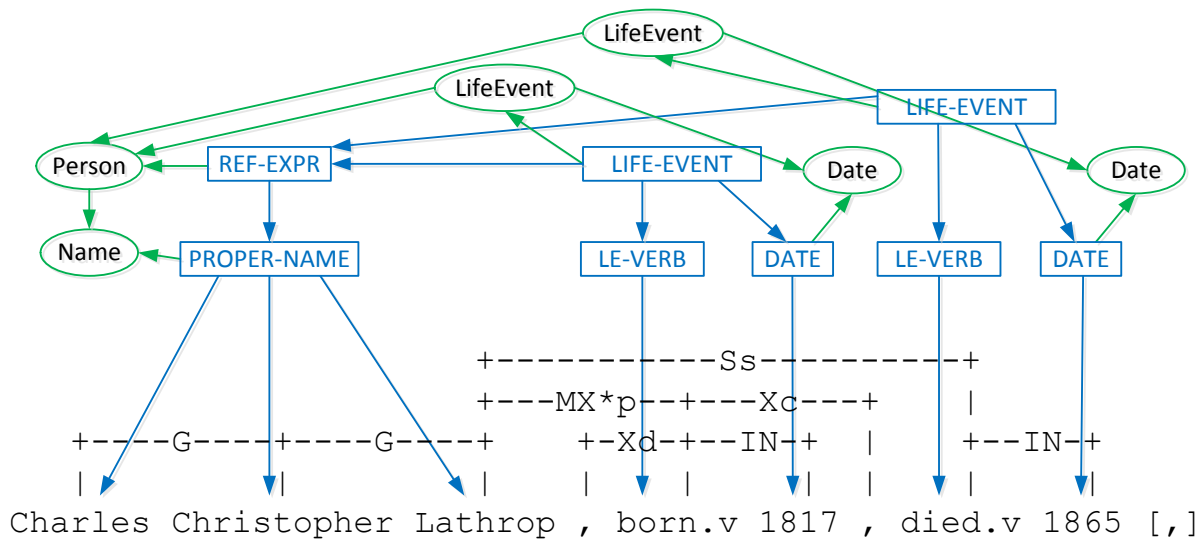


Figure 9: Construction Example 1 with Meanings

The root structures of this meaning network are the LifeEvent structures. Each requires a Person subject, and the Person is shown here with a Name. Date structures are also connected to each LifeEvent, but these are optional.

This drawing has a couple of simplifications of what the real meaning structures look like. First of all, each meaning structure has a number of internal slots to hold values of properties or references to other meaning structures. In ECG terminology these slots are called *roles*. For example, the

drawing shows a LifeEvent as having a subject role to be filled with a Person and a date role to be filled by a Date. A Person is shown as having a name role filled by a Name, but it also has birth and death roles which point to LifeEvents, if filled.

Another major simplification in this drawing has to do with referring expressions. These include proper nouns, pronouns, and other noun phrases that refer to an entity of some sort. The current OntoSoar only considers Person entities, but the structure is there to handle places, organizations, and other entity types.

The main construction for a referring expression is called REF-EXPR. It must have a single child which is some more specific type of expression. The only one shown here is a PROPER-NAME. However, the primary meaning structure associated with a REF-EXPR is something called a RefDesc (short for Referent Descriptor). A RefDesc has a number of roles to keep track of things like the number and gender of the referent, as well as a role called referent which points to the meaning structure for the actual entity referred to. The RefDesc structures are not shown in these drawings just to keep the drawing from being too cluttered. Instead we show the meaning structure for the entity referred to, which can be thought of as a merger of a Person and a RefDesc.

The purpose of the RefDesc structures is to allow for several referring expressions to refer to the same entity. For example, in Figure 9 we see an example of where the REF-EXPR is a PROPER-NAME, in which case the referent of the RefDesc is a Person structure created right there. However, in

segment CCL 3 we have a GP pronoun which should refer to this same person. This is accomplished by having a RefDesc based on that pronoun whose referent will eventually be filled in as being the Person built from Charles Christopher Lathrop. Thus every REF-EXPR construction has its own unique RefDesc structure, but several RefDescs may point to the same referent.

Another example extracted from the same CCL 2 segment is shown in Figure 10.

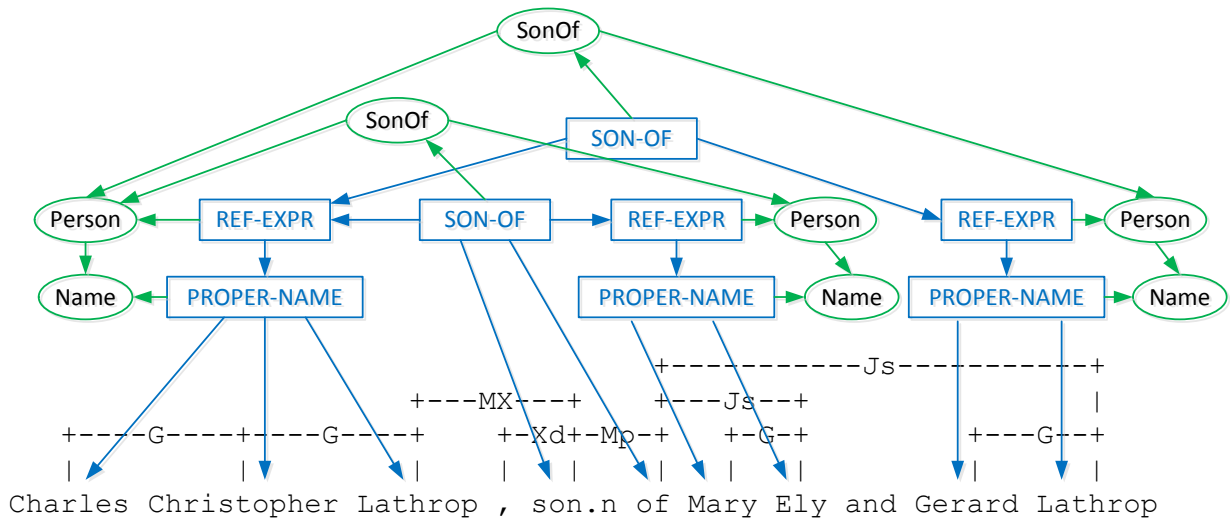


Figure 10: Construction Example 2 with Meanings

Here we see three Persons built from three PROPER-NAMEs. We also see a different kind of predicate. The predicates in Figures 8 and 9 were built from LIFE-EVENT constructions built from verbs. Here we have SonOf relations built from SON-OF constructions built from the noun *son*, a noun which represents a relationship. Of course many other relations of this sort based on nouns are possible, including one based on the phrase *his widow* which we see in the Myra 18 segment.

Time and space do not permit going into all the details of how meaning structures are built, but we can show some representative pieces. In (9) and (10) we will show both the construction and the meaning schema for proper names. These are shown in a format that is modeled after the ECG formalism of Bryant (2008), but adapted somewhat for OntoSoar. The current version of OntoSoar does not actually use this notation directly, but the structures shown in (9) and (10), and similar structures for other constructions and schemas, have been hand-coded as Soar productions.

```
(9) construction PROPER-NAME-CXN
    subcase of REF-EXPR
    constituents
      w : WORD
      pn : PROPER-NAME-CXN
    form
      constraints
      (1)   pn -G- w
      (2)   w
    meaning : ProperName
      constraints
      (1)   self.m.value <- concat(pn.value, w.text)
      (2)   self.m.value <- w.text
```

This construction shows that it is a subcase of REF-EXPR and that its meaning pole is a ProperName schema. This declarative knowledge is used by a `proper-name` Soar operator that has the procedural knowledge necessary to build an instance of a PROPER-NAME from one or more unknown words connected by G links. Then the ProperName *schema* is used to build ProperName meaning structures (called simply Names in our drawings). The schema is shown in (10).

```

(10) schema ProperName
      subcase of RefDesc
      roles
          value : string
          gender : { M | F | N }
          number : { S | P }
          person : { 1 | 2 | 3 }
          case : { N | D | P }
          givenness : { NAMED | ANAPHOR }
      matching
      keywords : name
      lexical : true

```

Here we see that a ProperName is a subcase of RefDesc. It has a role unique to ProperName called value, and also the gender, number, person, case, and givenness roles that are used to fill in the corresponding roles in RefDesc. These roles become very important later on in resolving pronouns and inferring gender.

To give a general idea of how the building of meanings is carried out, here is an abbreviated Soar trace for building the basic meaning structures for the phrase Charles Christopher Lathrop:

```

(11) 37:      O: 032 (comprehend-word)
      39:      O: 033 (setup-word)
      [Charles] G -> 2
      40:      O: 034 (lexical-construction)
      Building a WORD construction for 'Charles'.
      41:      O: 035 (word-done)
      Top of stack is WORD, nothing below it.
      42:      O: 036 (comprehend-word)
      1 -> G [Christopher] G -> 3
      45:      O: 038 (lexical-construction)
      Building a WORD construction for 'Christopher'.
      46:      O: 040 (proper-name)
      47:      O: 042 (build-meaning)
      49:      O: 043 (get-schema)
      Built a ProperName schema.
      50:      O: 044 (fill-defaults)
      51:      O: 046 (add-roles)
      Add roles to a ProperName schema.
      52:      O: 047 (fill-roles)
      53:      O: 048 (mark-ref-features)
      54:      O: 045 (meaning-done)

```

```

Attaching a ProperName schema to a PROPER-NAME construction.
 55:      O: 049 (generalize-cxn)
Generalizing a PROPER-NAME construction to a REF-EXPR construction.
 56:      O: 050 (build-meaning)
 58:      O: 051 (get-schema)
Built a RefDesc schema.
 59:      O: 052 (fill-defaults)
 60:      O: 054 (add-roles)
Add roles to a RefDesc schema.
 61:      O: 055 (fill-roles)
 62:      O: 053 (meaning-done)
Attaching a RefDesc schema to a REF-EXPR construction.
 63:      O: 039 (word-done)
Top of stack is REF-EXPR, nothing below it.
 64:      O: 056 (comprehend-word)
 65:      ==>S: S22 (operator no-change)
 66:      O: 057 (setup-word)
0 -> Wd 2 -> G [Lathrop] MXp -> 5 Ss -> 8
 67:      O: 058 (lexical-construction)
Building a WORD construction for 'Lathrop'.
 68:      O: 060 (proper-name)
 69:      O: 062 (build-meaning)
 71:      O: 063 (get-schema)
Built a ProperName schema.
 72:      O: 064 (fill-defaults)
 73:      O: 066 (add-roles)
Add roles to a ProperName schema.
 74:      O: 067 (fill-roles)
 75:      O: 068 (mark-ref-features)
 76:      O: 065 (meaning-done)
Attaching a ProperName schema to a PROPER-NAME construction.
 77:      O: 069 (generalize-cxn)
Generalizing a PROPER-NAME construction to a REF-EXPR construction.
 78:      O: 070 (build-meaning)
 80:      O: 071 (get-schema)
Built a RefDesc schema.
 81:      O: 072 (fill-defaults)
 82:      O: 074 (add-roles)
Add roles to a RefDesc schema.
 83:      O: 075 (fill-roles)
 84:      O: 073 (meaning-done)
Attaching a RefDesc schema to a REF-EXPR construction.
 85:      O: 059 (word-done)
Top of stack is REF-EXPR, nothing below it.

```

Here we can see that for each input word there is a comprehend-word operator. It in turn causes the build-meaning operator to execute. We also see a lexical-construction operator firing for each word, as well as proper-name operator and many others.



One key concept in the Meaning Builder is how entity types are determined. For example, in the CCL 2 segment, how do we know that Charles Christopher Lathrop is a Person and N. Y. City is not? The answer is that a RefDesc built from a ProperName does not have its category role filled until it is assigned as the subject or object of some predicate. We assume here that each predicate, such as *born* or *son of*, knows the types of its arguments. So when a RefDesc is assigned to an argument slot of a predicate its category (ie. its entity type) is assigned according to the type of that argument. This approach allows OntoSoar to know which proper names refer to people and which do not without having any kind of name dictionary or other sophisticated way of deriving entity types just from their names.

This should give something of the flavor of how the Meaning Builder works. In the end it builds a network of meaning structures with their roles, many of which are not yet filled. This network provides the basis for further semantic analysis.

### **Semantic analysis**

The next component in the pipeline shown in Figure 7 is called the Conceptual Semantic Analyzer. It takes the meaning structures supplied by the Meaning Builder and expands and enhances them using inference rules implemented as Soar productions. The best way to see how this works is with some examples.

Figure 11 shows how the meaning structures shown schematically in Figures 9 and 10 can be used to build a set of populated schemas like the ones we saw in Figures 3 and 4.

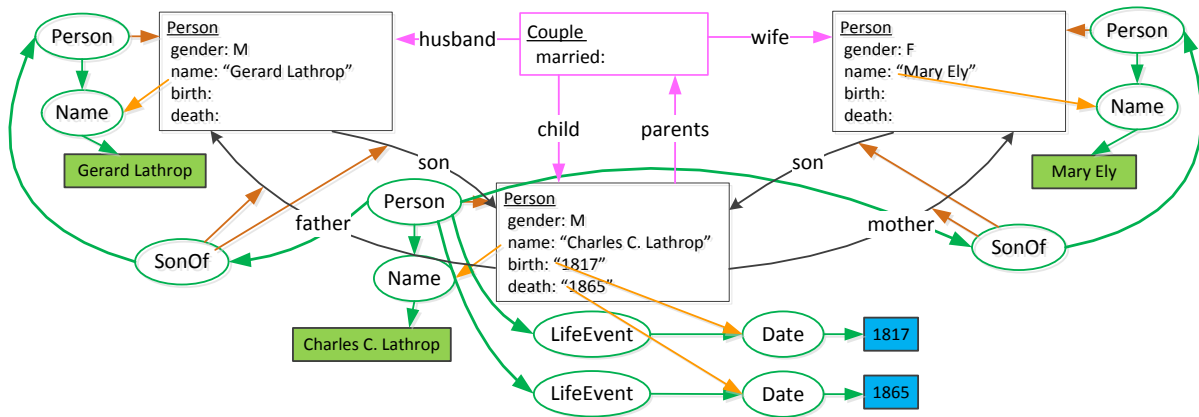


Figure 11: Semantic Analysis of CCL 2

Here we see lexical values for in green and blue boxes and light orange arrows showing how the different meaning structures connect to each other. The darker orange arrows show how the simplified Person shown in a green oval is actually a full structure with a number of internal roles.

A somewhat different view for the structures derived from a simplified version of Myra 18-20 is given in Figure 12. Here we see the input text segment, the parse, and the final meaning structures. The green boxes on the parse indicate referential expressions that are not proper nouns, and the pink boxes show predicates that can be used to derive information about some of the family relationships.

his widow married JONATHAN SQUIRES, ..., by whom she had one son, J. Wilbur, born June 16, 1865, ... .

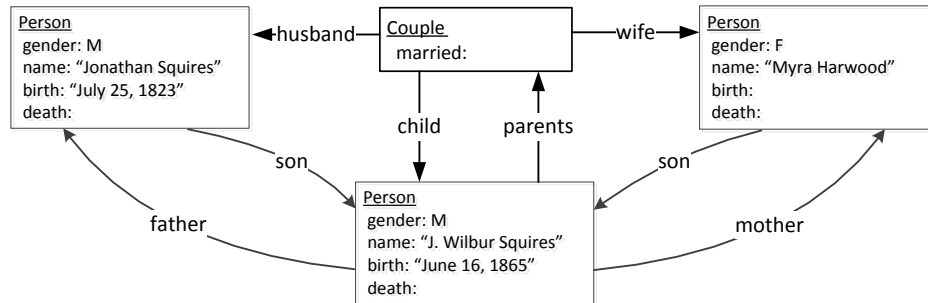
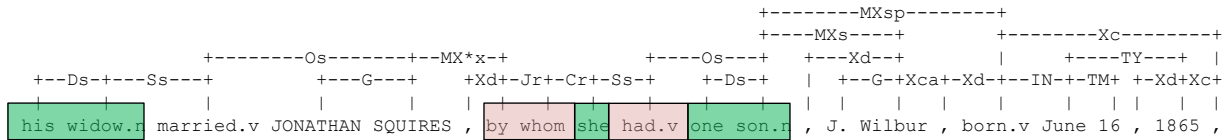


Figure 12: Meanings Derived from Myra 18-20

One feature of how the Semantic Analyzer works is not shown in these diagrams. When we have a phrase like *is not living* in the text we create a death event for the subject person, and also a Date schema with its value set to UNKNOWN. This allows us to distinguish between a situation in which the death is not reported at all from one in which it is reported without a date being specified. Similarly, for a segment like *Children of JAMES HARWOOD* in CCL 1, we can deduce that there is a couple with a partner for James, but we don't know the name or anything else about that second person. In this case we will report a second person whose name is UNKNOWN.

Another feature of the Semantic Analyzer is a reference resolver to find concrete referents for every RefDesc. When the RefDesc is a ProperName and we have determined that it fills an argument slot that needs a person, then we simply create a Person object for it. If it is a Pronoun or some other noun phrase we have to search backward in the context for an appropriate referent.

This is implemented with an operator called `resolve-reference`. At this time it works well enough to find the correct referent for a number of the cases in our two samples, but it still makes mistakes since it doesn't yet take advantage of gender and number agreement or the specific meanings of nouns like *widow*.

At this writing only part of the semantic analysis has been implemented, enough to produce the results we will see later on. Several more parts remain to be built or need more work. One of the most important is reference resolution, which is especially important because of the additional pronouns introduced by the segmentation procedure discussed above. As we shall see in the Results chapter, it works reasonably well but needs more improvement.

Nevertheless, the Semantic Analyzer is a key part of OntoSoar which can be built on in the future. It provides a structure within which it should be fairly easy to implement inference rules not only for reference resolution but also for such things as deducing surnames and finding cases of multiple names that refer to the same person. One example of the power of this approach is given in Figure 13, only partially implemented at present.

Here we see our Sample 2 or Myra text in its original form with several things overlaid on top of it. The green boxes are referring expressions, the blue boxes are dates, the yellow are life event verbs, and the pink are relationship phrases. The red dots and arrows show an inferencing chain that allows us to infer that the Mrs. Myra Squires, whose death is mentioned at the end of the paragraph, is the same person as the MYRA whose birth is at the beginning.

Children of JAMES HARWOOD, No. 103.

229. MYRA, born July 26, 1835 in Eden, Vt. She married ELIJAH SPENCER, Dec. 25, 1851. They had five children: Arvilla, born in 1852, is not living; Mariette, born Dec. 25, 1854, married Jonathan Snyder, have a family; Leverett, born Feb. 6, 1857, married Cora Smith, Nov. 2, 1879, had two children, Perry F. and Ida I. Leverett died May 21, 1910; Rosa E., born Jan. 13, 1860, married Emmett Byers, and have children; and Harrison, born about 1862, is not living. Elijah Spencer died in the Union army in 1863, and his widow married JONATHAN SQUIRES, who was born in Ohio, July 25, 1823, by whom she had one son, J. Wilbur, born June 16, 1865 in DeKalb county, Ind., married Cora M. Thomas, Aug 24, 1887, they reside in St. Joseph, Mich., five children. Mrs. Myra Squires died in Allen county, Ind., Feb. 13, 1874.

Figure 13: An Example of Inferencing

The reasoning chain goes as follows. First we see that MYRA is a child of JAMES HARWOOD, so her full name at birth must have been Myra Harwood. Then we see that She married ELIJAH SPENCER, giving her a married name of Myra Spencer. Then Elijah Spencer died ... in 1863, making her his widow. As a widow she married JONATHAN SQUIRES, giving her the new married name of Myra Squires. Thus it seems highly likely (OntoSoar does not presently have any provision for assigning probabilities to these associations) that Mrs. Myra Squires is the same person as MYRA. We also infer from the phrase by whom she had one son that J. Wilbur is the son of the Squires.

The design of OntoSoar makes it possible to build inferencing of this sort. One of the key concepts that enables this inferencing power is the meaning schemas that we have modeled after the concept of *image schemas*, an idea dating back in the literature to at least Johnson (1987). He says:

*... image schemata ... are rich enough in internal structure to constrain our understanding and to generate definite patterns of inference.*  
Johnson (1987) p. 137

In OntoSoar the meaning schemas we define are not quite the same as Johnson's image schemata since they are not connected to perception in any direct way. Nevertheless the structure of having one schema with roles that connect to other schemas in a network provides declarative knowledge that enables adding the procedural knowledge that does inferencing. An important part of the meaning of things is built into the structures of these schemas. This contrasts with a system like LG-Soar which produces simple predicates without any of the additional knowledge required to know what these predicates actually mean. As more is added to OntoSoar's Semantic Analyzer, in both declarative and procedural knowledge, we expect the power of this approach will become ever more apparent.

### **Ontology matching**

Once we have analyzed an input segment to build our internal meaning structures, the final step is to project those meanings onto the ontology provided by the user. This work is done in two steps. Since both the internal meaning schemas of OCG and the user ontology are static, we can find object and relationship sets in the ontology that match parts of our schemas statically

before we have seen any input data. Then when a segment has been completely analyzed, we can use these matches to map the specific meanings found in the segment onto facts in the ontology. In Figure 7 both these steps are grouped together into a single component called the Mapper.

The matching operation is performed at the beginning of a run right after the ontology input file has been read in by a Soar operator called `find-matches`. In (12) we see the Soar trace for the matching part of a run using our example Ontology 2 from Figure 6.

```
(12)      7:      O: 06 (find-matches)
          10:      O: 012 (match-lexical)
          Lexical schema 'ProperName' matches lexical object set 'Name'.
          11:      O: 014 (match-person)
          Person matches object set 'Person'(osmx5) in the ontology.
          Person-to-ProperName matches rel set 'identified by'(osmx37) in the
          ontology.
          12:      O: 015 (match-couple)
          Couple matches rel set'married'(osmx304) in the ontology.
          13:      O: 09 (match-lexical)
          Lexical schema 'Date' matches lexical object set 'Date'.
          14:      O: 08 (match-lexical)
          Lexical schema 'Date' matches lexical object set 'MarriageDate'.
          15:      O: 011 (match-lexical)
          Lexical schema 'Date' matches lexical object set 'BirthDate'.
          16:      O: 010 (match-lexical)
          Lexical schema 'Date' matches lexical object set 'DeathDate'.
          17:      O: 016 (match-children)
          Found 'Son' specializing 'Person' by Q24.
          Found 'Daughter' specializing 'Person' by Q25.
          Found 'Child' specializing 'Person' by Q26.
          18:      O: 018 (match-life-event)
          Person role 'death' connects Person(osmx5) to DeathDate(osmx8) via died
          on(osmx49) in the ontology.
          19:      O: 017 (match-life-event)
          Person role 'birth' connects Person(osmx5) to BirthDate(osmx7) via born
          on(osmx43) in the ontology.
          20:      O: 013 (find-matches-done)
```

In general the matching operators work by matching keywords coded into the internal schemas with words taken from the names of the sets in the ontology. A lexical schema will match against any lexical object set that has a

word in its name matching one of the keywords coded on the schema. The Person schema matches to any object set regardless of its name as long as it has a relationship set connecting to a lexical object set that matches ProperName.

The Couple schema will match against a pattern with a relationship set with three or more arguments connecting the object set that matches Person with one of its specializations and a third argument that matches Date if that relationship set also has *married* in its name. This is a good example of how the matching process looks for words in the names in the ontology and also structural patterns that match up.

In the case of the FamilyRelationship schema, its matching algorithm looks for specializations of the object set which matches Person whose names contain the keywords *son*, *daughter*, or *child*.

The LifeEvent schema looks for matches to relationship sets where the name of the relationship set has a word that matches one of the verbs that can generate a LifeEvent. These matches are recorded according to the verb that matches, so that the general LifeEvent schema will match several relationship sets, with the correct match being chosen later on according to the specific verb present. This matching also connects to the correct role of Person, as shown in (12).

### **Extraction of facts**

When the semantic analysis of a given segment has been completed, the `extract-facts` Soar operator runs to project as many facts as possible from



the meanings found for the segment into the user ontology. Separate sub-operators extract facts according to the various types of matches found previously. This fact extraction process is fairly straightforward since we have already done the hard part in the matching.

In (13) we see the results for CCL 2, showing a person with a birth, a death, and two parents. In (13a) we have the input text, in (13b) a Soar trace of the process of extracting the facts for that segment, and in (13c) we have the console report generated by the Java code as it puts the facts into the actual OSMX file. The  $X_n$  and  $Y_n$  symbols are Soar internal symbols, while the `osmxnnn` symbols are OSMX identifiers. This example shows how the reference resolver can find the subject of the *son of* predicate and connect the son to his parents.

- (13) a. 2: Charles Christopher Lathrop, N. Y. City, born 1817, died 1865, son of Mary Ely and Gerard Lathrop ; ';'
   
b. 403: O: O358 (extract-facts)
   
404: ==>S: S120 (operator no-change)
   
405: O: O359 (setup-for-facts)
   
406: O: O360 (person-facts)
   
Extracting facts from Person(M131) 'Charles Christopher Lathrop'.
   
Extracting facts from Person(M202) 'Mary Ely'.
   
Extracting facts from Person(M229) 'Gerard Lathrop'.
   
407: O: O362 (life-event-facts)
   
Extracting facts from LifeEvent(M124) 'Charles Christopher Lathrop born'.
   
Extracting facts from LifeEvent(M146) 'Charles Christopher Lathrop died'.
   
408: O: O363 (couple-facts)
   
409: O: O364 (children-facts)
   
Extracting facts from FamilyRelation(M176) 'son' s(X1), o1(X3), o2(X5).
   
410: O: O365 (generalize-objects)
   
411: O: O361 (make-report)
   
412: O: O366 (extract-facts-done)

```

c. Facts extracted:
  Reporting 8 objects:
    X2: Name(osmx327, "Charles Christopher Lathrop")
    X1: Son(osmx331)
    X1: Person(osmx331)
    X4: Name(osmx336, "Mary Ely")
    X3: Person(osmx339)
    X6: Name(osmx342, "Gerard Lathrop")
    X5: Person(osmx345)
    X7: Date(osmx349, "1817")
    X7: BirthDate(osmx349, "1817")
    X8: Date(osmx354, "1865")
    X8: DeathDate(osmx354, "1865")
  Reporting 7 relations:
    Y1(osmx359): Person(osmx331) identified by Name(osmx327)
    Y2(osmx362): Person(osmx339) identified by Name(osmx336)
    Y3(osmx365): Person(osmx345) identified by Name(osmx342)
    Y4(osmx368): Person(osmx331) born on BirthDate(osmx349)
    Y5(osmx371): Person(osmx331) died on DeathDate(osmx354)
    Y7(osmx374): Son(osmx331) of Person(osmx345)
    Y6(osmx377): Son(osmx331) of Person(osmx339)

```

It is interesting that the facts extracted in (13) include populating both the object set and the relationship set associated with the *son of* relation. Part of this process involves entering an entity like `osmx331` as both a member of `Person` and of `Son` in the ontology, since `Son` is a specialization of `Person`.

## Final output

We just saw something of how `extract-facts` works and the results it produces for CCL 2. Now we look at the final fact listings for several other segments from our sample texts to see where the system does well and where it fails.

In (14) we see the results for the CCL 3 segment, which shows a marriage relation. This example shows the ability of the Mapper to handle relationship sets with an arity greater than 2. It also shows the ability of the reference resolver to find the antecedent for the GP at the beginning of CCL 3 all the way back in the subject of CCL 2.

- (14) a. 3: GP married 1856, Mary Augusta Andruss, 992 Broad St., Newark, N. J. ', '
- b. Facts extracted:
- Reporting 3 objects:
    - X10: Name(osmx380, "Mary Augusta Andruss")
    - X9: Spouse(osmx384)
    - X9: Person(osmx384)
    - X11: Date(osmx388, "1856")
    - X11: MarriageDate(osmx388, "1856")
  - Reporting 2 relations:
    - Y8(osmx393): Person(osmx384) identified by Name(osmx380)
    - Y9(osmx396): Person(osmx331) married spouse(osmx384)
- MarriageDate(osmx388)

In (15) we see a more complex use of reference resolution. The pronoun who in CCL 4 is matched to the object in CCL 3, and then other relations are attached to that referent.

- (15) a. 4: who was born 1825, daughter of Judge Caleb Halstead Andruss and Emma Sutherland Goble. '. '
- b. Facts extracted:
- Reporting 5 objects:
    - X13: Name(osmx400, "Judge Caleb Halstead Andruss")
    - X12: Person(osmx403)
    - X15: Name(osmx406, "Emma Sutherland Goble")
    - X14: Person(osmx409)
    - X16: Date(osmx412, "1825")
    - X16: BirthDate(osmx412, "1825")
  - Reporting 5 relations:
    - Y10(osmx416): Person(osmx403) identified by Name(osmx400)
    - Y11(osmx419): Person(osmx409) identified by Name(osmx406)
    - Y12(osmx422): Person(osmx384) born on BirthDate(osmx412)
    - Y14(osmx425): Daughter(osmx384) of Person(osmx409)
    - Y13(osmx428): Daughter(osmx384) of Person(osmx403)

Finally for CCL we see in (16) an example of how the OntoSoar syntactic and semantic analyzers combine to attach a death date to the proper person even over a large distance in the surface form of the sentence.

- (16) a. 5: Mrs. Lathrop died at her home, 992 Broad St., Newark, N. J, Friday morning, Nov. 4, 1898. '. '

- b. Facts extracted:  
 Reporting 3 objects:  
 X18: Name(osmx431, "Mrs Lathrop")  
 X17: Person(osmx434)  
 X19: Date(osmx437, "Nov 4 1898")  
 X19: DeathDate(osmx437, "Nov 4 1898")  
 Reporting 2 relations:  
 Y15(osmx441): Person(osmx434) identified by Name(osmx431)  
 Y16(osmx444): Person(osmx434) died on DeathDate(osmx437)

Many of the segments in the Myra sample show similar good results.

However, since the Myra sample uses much more complex linguistic structures the current limitations of the Semantic Analyzer are manifest in several mistakes it makes.

In (17-19) we see three consecutive segments from Myra 10-12. The first two work correctly, but the third has problems.

- (17) a. 10: Leverett, born Feb. 6, 1857. ', '  
 b. Facts extracted:  
 Reporting 3 objects:  
 X17: Name(osmx416, "Leverett")  
 X16: Person(osmx419)  
 X18: Date(osmx422, "Feb 6 1857")  
 X18: BirthDate(osmx422, "Feb 6 1857")  
 Reporting 2 relations:  
 Y11(osmx426): Person(osmx419) identified by Name(osmx416)  
 Y12(osmx429): Person(osmx419) born on BirthDate(osmx422)
- (18) a. 11: GP married Cora Smith, Nov. 2, 1879. ', '  
 b. Facts extracted:  
 Reporting 3 objects:  
 X20: Name(osmx432, "Cora Smith")  
 X19: Spouse(osmx435)  
 X19: Person(osmx435)  
 X21: Date(osmx439, "Nov 2 1879")  
 X21: MarriageDate(osmx439, "Nov 2 1879")  
 Reporting 2 relations:  
 Y13(osmx443): Person(osmx435) identified by Name(osmx432)  
 Y14(osmx446): Person(osmx419) married Spouse(osmx435)  
 MarriageDate(osmx439)

Instead of seeing that the couple represented by the GP in (19) had two children named Perry F. and Ida I., it thinks it found a person called Ida I. Leverett. A human being looking at this segment may also find it difficult to

understand it, but can figure out that Leverett is the first name of the same person mentioned in Myra 10. The real problem here is a serious ambiguity in the meaning of the period at the end of *Ida I.* Is it simply the period marking an abbreviation, or is it also the end of a sentence? It should really be the end of the sentence, but the OntoSoar architecture is not capable of understanding this. It would have to try doing syntactic and semantic analysis, find it doesn't work, and then go back and change the segmentation and start over.

- (19) a. 12: GP had two children, Perry F. and *Ida I.* Leverett died May 21, 1910; ','
- b. Facts extracted:
- Reporting 3 objects:
    - X23: Name(osmx450, "*Ida I.* Leverett")
    - X22: Person(osmx453)
    - X24: Date(osmx456, "May 21 1910")
    - X24: DeathDate(osmx456, "May 21 1910")
  - Reporting 2 relations:
    - Y15(osmx461): Person(osmx453) identified by Name(osmx450)
    - Y16(osmx464): Person(osmx453) died on DeathDate(osmx456)

Another limitation seen in (19) is that OntoSoar currently does not understand the *{x} had {n} children* construction. This, however, can be fixed with additional logic along the same lines as what is already there.

In Figure 13 we saw a complex line of reasoning to conclude who was who in the Myra sample. In (20-24) we see what the current OntoSoar does with this.

- (20) a. 17: Elijah Spencer died in the Union army in 1863. ','
- b. Facts extracted:
- Reporting 3 objects:
    - X35: Name(osmx517, "Elijah Spencer")
    - X34: Person(osmx520)
    - X36: Date(osmx523, "1863")
    - X36: DeathDate(osmx523, "1863")
  - Reporting 2 relations:
    - Y23(osmx527): Person(osmx520) identified by Name(osmx517)
    - Y24(osmx530): Person(osmx520) died on DeathDate(osmx523)

Here in (20) the system has no problem analyzing Elijah Spencer and his death date, but it does not yet have any logic to discover that he is probably the same person as the ELIJAH SPENCER in Myra 4 who married MYRA.

- (21) a. 18: and his widow married JONATHAN SQUIRES. ', '  
b. Facts extracted:  
Reporting 3 objects:  
X38: Name(osmx533, "JONATHAN SQUIRES")  
X37: Spouse(osmx536)  
X37: Person(osmx536)  
X39: Date(osmx540, "UNKOWN")  
X39: MarriageDate(osmx540, "UNKOWN")  
Reporting 2 relations:  
Y25(osmx544): Person(osmx536) identified by Name(osmx533)  
Y26(osmx547): Person(osmx520) married Spouse(osmx536)  
MarriageDate(osmx540)

Now in (21) we really go beyond what OntoSoar is now capable of. It does not have a construction to match *his widow* yet, nor the intelligence in the reference resolver to use gender cues and marriage relationships to discover that this refers to the original MYRA. As a result it decides that the subject of the marriage here is Elijah Spencer, person osmx520, which is clearly not correct.

- (22) a. 19: who was born in Ohio, July 25, 1823. ', '  
b. Facts extracted:  
Reporting 1 objects:  
X40: Date(osmx551, "July 25 1823")  
X40: BirthDate(osmx551, "July 25 1823")  
Reporting 1 relations:  
Y27(osmx555): Person(osmx536) born on BirthDate(osmx551)

Next in (22) the reference resolver correctly resolves who to JONATHAN SQUIRES, person osmx536.

- (23) a. 20: by whom she had one son, J. Wilbur, born June 16, 1865, in DeKalb county, Ind.. ', '

- b. Facts extracted:
  - Reporting 1 objects:
    - X41: Date(osmx558, "June 16 1865")
    - X41: BirthDate(osmx558, "June 16 1865")
  - Reporting 1 relations:
    - Y28(osmx562): Person(osmx536) born on BirthDate(osmx558)

Then in (23) things get really complicated. The current system does not understand either the *by whom* or *had one son* constructions, nor does it know how to attach J. Wilbur to one son as an appositive and therefore the subject of born. As a result it looks clear back to JONATHAN SQUIRES to find the subject of this born, not noticing that he already has a birth date.

- (24) a. 21: GP married Cora M. Thomas, Aug. 24, 1887. ', '
  - b. Facts extracted:
    - Reporting 3 objects:
      - X43: Name(osmx565, "Cora M. Thomas")
      - X42: Spouse(osmx568)
      - X42: Person(osmx568)
      - X44: Date(osmx572, "Aug 24 1887")
      - X44: MarriageDate(osmx572, "Aug 24 1887")
    - Reporting 2 relations:
      - Y29(osmx576): Person(osmx568) identified by Name(osmx565)
      - Y30(osmx579): Person(osmx536) married Spouse(osmx568)
- MarriageDate(osmx572)

Finally, since so many other pieces were missed, the system tells us in (24) that it was also JONATHAN SQUIRES who got married here, again. All this illustrates that the reference resolver needs a lot more constraints to keep it from making these false attachments, and that we need to implement more complex constructions so that examples such as these can be resolved properly.

Thus we see that OntoSoar is still a work in progress, but all these errors it makes currently, with the exception of the Ida I. Leverett one, can be corrected within the current architecture.

## 5. Results

In this chapter we examine the accuracy of the facts extracted by OntoSoar from various texts, starting with a detailed analysis of the results for the two samples given in Figures 1 and 2. Next we will look at each of the errors the system made and what would be needed to correct those errors. Then we quantify what happens when we apply the system to several samples taken from different family history books. Finally we examine how well the system responds to using different user ontologies.

### Results for the two samples

The working OntoSoar code<sup>8</sup> was applied to our two main sample texts shown in Figures 1 and 2, using the ontology shown in Figure 6. For each sample text an output OSMX file was produced which contained facts populating the ontology with persons identified by names, birth and death dates, and marriages. We discuss its performance on each of these types of facts.

#### *Persons*

Tables 1 and 2 show the results for Persons on the two samples. The system requires two pieces of information to create a Person: there must be a proper name, and that name must be the grammatical subject or object of a predicate which applies to people, such as *born*, *married*, or *son of*.

---

<sup>8</sup> All results reported here were obtained using the version of code as of 8 May 2014, change number 808.



CCL Example					
P Id	Osmx Id	Person by Name	OntoSoar	Correct	Reason
1	osmx331	Charles Christopher Lathrop	1	1	
2	osmx339	Mary Ely	1	1	
3	osmx345	Gerard Lathrop	1	1	
4	osmx384	Mary Augusta Andruss	1	1	
5	osmx403	Judge Caleb Halstead Andruss	1	1	
6	osmx409	Emma Sutherland Goble	1	1	
7	osmx434	Mrs. Lathrop	1	1	
8	osmx450	Charles Halstead	1	1	
9	osmx473	William Gerard	1	1	
10	osmx496	Theodore Andruss	1	1	
11	osmx512	Emma Goble	1	1	
12		Miss Emma Goble Lathrop	0	0	5
12		Totals	11/12	11/11	

Table 1: Person Facts for Sample 1

Myra Example					
P Id	Osmx Id	Person by Name	OntoSoar	Correct	Reason
1		JAMES HARWOOD	0	0	7A
2	osmx331	MYRA	1	1	
3	osmx350	ELIJAH SPENCER	1	1	
4	osmx369	Arvilla	1	1	
5	osmx385	Jonathan Snyder	1	1	
6	osmx401	Mariette	1	1	
7	osmx419	Leverett	1	1	
8	osmx435	Cora Smith	1	1	
9		Perry F.	0	0	2A, 6
10		Ida I.	0	0	2A, 6
11	osmx453	Leverett	1	0	6
12	osmx470	Rosa E.	1	1	
13	osmx486	Emmett Byers	1	1	
14	osmx504	Harrison	1	1	
15	osmx520	Elijah Spencer	1	1	
16	osmx536	JONATHAN SQUIRES	1	1	
17		J. Wilbur	0	0	4
18	osmx568	Cora M. Thomas	1	1	
19	osmx586	Mrs. Myra Squires	1	1	
19		Totals	15/19	14/15	

Table 2: Person Facts for Sample 2

Each row in these and the other results tables represents a fact found by a human being. The various columns in these tables are defined as follows: *P Id* is simply a number applied after the fact to easily identify which person we're talking about, *Osmx Id*, if present, is the unique identifier the OSMX file logic applies to this entity, *Person by Name* is the name from the original text that identifies this person, *OntoSoar* states 1 or 0 whether the OntoSoar system found this fact, *Correct* indicates whether the OntoSoar result was correct or not, and *Reason* gives a code number to be explained shortly for why OntoSoar did not get a correct answer where this is true.

From these two tables we see that there are six people mentioned in the texts that OntoSoar did not find correctly. In the CCL example the only missed person is Miss Emma Goble Lathrop. She is missed because segment CCL 15 does not have any predicates that the current OntoSoar understands.

The Myra 12 segment mentions three people, Perry F., Ida I., and Leverett, all of whom could be deduced by a human to have the last name *Harwood*. However, the first two are not found at all and the third is found incorrectly as Ida I. Leverett. All these errors are caused by a serious segmentation problem in Myra 12 due to the ambiguous period in Ida I.

J. Wilbur is not found because the entire semantic analysis of Myra 17-21 is crippled by the fact that the system does not yet understand three important constructions here: *his widow*, *she had one son*, and the use of J. Wilbur as an appositive. This lack of understanding causes errors in reference resolution as well, and thus some other facts are found incorrectly.

### *Births and Deaths*

In Tables 3 and 4 we show the results for births, and in Tables 5 and 6 those for deaths. In all four of these tables the rows have been removed for persons that do not have that event indicated in the text. The system finds every birth event, but the one for J. Wilbur is assigned to the wrong person.

CCL Example						
P Id	Osmx Id	Person by Name	Birth	OntoSoar	Correct	Reason
1	osmx331	Charles Christopher Lathrop	1817	1	1	
4	osmx384	Mary Augusta Andruss	1825	1	1	
8	osmx450	Charles Halstead	1857	1	1	
9	osmx473	William Gerard	1858	1	1	
10	osmx496	Theodore Andruss	1860	1	1	
11	osmx512	Emma Goble	1862	1	1	
6	Totals			6/6	6/6	

Table 3: Births for Sample 1

Myra Example						
P Id	Osmx Id	Person by Name	Birth	OntoSoar	Correct	Reason
2	osmx331	MYRA	July 26, 1835	1	1	
4	osmx369	Arvilla	1852	1	1	
6	osmx385	Mariette	Dec 25, 1854	1	1	
7	osmx419	Leverett	Feb 6, 1857	1	1	
12	osmx470	Rosa E.	Jan 13, 1860	1	1	
14	osmx504	Harrison	abt. 1862	1	1	
16	osmx536	JONATHAN SQUIRES	July 25, 1823	1	1	1
17		J. Wilbur	June 16, 1865	1	0	4
8	Totals			8/8	7/8	

Table 4: Births for Sample 2

For the deaths, shown in Tables 5 and 6, some dates are marked as UNKNOWN. This indication is used where the English text states that a person died, in these cases with the phrase `is not living`, but does not specify the

date. The *is not living* construction has not yet been programmed into OntoSoar. Other than these, all the other death dates were found correctly.

CCL Example						
P Id	Osmx Id	Person by Name	Death	OntoSoar	Correct	Reason
1	osmx331	Charles Christopher Lathrop	1865	1	1	
7	osmx434	Mrs. Lathrop	Nov 4, 1898	1	1	
8	osmx450	Charles Halstead	1861	1	1	
9	osmx473	William Gerard	1861	1	1	
4	Totals			4/4	4/4	

Table 5: Deaths for Sample 1

Myra Example						
P Id	Osmx Id	Person by Name	Death	OntoSoar	Correct	Reason
4	osmx369	Arvilla	UNKNOWN	0	0	2B
11	osmx453	Leverett	May 21, 1910	1	1	
14	osmx504	Harrison	UNKNOWN	0	0	2B
15	osmx520	Elijah Spencer	1863	1	1	
19	osmx586	Mrs. Myra Squires	Feb 13, 1874	1	1	
5	Totals			3/5	3/3	

Table 6: Deaths for Sample 2

### *Marriages*

Next we'll look at marriages, as shown in Tables 7 and 8. There is one marriage in the Sample 1 text, and six in Sample 2. OntoSoar finds all these marriages, but in two cases in Sample 2 it attaches the wrong subject to them. Both these errors are due to the problems with not understanding parts of the Myra 17-21 segments, as mentioned above.

CCL Example						
P Id	Osmx Id	Person by Name	Spouse	OntoSoar	Correct	Reason
1	osmx331	Charles Christopher Lathrop	Mary Augusta Andruss	1	1	
1		Totals		1/1	1/1	

Table 7: Marriages for Sample 1

Myra Example						
P Id	Osmx Id	Person by Name	Spouse	OntoSoar	Correct	Reason
2	osmx331	MYRA	ELIJAH SPENCER	1	1	
			JONATHAN SQUIRES	1	0	1, 2C
6	osmx385	Mariette	Jonathan Snyder	1	1	
7	osmx419	Leverett	Cora Smith	1	1	
12	osmx470	Rosa E.	Emmett Byers	1	1	
17		J. Wilbur	Cora M. Thomas	1	0	1
6		Totals		4/6	4/6	

Table 8: Marriages for Sample 2

### *Sons and Daughters*

So far the constructions for *son of* and *daughter of* have been implemented in OntoSoar. These are fairly straightforward to implement. However, many of the parent child relationships in these sample texts, and in many other texts as well, are represented as lists of children introduced by phrases like *Children of {person}.*, *Their children.*, or *They had {n} children.* OntoSoar does not yet implement any of these constructions for lists of children. Tables 9 and 10 show the results for parent/child relationships with the current system. In these two tables the OntoSoar column has been deleted to make the table fit on the page. We see that in the CCL example both parents were identified for both children connected to their parents with the constructions the system understands, but the rest of the parent/child

relations in CCL and all those in Myra are not found since they use constructions the system does not yet understand.

CCL Example					
P Id	Person by Name	Parent 1	Parent 2	Correct	Reason
1	Charles Christopher Lathrop	Mary Ely	Gerard Lathrop	1	
4	Mary Augusta Andruss	Judge Caleb Halstead Andruss	Emma Sutherland Goble	1	
8	Charles Halstead	Charles Christopher Lathrop	Mary Augusta Andruss	0	7B
9	William Gerard	Charles Christopher Lathrop	Mary Augusta Andruss	0	7B
10	Theodore Andruss	Charles Christopher Lathrop	Mary Augusta Andruss	0	7B
11	Emma Goble	Charles Christopher Lathrop	Mary Augusta Andruss	0	7B
12	Miss Emma Goble Lathrop	Charles Christopher Lathrop	Mary Augusta Andruss	0	7B
7	Totals			2/7	

Table 9: Sons and Daughters for Sample 1

Myra Example					
P Id	Person by Name	Parent 1	Parent 2	Correct	Reason
2	MYRA	JAMES HARWOOD		0	7A
4	Arvilla	MYRA	ELIJAH SPENCER	0	7C
6	Mariette	MYRA	ELIJAH SPENCER	0	7C
7	Leverett	MYRA	ELIJAH SPENCER	0	7C
9	Perry F.	Leverett	Cora Smith	0	2A, 6
10	Ida I.	Leverett	Cora Smith	0	2A, 6
12	Rosa E.	MYRA	ELIJAH SPENCER	0	7C
14	Harrison	MYRA	ELIJAH SPENCER	0	7C
17	J. Wilbur	MYRA	JONATHAN SQUIRES	0	2A, 4
9	Totals			0/9	

Table 10: Sons and Daughters for Sample 1

*Accuracy measures*

Tables 11 and 12 present the precision, recall, and F-measure for all the result types shown in Tables 1-10 for Samples 1 and 2, respectively. Table 13 combines these numbers into a single overall result set.

Accuracy Sample 1								
Category	Exist	Found	Correct	P Errors	R Errors	P	R	F
Persons	12	11	11	0	1	100.0%	91.7%	95.7%
Births	6	6	6	0	0	100.0%	100.0%	100.0%
Deaths	4	4	4	0	0	100.0%	100.0%	100.0%
Marriages	1	1	1	0	0	100.0%	100.0%	100.0%
Sons & Daughters	7	2	2	0	5	100.0%	28.6%	44.4%
Totals/Average	30	24	24	0	6	100.0%	80.0%	88.9%

Table 11: Accuracy Measures for Sample 1

Accuracy Sample 2								
Category	Exist	Found	Correct	P Errors	R Errors	P	R	F
Persons	19	15	14	1	4	93.3%	73.7%	82.4%
Births	8	8	7	1	0	87.5%	87.5%	87.5%
Deaths	5	3	3	0	2	100.0%	60.0%	75.0%
Marriages	6	6	4	2	0	66.7%	66.7%	66.7%
Sons & Daughters	9	0	0	0	9	N/A	0.0%	0.0%
Totals/Average	47	32	28	4	15	87.5%	59.6%	70.9%

Table 12: Accuracy Measures for Sample 2

Combined Accuracy for Samples 1 and 2								
Category	Exist	Found	Correct	P Errors	R Errors	P	R	F
Persons	31	26	25	1	5	96.2%	80.6%	87.7%
Births	14	14	13	1	0	92.9%	92.9%	92.9%
Deaths	9	7	7	0	2	100.0%	77.8%	87.5%
Marriages	7	7	5	2	0	71.4%	71.4%	71.4%
Sons & Daughters	16	2	2	0	14	100.0%	12.5%	22.2%
Totals/Average	77	56	52	4	21	92.9%	67.5%	78.2%

Table 13: Combined Accuracy Measures

Overall we see that the precision is quite high, but the recall is lower. The primary reason for all the errors is the lack of understanding of all the linguistic constructions used in the text.

### Analysis of errors

In order to understand better the errors and omissions that OntoSoar makes, Tables 1-10 have a column on the right giving a reason code for every case in which OntoSoar did not get the correct answer. Table 14 lists these codes and their meanings.

Error Reason Codes		
Reason	Description	Count
1	Handling lists of children	12
2	Construction not yet implemented	11
A	{p} had {x} son/daughter/child/children	6
B	{p} is not living	2
C	his widow	2
D	{x} is {y}	1
3	Inability to segment on ambiguous period	5
4	Appositive not connected	3
5	Not finding alternative names	2

Table 14: Error Reason Codes

Reason 1 summarizes all the cases where a construction that initiates a list of children is not yet understood, causing a total of 12 errors. Recognizing these constructions is straightforward within the existing structure. However, additional semantic logic is required to attach a new person to the current list of children that is being constructed. This is complicated because we also have to detect when a given list has ended and we should not consider it anymore, and also because these lists can be nested, as shown in Myra 12.



There are several other constructions that appear in our two samples that are not yet understood, causing a total of 11 errors collected under Reason 2. All these can be implemented directly within the current structure, with the most complicated being 2C, *his widow*, since it requires applying both gender and relationship constraints.

The marriage in segment Myra 18, described by `his widow married JONATHAN SQUIRES`, needs the pronoun `his` to be resolved to `Elijah Spencer`, which the reference resolver already does successfully. However, this will not be enough until we also recognize the *his widow* construction and the semantics of *widow*, as Reason 2C says, and resolve that `ELIJAH SPENCER` and `Elijah Spencer` are the same person.

Reason 3 is a problem that seems beyond the scope of this project to resolve. Segment Myra 12 says: `GP had two children, Perry F. and Ida I. Leverett died May 21, 1910;.` The problem is that the period in `Ida I.` might be just part of an abbreviation or it might indicate the end of a sentence, as it should here. However, there's no way to tell that without using higher-level semantics to go back and change the way the segmentation was done so that the parser can get the right answer. This backward flow in the system's pipeline does not fit into the current OntoSoar architecture.

Reason 4 also involves additional logic to connect appositives to the things they refer to, as in the case of `J. Wilbur` in Myra 20. This logic can be patterned after the reference resolver that is already working.

Performing inferences on names is a fairly complex piece of logic summarized here as Reason 5. The system should be able to deduce the various alternative surnames of women caused by marriage, determine that ELIJAH SPENCER and Elijah Spencer are the same person, and even discover that Mrs. Myra Squires is the same person as MYRA. The meaning structures already built provide the framework in which additional inferencing logic can solve these problems.

### **Results on additional samples**

We have seen that the current OntoSoar system does a pretty good job on our too sample texts, and looked in some detail at how the remaining problems could be solved within the existing architecture. However, these are only two small samples. Here we examine the results of applying the system to a larger sampling of texts from family history books.

The BYU Data Extraction Group has access to a private repository of over a hundred thousand of such books. Previous work by this group produced a randomized list of the books, and then selected 200 books from the beginning of this randomized list. Another process randomly chose a sequence of three consecutive pages from each of these books. The data reported here are based on building twelve text files from the three identified pages of twelve arbitrarily chosen books from the list of 200. Each of these twelve text files, with three pages of data each, was run through OntoSoar and the results collected.

As might be expected, the first time these twelve files were run through the system several issues were uncovered that caused OntoSoar to crash before

finishing a given file. One issue was that some of the files had Unicode characters that the code could not handle, so the character set for the input reader was changed to resolve this. A bug in the Java code of the LG Parser was found that caused an exception for certain unusual words, and this was fixed. Improvements were made to the Segmenter to make it handle more abbreviations. It was also changed to force a segment break after 40 tokens, since the time taken by the LG Parser can grow exponentially with the length of the segment, and some very long segments were taking many minutes to parse. Once all these changes were made, all twelve of the text files ran through OntoSoar with no problems.

Doing a complete measure of the precision and recall of OntoSoar on this data would require manually annotating all the texts for all the relations of interest, which was beyond the scope of the available resources. However, we have looked through all the output files to examine the facts that OntoSoar claims to have found and evaluated each claimed fact as correct or not. The results are summarized in Table 15. The *OD* in the file names stands for *Other Data*. The numbers for the CCL and Myra samples are included as the first two rows in Table 15 for comparison, but the Totals row only includes the OD files, those below the double line.

The analysis performed to get the results in Table 15 was rather complex and tedious. Persons were considered correct if they were identified by at least a subset of the name given in the text with no extraneous material. Births and deaths were considered correct if they were attached to a legitimate person and

the date was complete. A marriage was considered correct if it connected the two correct people, even if the date was not found or incomplete. A child was considered correct if a person of the right gender was connected as a son or daughter to at least one of the correct parents.

Results for Other Data Files											
		Persons		Births and Deaths		Marriages		Children		Run Time	
File	Segs	Found	Correct	Found	Correct	Found	Correct	Found	Correct	Secs	Segs/Sec
CCL	15	11	100.00%	10	100.00%	1	100.00%	2	100.00%	15	1.000
Myra	23	15	93.33%	11	100.00%	6	66.67%	0	0.00%	10	2.300
OD1	174	82	76.83%	56	14.29%	20	70.00%	10	70.00%	296	0.588
OD2	141	19	42.11%	13	61.54%	5	40.00%	0	N/A	119	1.185
OD3	67	2	100.00%	0	N/A	0	N/A	0	N/A	126	0.532
OD4	103	9	100.00%	0	N/A	0	N/A	0	N/A	106	0.972
OD5	149	5	40.00%	4	50.00%	1	0.00%	0	N/A	153	0.974
OD6	57	5	40.00%	0	N/A	0	N/A	2	50.00%	35	1.629
OD7	57	55	80.00%	15	6.67%	2	50.00%	16	81.25%	65	0.877
OD8	152	34	55.88%	6	50.00%	13	30.77%	2	100.00%	106	1.434
OD9	174	44	90.91%	35	82.86%	11	45.45%	0	N/A	115	1.513
OD10	256	32	78.13%	23	73.91%	13	46.15%	1	100.00%	212	1.208
OD11	154	41	65.85%	24	12.50%	13	61.54%	0	N/A	124	1.242
OD12	63	0	N/A	0	N/A	0	N/A	0	N/A	32	1.969
Totals	1547	328	73.48%	176	40.34%	78	51.28%	31	77.42%	1489	1.039

Table 15: Precision Data for Additional Texts

Table 15 only gives an estimate of precision, no attempt was made to measure either recall or F-measure. In general, however, we can say that the overall recall for these twelve files is rather low. If no facts were found in a particular case, the precision is marked as N/A.

Many issues contribute to both recall and precision being much lower than for our original two samples. Some, such as OCR errors, are mostly beyond the reach of OntoSoar to solve. Other types of errors, however, could

be reduced substantially by further improvements to OntoSoar within the scope of its existing architecture.

The OD files contain many instances of dates formatted like 25 June 1823 or 6/25/1823. At the moment OntoSoar does not understand either of these date formats, but that could be fixed with not too much effort. Also, in the OD10 document many dates are listed as `Private`, presumably because the persons are still living. The grammar of the LG Parser could be easily modified to interpret this as a date.

Much of the lack of recall and many precision errors as well are caused by constructions that OntoSoar does not yet understand. One example from OD7 of a pattern that appears in many of these files is shown in (25).

(25) 1: (945) Gordon John Harris, son of John Phillip and Alice Adel (Billeter) Harris, was born 16 Aug 1937 in Gordon, Sheridan, Nebraska.

The Semantic Analyzer currently does not know how to build the names of the two parents correctly here, especially dealing with the maiden name of the mother in parentheses. It concludes the parents are John Phillip and Alice Adel, without any surnames. It also concludes that Harris is another person, the one born on 16 Aug 1937. Nevertheless, it succeeds in asserting that Gordon John Harris is the son of John Phillip and Alice Adel. All this could be improved upon with more intelligent analysis of names and conjunctions.

Many of the OD files have various forms of list item labels, generation numbers as superscripts, and other extraneous information mixed in with the data. In (26) we see some of these issues in a snippet from OD2.

```
(26) 9: 13 15 I Tryntje Kool, Bapt. '.'
      10: Mar. 25, 1724 at Hackensack N. J. '.'
      11: J. II Saartje Kool, Bapt. '.'
      12: Dec. 19, 1725 at Hackensack N. J. III Abram Kool, born Jan. 2,
      1729. '.'
```

Here we not only have extra numbers and roman numerals, but also the abbreviation `Bapt.`, which `OntoSoar` doesn't understand. As a result of these problems and related segmentation errors, the only facts `OntoSoar` finds from these four segments are that `Hackensack N. J. III Abram Kool` is a person who was born on `Jan 2 1729`. Well, it got the date right anyway. Fixing errors of this sort will require improvements to the `Segmenter`, the `LG Parser`, and the `Semantic Analyzer`.

Table 15 shows clearly that each of the OD files has its own idiosyncrasies. OD1 seems to have the best overall performance except for the birth and death dates, which are confused by a pattern of putting the place between the verb and the date. OD3 is just a list of deaths, with no verbs to connect the names with the dates. `OntoSoar` only manages to find two people in the whole file, which are found due to other constructions mixed in. OD4 gives very few facts since it uses abbreviations for our predicate words without any periods, and the `Segmenter` does not yet recognize these to expand them. OD7 gives good results for persons, sons, and daughters but not for any events. We get the highest performance for persons, births, and deaths on OD9, but

marriage accuracy is poor and we don't find any sons or daughters. It seems strange that we find no facts at all in OD12, but it turns out that this file, or at least the three pages chosen to process, is all text from legal documents. A few names of people are mentioned, but without any of the genealogy relations we are looking for.

The main takeaway from Table 15 is that each of these books has its own idiosyncratic style and that an effective information extractor for all of them must somehow cover or adapt to all these styles. The approach that OntoSoar takes requires it to be provided with knowledge of the syntax and semantics of many different linguistic constructions, and new ones to for each new style. It has the advantage, though, that the constructions it already knows don't seem to cause much harm if they don't fit a new style.

Another general observation from looking at all this data is that the reference resolver works pretty well most of the time. However, as in some cases in our Myra sample, when certain noun phrases are not understood it just skips over them and goes much too far, finally finding an incorrect referent. This could be improved by doing a better job of recognizing all the referring expressions, or by making the reference resolver smart enough to recognize that it is passing over an unrecognized reference and not go any further.

As we saw with our two original samples, OntoSoar's performance can be improved by giving it more knowledge at each of its processing levels.

## **Run time performance**

As well as precision data, Table 15 gives information on how fast OntoSoar is at processing data. It is interesting to see that it goes 2.3 times as fast on the Myra sample than on the CCL sample, which seems surprising at first since the Myra sample has more complex language. The reason for this is in the performance of the LG Parser. The LG Parser carries the full weight of searching through a space of possible alternative parses, a task which tends to grow exponentially with the length of the input segment. Thus the longer the segments the slower the parser goes. CCL has fewer, longer segments than Myra, and thus takes longer to parse.

Overall for this whole collection of data OntoSoar processes consistently at around one segment per second. Considering that in this time it is not just reading and understanding the text, but also using what it understands to populate an ontology and output the facts in a very structured form, this is much faster than a human indexer could produce the same results.

## **Results with different ontologies**

As one might expect, when OntoSoar is run using Ontology 1 (shown in Figure 5), it finds the same facts for persons, births, and deaths as mentioned above using Ontology 2 (from Figure 6). It is interesting to note that when using Ontology 1 it succeeds in finding a number of persons who are connected by relations that are not in the ontology. For instance, in the first few segments of the CCL text, it finds the four parents that are objects of the *son of* and *daughter of* relations even though the ontology cannot represent these



relations. Thus OntoSoar reports the existence of these four individuals without being able to connect them up to anything else. This shows how the internal meaning representations in OntoSoar are richer than the ontologies we are using.

Figure 14 gives another interesting ontology for this domain.

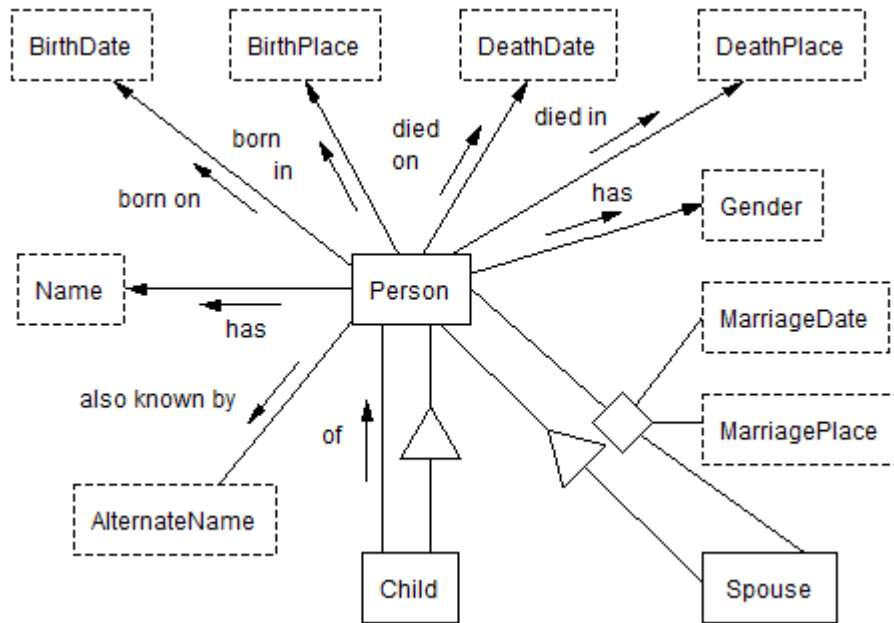


Figure 14: Ontology Example 3

When the system is run with this ontology it finds all the same facts as were found with Ontology 2, except for sons and daughters. At the present time OntoSoar has no way of knowing that a *son* is also a *child*, and Ontology 3 has no specific object sets for sons and daughters. When more inference rules for reasoning about all the possible arrangements of family relationships are added, this problem should be solved.

## **6. Conclusions and Future Work**

In this chapter we examine what has been demonstrated by this work, both positively and negatively, with respect to our Thesis Statement. Then we consider possible improvements that could be made to the existing system, some incremental and some more major. Finally we look at possible future research directions that are suggested by this work above and beyond the current OntoSoar architecture.

### **What has been demonstrated**

This thesis has demonstrated a number of important points that relate to our Thesis Statement:

- Linguistic analysis can find genealogy facts.
- The Link Grammar Parser can be modified to adapt to domain-specific language variations.
- A fairly simple preprocessor can segment an input text into segments that are reasonable for the LG Parser to process as individual chunks.
- A construction grammar approach can extract useful meaning structures from LG Parser linkages using built-in rules.
- Meaning structures built using construction grammar can be mapped onto ontologies to populate a conceptual model read from an input file with facts found in the text being processed.

- The Soar architecture can support the above processes as well as providing a basis for more extensive inferencing for reference resolution, name inferencing, and duplicate identification.
- Inference rules written in Soar code can find referents for pronouns and other referential expressions with an accuracy that depends on correctly recognizing the referential expressions themselves.

In addition to these positive results, several limitations of the existing system have been identified:

- Knowledge about the syntactic structure and meaning of every word and grammatical construction that the system should recognize must be built in by hand in Soar or Java code.
- OCR errors, different usages of punctuation, unknown abbreviations, and other additional textual items such as list item labels can confuse the system and make it either not find important facts or find them incorrectly.
- In general all the books looked at here use a highly abbreviated form of English. However, the style of representing genealogy facts varies considerably from one book to another. To cover a wide range of books a wide range of possible representations of facts must be built into the system's internal knowledge.

### **Possible incremental improvements**

OntoSoar can be improved incrementally by adding or modifying rules in several parts of the system: the Segmenter can be made to recognize and expand new abbreviations such as *b*, *dau*, and *Bapt.*, the grammar of the LG Parser can be augmented to understand different date formats, and the constructions in the Semantic Analyzer can be expanded to recognize phrases like *his widow* and *she had one son*, as well as a host of others that can be found in family history books.

### **Possible major additions**

Adding new rules at various levels can improve OntoSoar considerably, but some things will be difficult to accomplish in this way. More major changes or additions could be beneficial.

One thing that causes considerable difficulty in some of the texts we have looked at is that the LG Parser often has great difficulty in properly parsing place names. Also, the system often confuses place names with person names. A good named entity recognizer might help considerably. Suppose the input text were first run through a named entity recognizer, probably even before segmentation, that could accurately identify which phrases are names of people, which are names of places, which are dates or time expressions of some sort, and which are names of organizations or other entities. Then the input to the LG Parser could simply be a single unique identifier for each entity recognized, and the phrases these identifiers represent could enter the system in parallel and be used to provide real lexical strings farther downstream. This

would greatly simplify the job of the parser, and help the Semantic Analyzer match up entities with argument slots in the predicates that are found. This has the potential of improving considerably the overall accuracy of the system and the difficulty of writing the constructions and inference rules the system needs.

Many of the texts we have seen contain structures that the current OntoSoar does not understand at all. These include the child numbers seen in our CCL sample and similar things in many other documents, as well as indentation and paragraph markings. If we had a preprocessor of some sort that could analyze the text to find these structures, this could help segmentation and help the reference resolver know where important contextual boundaries fall. Also things like list item labels could be associated with regions of text without being included in the segments the parser sees, greatly reducing confusion in the parser.

The biggest obstacle to expanding the coverage of OntoSoar to a much wider range of texts is the time required to write and debug the Soar productions that implement the system's understanding of a wider range of grammatical constructions. It should be possible to design a higher-level language, patterned partly after the ECG formalism given in Bryant (2008), to represent construction recognizers, meaning schemas, and inference rules. Then a compiler could be written to compile this language into Soar code. This idea was considered early in this project, but at that time it was felt that we didn't yet know enough about the nature of these various rules and the Soar

code required to implement them until a reasonable set of concrete examples had been built and debugged. With OntoSoar as it is today, it is ripe for undertaking such a project.

Evaluating the performance of the current OntoSoar system is very tedious and time consuming. The OntoES tool set has an Annotator tool that allows a human being to annotate a given text easily in a graphical interface without having any technical knowledge of the internals of the system. There is also a tool that can compare the output of this human annotation with the output of OntoSoar for the same input text. However, the usefulness of such an approach is somewhat limited by the fact that the Annotator keeps track of the exact physical location in the original PDF file of every string it captures, whereas OntoSoar does not. OntoSoar currently does keep track of the segment and range of tokens that each construction represents, and this could be augmented with the additional information needed to provide the exact physical locations for comparison.

### **Future research directions**

As a master's thesis project, OntoSoar is naturally rather limited in scope. Its successes, however, can point the direction for more ambitious research in at least three areas: parsing, learning from human interaction, and deeper learning of the semantics of words and phrases.

One major limitation of the OntoSoar architecture is its pipeline approach to the problem. The Segmenter operates on a whole text file as a single unit, producing a list of segments before any have been processed by the

rest of the system. Similarly, the LG Parser processes an entire segment as a single unit before the Semantic Analyzer has a chance to see any of it. The Semantic Analyzer does work incrementally one word at a time, but it is limited by the constraints imposed by the upstream components. There is no way for the semantics to feed information back to the parser, or for the syntax and semantics to feed information back to the segmenter.

Another approach would be to have the whole system work on an incremental basis. As each new word comes in it can be looked up in a lexicon and its related syntactic and semantic roles used to recognize grammatical constructions, what they mean, and where one ends and the next begins. Then the search through a space of alternative parses would not be limited to just using syntactic knowledge, but semantic and textual knowledge as well. It might seem that this would make the search space explode even more, but actually the constraints supplied by the additional knowledge available at each step of the process could actually reduce the number of alternatives at each point.

Even better than developing a high-level language to program rules into the system would be having a way that the system can learn the rules itself in some way. One possible approach to this problem would be to use human interaction to help the system learn. For example, suppose a human annotator uses something looking like the existing Annotator to start marking up a text, but behind the scenes the system is analyzing the human's decisions and building construction patterns and inference rules to produce those same

results. Then the system can use its new hypothesized rules to label a lot more data, with these hypothetical results being presented to the user for further refinement. In this way the system could learn until it can provide adequate performance without any further human input. It may be necessary to have some of this human input for each new book to be processed.

Another possible approach to learning would be to have a system that really learns from scratch a large amount of linguistic knowledge in a way similar to the way humans learn a new language, either as children or adults. Tomasello (2003) describes a good deal of empirical evidence of how children learn words, simple phrases, and then abstract language through social engagement with adult language users. A system that could learn a large amount of a given language in this manner, and then be refined to learn the specifics of a particular domain like family history, could probably be much more flexible and robust than a system like OntoSoar based on programmed or learned domain-specific rules.

Some of these ideas are very ambitious dreams at this point in time. Nevertheless, OntoSoar has pioneered a new approach to extracting information from text which can inspire a lot of further research. We look forward to participating in that endeavor.



## 7. References

- Akbik, Alan and Jurgen Bross (2009). Wanderlust: Extracting Semantic Relations from Natural Language Text Using Dependency Grammar Patterns. *Proceedings of the World Wide Web Conference (WWW2009) Semantic Search 2009 Workshop (SemSearch09)*, Madrid, Spain.
- Anderson, John R. (2007). *How Can the Human Mind Occur in the Physical Universe?* Oxford University Press, Oxford and New York.
- Bergen, Benjamin and Nancy Chang (2013). *Embodied Construction Grammar*. Available 4/22/2014 at [http://www.cogsci.ucsd.edu/~bkbergen/papers/ECG\\_Handbook.pdf](http://www.cogsci.ucsd.edu/~bkbergen/papers/ECG_Handbook.pdf).
- Bergen, Benjamin E. and Nancy C. Chang (2005). Embodied Construction Grammar in Simulation-Based Language Understanding, in *Construction Grammars: Cognitive grounding and theoretical extensions*, Jan-Ola Östman and Mirjam Fried, eds. John Benjamins Publishing Company, Amsterdam and Philadelphia.
- Bleiholder, Jens and Felix Naumann (2008). Data Fusion. *ACM Computing Surveys*, Vol. 41, No. 1, Article 1, pp. 1:1-1:41.
- Bryant, John Edward (2008). *Best-Fit Constructional Analysis*. PhD dissertation, University of California at Berkeley.
- Buitelaar, Paul, Philipp Cimiano, Peter Haase, and Michael Sintek (2009). Towards Linguistically Grounded Ontologies, *Proceedings of the 6th European Semantic Web Conference (ESWC'09)*, Heraklion, Greece, May/June 2009.
- Chang, Nancy Chih-Lin (2009). *Constructing grammar: A computational model of the emergence of early constructions*. PhD dissertation, University of California at Berkeley.
- Chierchia, Gennaro and Sally McConnell-Ginet (2000). *Meaning and Grammar: An Introduction to Semantics, second edition*. The MIT Press, Cambridge, MA.
- Chomsky, Noam (1957). *Syntactic Structures*. Mouton & Co.
- Chomsky, Noam (1995). A Minimalist Program for Linguistic Theory. In N. Chomsky *The Minimalist Program* (pp. 167-217). Cambridge MA, MIT Press.
- Cimiano, Phlipp (2006). *Ontology Learning and Population from Text: Algorithms, Evaluation and Applications*. Springer, New York.
- Embley, David w., Barry D. Kurtz, and Scott N. Woodfield (1992). *Object-Oriented Systems Analysis: A Model-Driven Approach*. Yourdon Press, Englewood Cliffs, New Jersey.
- Embley, David W., Steven W. Liddle, and Deryle W. Lonsdale, (2011). Conceptual Modeling Foundations for a Web of Knowledge, in *Handbook of Conceptual Modeling*, David W. Embley and Bernhard Thalheim, eds., Chapter 15. Springer. Available 4/25/2014 at <http://deg.byu.edu/papers/ExtractionOntologies.pdf>.

- Euzenat, Jerome, and Pavel Schvaiko (2007). *Ontology Matching*. Springer, Berlin.
- Feldman, Jerome A. (2006). *From Molecule to Metaphor: A Neural Theory of Language*. MIT Press, Cambridge, MA.
- Harwood, Watson H., M.D. (1911). A Genealogical History of the Harwood Families, descended from Andrew Harwood, Whose English home was in Dartmouth, Devonshire, England, And who emigrated to America, and was living in Boston, Mass., in 1643. Third Edition. Chasm Falls, N. Y.
- Hoffman, Thomas and Graeme Trousdale, eds. (2013). *The Oxford Handbook of Construction Grammar*. Oxford University Press, New York.
- Hruschka, Estevam R. Jr. (2013). Machine Reading the Web. Tutorial given at the 22nd International World Wide Web Conference, Rio de Janeiro, Brazil, 13-17 May, 2013.
- Jackendoff, Ray (1990). *Semantic Structures*. The MIT Press.
- Jackendoff, Ray (1996). Semantics and Cognition, in Shalom Lappin ed. *The Handbook of Contemporary Semantic Theory*. Blackwell.
- Jackendoff, Ray (2002). *Foundations of Language: Brain, Meaning, Grammar, Evolution*. Oxford University Press.
- Jackendoff, Ray (2003). Précis of *Foundations of Language: Brain, Meaning, Grammar, Evolution*. *Behavioral and Brain Sciences*, 26, 651-707.
- Johnson, Mark (1987). *The Body in the Mind: The Bodily Basis of Meaning, Imagination, and Reason*. The University of Chicago Press, Chicago.
- Jurafsky, Dan and James H. Martin (2008). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, Second Edition. Prentice Hall.
- Laird, John E. (2012). *The Soar Cognitive Architecture*. The MIT Press, Cambridge, MA.
- Lakoff, George (1987). *Women, Fire, and Dangerous Things: What Categories Reveal About the Mind*. University of Chicago Press.
- Lakoff, George and Mark Johnson (1980a). *Metaphors We Live By*. University of Chicago Press.
- Lakoff, George and Mark Johnson (1980b). The Metaphorical Structure of the Human Conceptual System. *Cognitive Science* 4, 195-208.
- Lonsdale, D. W., C. Tustison, C. G. Parker, and D. W. Embley (2008). Assessing clinical trial eligibility with logic expression queries, *Data & Knowledge Engineering*, Volume 66 Issue 1, July, 2008, Pages 3-17.
- Marcus, Gary F. (2001). *The Algebraic Mind: Integrating Connectionism and Cognitive Science*. MIT Press, Cambridge, MA.

- Mitra, Prasenjit, Natalya F. Noy, and Anuj R. Jaizwal (2004). OMEN: A probabilistic ontology mapping tool. In Proceedings of the Meaning Coordination and Negotiation workshop at the International Semantic Web Conference (ISWC), pp. 537-547.
- Mohan, Shiwali, Aaron H. Mininger, and John E. Laird (2013). Towards an Indexical Model of Situated Language Comprehension for Real-World Cognitive Agents, in *Proceedings of the Second Annual Conference on Advances in Cognitive Systems*. ACS-2013 (153-170).
- Mohan, Shiwali, Aaron H. Mininger, James R. Kirk, and John E. Laird (2012). Acquiring Grounded Representations of Words with Situated Interactive Instruction, in *Advances in Cognitive Systems 2(2012)* 113-130.
- Newell, Allen (1990). *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press.
- Sarawagi, Sunita (2008). Information Extraction, *Foundations and Trends in Databases*, Vol. 1, No. 3 (2007) 261-377.
- Sleator, Daniel D. K. and Davy Temperley (1991), *Parsing English with a Link Grammar*, Carnegie Mellon University Computer Science technical report CMU-CS-91-196, October 1991.
- Sleator, Daniel D. K. and Davy Temperley (1993), Parsing English with a Link Grammar, *Third International Workshop on Parsing Technologies*.
- Tomasello, Michael (2003). *Constructing a Language: A Usage-Based Theory of Language Acquisition*. Harvard University Press, Cambridge, MA.
- Tustison, Clint A. (2004). *Logical Form Identification for Medical Clinical Trials*. BYU Linguistics MA Thesis, December, 2004.
- Vanderpoel, Geo. B., ed. (1902). *The Ely Ancestry: Lineage of Richard Ely of Plymouth England, who came to Boston, Mass., about 1655, & settled at Lyme, Conn, in 1660*. The Calumet Press, New York.