2012-12-10

Using a Cognitive Architecture in Incremental Sentence Processing

Jeremiah Lane McGhee
Brigham Young University - Provo

Using a Cognitive Architecture in Incremental Sentence Processing

Jeremiah McGhee

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

Master of Arts

Deryle Lonsdale, Chair
Alan K. Melby
Dallin D. Oaks

Department of Linguistics and English Language

Brigham Young University

December 2012

ABSTRACT

Using a Cognitive Architecture in Incremental Sentence Processing

Jeremiah McGhee
Department of Linguistics and English Language
Master of Arts

XNL-Soar is a specialized implementation of the Soar cognitive architecture. The version of XNL-Soar described in this thesis builds upon and extends prior research (Lewis, 1993; Rytting, 2000) using Soar for natural language processing. This thesis describes the updates made to operators creating syntactic structure and the improved coverage of syntactic phenomena. It describes the addition of semantic structure building capability. This thesis also details the implementation of semantic memory and describes two experiments utilizing semantic memory in structural disambiguation. This thesis shows that XNL-Soar, as currently instantiated, resolves ambiguities common in language using strategies and resources including: reanalysis via snip operators, use of data-driven techniques with annotated corpora, and complex part-of-speech and word sense processing based on WordNet.

Keywords: language processing, cognitive modeling, incremental parsing, syntax, semantics, Soar

ACKNOWLEDGMENTS

I wish to express my sincere gratitude to my wife and children. Thank you for your patience through the many nights of reading and writing over the last several years. Completing this thesis has been a long process and your love, support and understanding has been invaluable.

My thanks also to Dr. Deryle Lonsdale, my thesis chair. His patience and diligence in mentoring provided me with the skills necessary to complete this work, an endeavor I would not have been able to pursue without his guidance and help.

I must also thank Drs. Alan K. Melby and Dallin D. Oaks for their time and effort in helping me complete the thesis and defense process.

Finally, I would like to thank the NL-Soar research group, my colleagues and friends in this work: Nathan Glenn, Ross Hendrickson, Tory Anderson, Carl Christensen, and Seth Wood.

Contents

List of Figures

List of Tables

Chapter 1: Introduction

Ambiguity in language is pervasive. Two principal types of ambiguity can be distinguished based on their cause:

- lexical ambiguity (more than one word meaning)
- structural ambiguity (more than one syntactic structure)

For example:

(1.1) '*My brother found an unusual bat.*'

(1.2) '*The officer shot the criminal with a gun.*'

(1.3) '*The officer saw the criminal with a telescope.*'

Example 1.1 illustrates a lexical ambiguity: is the '*bat*' a wooden club or winged mammal? Examples 1.2 and 1.3 contain the same structural ambiguity, namely the attachment site of the phrase beginning with the preposition '*with*'. The prepositional phrase can attach to the noun '*criminal*', meaning that the criminal has either the telescope or the gun. Alternatively if the prepositional phrase attaches to the verb ('*shot*' or '*saw*') it means the action was performed with the gun or telescope.

While humans can easily cope with these and other types of linguistic ambiguity, the existence of multiple possible interpretations presents an obstacle for automatic interpretation by machine. Creating systems to process language is a very active field of research. From Google's search algorithms to speech recognition on a cell phone, ambiguity creates problems that reduce these systems' effectiveness.

However, the state of the art in artificial intelligence and natural language processing offers a number of exciting possibilities for dealing with ambiguity. For example, the NL-Soar

project (Lewis, 1993; Lonsdale and Rytting, 2006) aims to extend this research by creating a cognitively plausible framework for language processing. NL-Soar is a specialized adaptation of the Soar cognitive architecture, a symbolic architecture first proposed by Newell (1990). The ultimate goal of the NL-Soar project is to create a robust system of language comprehension, production and discourse, all grounded in cognitive plausibility. A more detailed discussion of NL-Soar can be found in section 2.3.4.

This thesis describes XNL-Soar, a new implementation of the comprehension component of NL-Soar. The XNL-Soar system uses the Minimalist Program as the theoretical basis for syntactic representation (Chomsky, 1965, 1995; Adger, 2003). Additionally XNL-Soar generates semantic representations following the account of Lexical Conceptual Structures by Jackendoff (1992). Further, building on techniques used by Lewis (1993), Rytting (2000), and Lonsdale and Manookin (2004) this thesis presents a new approach for incremental sentence processing. By leveraging the capabilities of our cognitive architecture to create syntactic and semantic representations of a sentence, XNL-Soar can address many types of ambiguity.

Research Question

Can XNL-Soar, as described in this thesis, resolve ambiguities common in language and perform natural language processing tasks, namely: disambiguation of prepositional phrase attachment and disambiguating progressive versus copula+gerund constructions?

Chapter Preview

If the reader wishes a more thorough review of sentence processing and cognitive modeling than is possible in chapter 2 please see the following publications: sentence processing (Jurafsky and Martin, 2000), symbolic processing and cognitive architectures (Newell, 1990), Soar (Laird, 2012), and NL-Soar in Lewis (1993). In chapter 3's discussion of XNL-Soar Chomsky (1993) and Adger (2003) provide further insight to the Minimalist Program while Jackendoff (1992) gives the foundation for Lexical Conceptual Structures.

Chapter 2: Survey of Literature

Using computers to process language and to address ambiguity has been an active area of research for over half a century. Some of the first attempts to process language with computers took place in the early 1950s with the proposal of what became known as the 'Turing Test' (Turing, 1950). Shortly afterwards, researchers at Georgetown created a system to automatically translate Russian into English and famously claimed that machine translation would be a solved problem in 3 to 5 years (Hutchins, 1999). While the 3-5 year period has come and gone, machine translation and other areas of natural language processing (including speech recognition, sentiment analysis, parsing and discourse analysis) continue to grow in the research community. The Association for Computational Linguistics's website [1] comprises a sample of the variety of venues currently reporting such research.

2.1 Sentence Processing

Reading the newspaper or having a conversation can be an enjoyable experience in part because of how wholly unaware we are of the cognitive work happening in understanding individual sentences and words. Sentence processing involves the rapid integration of many different types of information (lexical, syntactic, semantic, etc.). A principal focus of research in sentence processing is ambiguity. As mentioned previously, ambiguity is ubiquitous but so easily resolved that it is rarely even noticed.

2.1.1 Ambiguity

There are many different types of ambiguity; examples 1.1 and 1.2 illustrate lexical and structural ambiguities respectively (Oaks, 2012). Another way to describe ambiguity is global versus local. A sentence is globally ambiguous if it has two distinct interpretations. Example 1.3 is globally

[1] http://www.aclweb.org/anthology/

ambiguous: does the cop or the criminal have the telescope? Unlike global ambiguity, local ambiguity persists only until enough information has been received subsequently to resolve it, so that the final utterance has only one possible interpretation. For example the sentence:

(2.1) '*The critic wrote the book was delightful.*'

is locally ambiguous after having reached the word '*book*'. Did the critic write the book? The ambiguity ends after reaching '*was*'; only one interpretation is now possible. When processing a local ambiguity humans select a possible interpretation immediately, without waiting to read or hear more words that might later disambiguate the sentence. This behavior is the result of incremental processing (Crocker and Brants, 1999). This means that we assign structure to words immediately as they are perceived, rather than storing them up and waiting for a break in the input to carry out processing.

One of the reasons processing happens this way is humans' limited short term memory capacity. To illustrate this, try to memorize a list of twelve random words versus a twelve word sentence. Even complex sentences are easier to remember than random words because structure can be assigned as we hear each word.

2.1.2 Memory

Memory is the capacity to encode, store, and retrieve information. It is generally agreed that there are two different types of resources, short term memory (STM) and long term memory (LTM). However, there is evidence to suggest that there are different types of memory within LTM. One of those proposed is semantic memory. Semantic memory refers to the stored knowledge of the world that underlies the capacity to understand language. It should also be noted that memory is a finite resource that has limits. As utterances are processed and information is passed back and forth from STM to LTM, if these limits are reached, problems in processing arise.

2.1.3 Lexicon

At its most basic the lexicon can be thought of as a simple list or repository of words. The lexicon is, however, more complex than that. It contains information on its entries including phonology, semantics, morphology and even syntax. The phonological component can describe multiple possible pronunciations of a given word while the semantic information can provide the different meanings (or senses) available (Melby et al., 2007). Some discussion focuses on exactly what morphology is contained in the lexicon. The question is whether all forms of a word are stored individually, or if those forms that can be accounted for using a small set of rules form a group related to a lemma or baseform. It is generally accepted that inflected forms are based on a word's lemma because they are usually predictable and do not usually result in a change of grammatical category. Alternatively, because derivation can produce a change in meaning and possibly a change in category, words created using derivational morphology would be separate entries in the lexicon. A major research area is the process by which words are accessed and retrieved during sentence processing: how much information is retrieved, which pieces are brought to bear more rapidly, and how ambiguity is processed during retrieval (Onifer and Swinney, 1981; Caramazza et al., 1988). These questions will be revisited in section 3.2.2, where lexical access in XNL-Soar is described.

2.1.4 Linguistic Approaches and Representations

Experimental research has generated a large number of hypotheses about the architecture and mechanisms of sentence processing. Issues like modularity versus interactive processing and serial versus parallel computation are just a few of the unanswered questions in computational and theoretical linguistics. A modular view of sentence processing assumes that each system involved in sentence processing (phonetics, morphology, syntax, semantics) is its own self-contained module, with limited communication to any other module. For example, syntactic processing and structure building take place independently of semantic analysis or context-dependent information. One common assumption of modular theories is a feed-forward architecture, meaning the result of one processing step is passed on to the next step. The interpretive hypothesis takes syntactic processing

to be a precursor to semantic processing and discourse/pragmatic operations. Alternatively, interactive accounts assume that all available information is processed by all modules simultaneously and can immediately influence the final analysis.

Another theoretical question centers on serial versus parallel processing (Gorrell, 1989; Boston et al., 2011). Serial sentence processing assumes that only one of the possible interpretations is created first, and other possibilities are generated only if the first one turns out to be incorrect. In systems using parallel processing, multiple interpretations are ranked; furthermore, it is assumed that humans are aware of only one possible analysis, derived from the highest ranking possibility.

2.2 Mechanisms and parsing

In implementing computational models of sentence processing, the component that carries out syntactic processing is called a parser. There are many different types of parsers, from bottom-up to top-down. Most employ a set of rules called a grammar that specify a set of well-formed constructions in the language.

(2.2) '*Does this flight include a meal?*'

S ⇒ NP VP	Det ⇒ that, this, a
S ⇒ Aux NP VP	Noun ⇒ book, flight, meal, money
S ⇒ VP	Verb ⇒ book, include, prefer
NP ⇒ Det Noun	Aux ⇒ does
NP ⇒ Noun	
VP ⇒ Verb	
VP ⇒ VP NP	

Table 2.1 A basic grammar.

Example 2.2 provides an example sentence that table 2.1's basic grammar can account for. Figure 2.1 illustrates how example 2.2 would be parsed following the simple grammar. However, a true parser's grammar would be much larger and contain far more complexity in its rules.

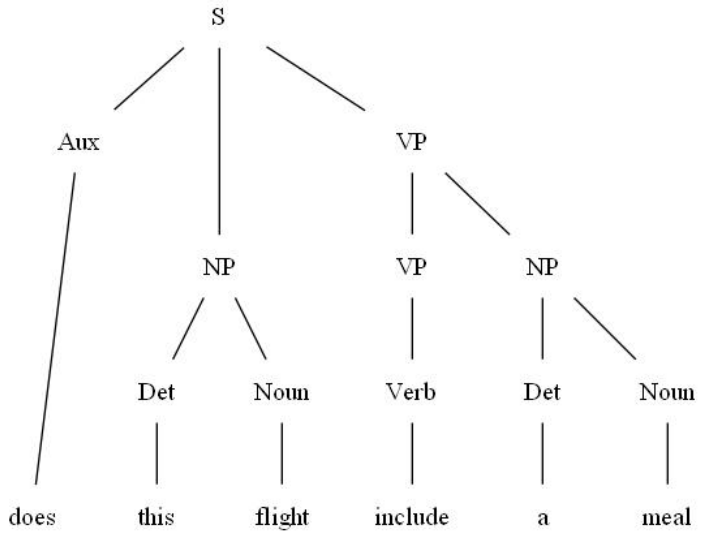

Figure 2.1 Example 2.2 after parsing with table 2.1's grammar.

Different parsers employ different fundamental strategies. A top-down parser, sometimes termed a "goal directed search," attempts to build structure by parsing first at a root node (e.g. sentence) and then building structure down to the leaf nodes. Following grammars, trees are grown downward until they eventually reach part-of-speech categories and lexical items at the bottom. Bottom-up, or "data-driven search" is the earliest recorded parsing strategy. First suggested by Yngve (1955), this type of parse begins with the lexical items and builds trees upward following a grammar from the words up to a root node S. See Jurafsky and Martin (2000) for a more detailed description of these and other parsing strategies.

Words like '*book*,' '*slip*' and '*crew*' can be used as both nouns and verbs, exhibiting lexical category ambiguity. Some sentences like '*Jump the log.*' or '*Book that flight.*' show local ambiguities. While the sentence '*Book that flight.*' is not ambiguous, a parser must wait until reaching the word '*flight*' to know whether '*book*' will be a noun or verb. Sentences like '*I shot an elephant in my pajamas.*' and '*I saw the man with a telescope.*' contain ambiguities that are structural. These and still more types of ambiguity cause parsers difficulty in particular ways. In bottom-up parsers, problems with ambiguity arise when all possible combinations of words and their parts of

speech are considered, which can quickly become unwieldy. While top-down parsers do not waste time with all possible subtrees, they can reach an untenable state creating trees for ungrammatical sentences. Neither of these approaches can address these issues of ambiguity entirely on their own.

Strategies for dealing with issues of ambiguity and other complications of language parsing are numerous, including dynamic programming (Earley, 1983), bottom-up filtering (Blache and yves Morin, 1990), probabilistic reasoning (Jurafsky, 1996) and depth-first search (Knuth, 1997). In view of this research the parsing strategies above lack two key components: the ability to work incrementally and a cognitive foundation (section 2.3).

A parser that works incrementally does not wait for the complete sentence, but instead builds as much structure as possible at any given point in the input. While the parsers outlined above can comply with incremental principles, many parsers wait for a complete utterance before 'tokenizing' the utterance, or breaking it into its constituent parts. Incremental parsers can work in several ways; Hale (2004) outlines how one can use probability to inform a parser's expectations of what might follow. Ball et al. (2009) use custom grammar constraints in the ACT-r cognitive architecture to generate a parse that is updated as each word enters "memory."

2.3 Cognitive Modeling and Cognitive Architectures

2.3.1 Incremental Processing

The field of cognitive modeling, in which this thesis is situated, has settled upon three principal strategies for parsing syntactic ambiguities incrementally:

1. Serial models: A single 'preferred' structure is computed for an ambiguous string. If this structure is incompatible with subsequent material, reanalysis is required.

2. Parallel Models: Multiple structures are computed for an ambiguous string. At points of disambiguation in the parse, untenable structures are deleted.

3. Minimal Commitment Models: Commitment to a given structure is postponed until disambiguating information becomes available.

Table 2.2 summarizes serial, parallel and minimal commitment models.

In a serial model, a single syntactic structure is computed for both ambiguous and unambiguous sentences. When an ambiguity is encountered, a decision must be made as to how to integrate it into the structure. Two possibilities arise when this occurs. First, the ambiguity is integrated following the preferred reading and the sentence is processed as if there were no ambiguity. Second, the ambiguity is integrated following the unpreferred reading. This situation causes a problem, either immediately with the next word or farther down in the utterance, with integrating subsequent lexical items. This failure causes reanalysis to happen in order to correctly integrate the words of the sentence. Some literature calls this a 'first pass' failure requiring a 'second pass' to successfully parse the sentence.

	Serial	*Parallel*	*Minimal Commitment*
Start of Ambiguity	single structure computed	multiple structures computed	underspecified structure computed
Ambiguous Region			
- preferred reading	computed; same as unambiguous counterpart	computed; ranked highest	subsumed under underspecified structure
- unpreferred reading	not computed	computed; ranked lowest	subsumed under underspecified structure
Point of disambiguation			
- preferred reading	no reanalysis	pruning of lower ranked structure	full specification
- unpreferred reading	reanalysis	pruning of higher ranked structure	full specification

Table 2.2 Summary of serial, parallel and minimal commitment models (Bader and Bayer, 2006, p. 33)

In a parallel model, multiple structures are computed immediately upon encountering an ambiguity. Once an ambiguity is processed, multiple structures are maintained until enough information is available to decode the ambiguity. Once the disambiguating information is integrated into existing structure, the structures that have now become irrelevant can be pruned (deleted) out of

memory. With this model, there is no need for reanalysis, but generating all possible structures can be expensive.

In a minimal commitment model, ambiguous and unambiguous sentences are again treated differently from the appearance of an ambiguity onward. If no ambiguity is encountered, this model will commit to a syntactic structure as quickly as possible. However, once it has encountered an ambiguity, it must create a representation that is underspecified so that all possibilities are maintained and available. This could mean delaying integration until disambiguating information is processed. But minimal commitment models most often employ strategies to 'commit' to certain pieces of structure while leaving other portions unspecified. Reanalysis in this model consists of filling those areas of structure left unspecified during a 'first pass' parse.

The descriptions above define 'pure' models of parsing (Gorrell, 1991). Many of the implementations of these models are better described as hybrid systems (this includes XNL-Soar), combining elements from all three to achieve the desired result.

2.3.2 Cognitive Modeling

Researchers in cognitive psychology have generated many theories pertaining to specific areas of human cognition, including hand-eye coordination, typing, memory usage and even fatigue. This research has generated large amounts of information about the mind's abilities and many theories about individual cognitive tasks, but has never consolidated the research into a single account of human cognition. In his seminal work on unified theories of cognition, Newell (1990) calls on the field to work towards that goal.

The many theories mentioned above are, as a general rule, implemented as cognitive models. A cognitive model is intended to be explanatory of some area, or task, of cognition: how the task can be carried out by a set of computational processes. A model makes predictions about behavior that can subsequently be compared to human performance data. Additionally, the great majority of these models are implemented inside a cognitive architecture.

Cognitive architecture refers to the design and structure of the mind. Theories of cognitive ar-

chitecture strive to provide a comprehensive framework for cognitive systems, a description of the functions and capacities of each, and a blueprint to integrate the systems. A cognitive architecture itself is a fixed set of computational mechanisms and operations. It consists of a central processing unit, a memory unit, and input and output units. Information is received, stored, and processed algorithmically to produce an output, or action. An architecture specifies how actions and reactions happen based on available data, how knowledge is stored, and the control structures used for processing. By creating a cognitive architecture to instantiate theories of human cognition, we can make more systematic, and theoretically based, predictions about cognition (Lewis, 1993). The final piece is content: for an architecture to mean anything, something has to be processed. This processing is the simulation or model of interest, defined by Laird (2008) as:

$$\text{Behavior} = \text{Architecture} + \text{Content}.$$

One of the first architectures developed was a production system. A production-based system consists of a working memory, a set of production rules, and a set of precedence rules determining the order for firing the production rules. A production rule is a condition-action pair that specifies actions to be carried out when certain conditions are met. The first general theory of this type was called the General Problem Solver (Newell et al., 1959). The idea was that a production system incorporating a few simple heuristics could solve many problems in a similar way to humans. These rules create an agent that, using the mechanisms provided by the architecture, is situated in an environment, responding to input and producing output to effectively model human behavior.

Symbolic modeling grew out of the computer science paradigms using the techniques of knowledge-based or expert systems. The requirement is that the system be capable of manipulating and composing physical patterns with processes that give the patterns the ability to represent either external entities or other internal structures (Newell et al., 1981; Newell, 1990; Simon, 1996). Symbolic cognitive models are characterized by: detailed accounts of chronometric data and error patterns; explorations of the explanatory role of the same basic architectural components across a range of cognitive tasks; attempts to clearly distinguish the contributions of relatively fixed architecture;

and flexible task strategies and background knowledge. One of the first and most developed architectures of this type was Soar.

Subsymbolic or connectionist approaches are based on the assumption that the mind/brain is composed of simple nodes and the power of the system comes primarily from the existence and manner of these connections between the simple nodes. Detractors of this approach contend that these models lack explanatory powers because complicated systems of connections are often less interpretable than the systems they model. Newell and Simon (1976) argue that, "A physical symbol system has the necessary and sufficient means for general intelligent action." That is to say, a symbolic system is capable of modeling human behavior.

The underlying goal of all these approaches is to produce more unified accounts of cognition explicitly embodied in computational mechanisms.

2.3.3 Soar

In Unified Theories of Cognition, Newell (1990) proposed Soar as a comprehensive account of human cognition. More importantly, Newell not only proposed the theoretical background and foundation for Soar, but also implemented it as a cognitive architecture. Soar is made up of mechanisms for decision making, for recursive application of operators to a hierarchy of goals and subgoals, and for learning of productions. The architecture has been applied to help understand a range of human performance from simple stimulus-response tasks, to typing, syllogistic reasoning, and more. The Soar cognitive architecture is a production system using symbolic processing and sub-goaling. Problems are decomposed into progressively smaller sub-goals until a solution is reached and then saved as a "chunk," available to be accessed the next time Soar encounters a similar problem. By using this process Soar mimics a human's ability to learn. The central mechanisms and structures specific to Soar—its memory structure and decision cycle—are summarized below.

Memory

Soar's working memory elements (WME's) are the basic structure that contains all the information about the current situation, including perception and encoded knowledge. A Soar production is like a conditional "if-then" statement, consisting of a set of conditions determining when it will fire and a set of actions (or changes) it will produce when it does fire. The conditions are true or false based on the WME's contained in Soar's state. Operators are bundles of these rules (productions) that modify the information contained in the current state.

Long-term memory is the store for content processed by the architecture's operators to produce behavior. Soar supports three types of long-term memory:

- Procedural memory is the set of productions accessed automatically by Soar during its decision cycle to change state and accomplish its goals. These are hand-coded and represent the different actions that the system will carry out.
- Semantic memory is a store of declarative structures that can only be accessed by specific productions using cues.
- Episodic memory contains knowledge encoded as episodes or snapshots of the current state that can only be matched against using specialized cues.

In all three cases Soar cannot access long-term memory directly, but only by firing an operator based on its productions. Semantic memory will be addressed further as relating to its implementation in a task-specific strategy (Section 4.1.3).

Decision Cycle

The perception/motor interface maps the external world to the internal representation and back. Through this interface, perception and action occur together with cognition.

The decision cycle is the basic architectural process underlying Soar's cognition. The cycle is made up of five phases:

1. Input takes information from the outside world (perception/motor interface) and allows Soar to add it to the current state.

2. The elaboration step provides access to procedural memory in order to elaborate the current state, suggest new operators, and evaluate those that have been proposed. The end of this phase is signalled by quiescence.

3. The decision phase examines the operators that have been proposed and their assigned preferences. This examination results in either selection of an operator or an impasse if the preferences are incomplete or in conflict (e.g. a tie when two operators have the same preference).

4. Following a decision comes the application phase where the selected operator fires and modifies the state. By selecting a single operator each cycle, a cognitive bottleneck is imposed by the architecture creating a limit on how much work cognition can achieve.

5. The final step in a decision cycle is output, which allows actions from cognition to change the outside world.

The impasses mentioned above are important in that they signal an inability to proceed, forcing Soar to use learning to continue. Some impasses can, however, be avoided by specifying operator preferences. Preferences, and their effect on operator selection, will be discussed further in Chapter 3.

2.3.4 NL-Soar

In Unified Theories of Cognition, Newell addressed the subject of language carefully: "Language should be approached with caution and circumspection. A unified theory of cognition must deal with it, but I will take it as something to be approached later rather than sooner." Within only a few years, the Natural Language Soar (NL-Soar) research group was formed at Carnegie Mellon. NL-Soar was implemented as an additional module within the Soar cognitive architecture and performed incremental (word-by-word) syntactic and semantic processing (Newell, 1988). An early

NL-Soar publication in 1991 was a CMU tech report (Lehman et al., 1991) modeling syntactic processing with dependency syntax (Mel'čuk, 1988), later updated to use X-bar theory (Jackendoff, 1977). This first model showed promising results and included lexical, syntactic, semantic and pragmatic productions to create dependency parse trees.

In a subsequent tech report the NL-Soar system was evaluated empirically based on its ability to match human parsing ability (Lewis, 1992). The system was designed not only to parse sentences successfully, but also to fail to parse them when human performance data indicates that humans encounter parsing breakdown.

Garden path sentences contain ambiguities that cause parsing to break down in humans, while still being grammatically correct. The classic example was from Bever (1970): "The horse raced past the barn fell." Lewis (1992) explored the different assumptions about sentence processing that could be modeled and their effect on parsing garden path sentences. This report was not a full implementation of NL-Soar, but outlined some important requirements, specifically those for a repair mechanism (later called the snip operator). Figure 2.2 outlines those requirements.

1. Must permit comprehension of non-ambiguous sentences.

2. Must fail on garden path sentences.

3. Must be correct. Repair must not produce interpretations inconsistent with the utterance. Repair must ultimately be bound by the same grammatical constraints that guide comprehension generally.

4. Must work without reprocessing input.

Figure 2.2 Requirements for a repair mechanism (snip operator).

Lewis followed up with a full implementation of NL-Soar in his dissertation (Lewis, 1993), which is still considered a foundational work. He successfully addressed multiple ambiguities including garden paths, acceptable and unacceptable embeddings, and other parser breakdowns. The output of NL-Soar's linguistic processing was syntactic and semantic structures. The syntac-

tic structure was called the utterance model, and the semantic structure was called the situation model. While humans regularly comprehend sentences that are ambiguous both syntactically and semantically, Lewis focused on structural ambiguity. In NL-Soar, structural ambiguity was resolved using the snip operator (Lewis, 1998), which allowed the restructuring of syntactic trees. Lewis called this *limited cue-driven repair*: cue-driven because it relied on a few simple local cues to be triggered, and limited because it classified structure types as potential garden paths, not definite garden paths. Another innovation he implemented was an integral control structure for syntactic parsing. The idea was to index node pairs by the potential syntactic relations that they might enter into. The structure is called the A/R set, for assigners and receivers. Figure 2.3 shows the contents of the A/R set during processing '*John hit the ball*', after the NP '*the ball*' has been created but not yet attached as a complement of '*hit*'.

ASSIGNERS	adjoin-N':	[$_{N'}$ *ball*]
	comp-V':	[$_{V'}$ *hit*]
	adjoin-V':	[$_{V'}$ *hit*]
RECEIVERS	comp-V':	[$_{NP}$ *the ball*],[$_{CP}$ *John hit*]
	adjoin-IP:	[$_{CP}$ *John hit*]
	spec-IP:	[$_{NP}$ *the ball*]

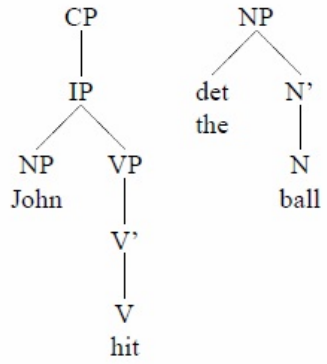

Figure 2.3 The A/R set during processing '*John hit the ball*'; '*ball*' can assign an adjoin-N', '*the ball*' can receive the comp-V' relation and so on (Lewis, 1993, pg. 84).

Many of the garden path phenomena could be explained by limiting the size of the A/R set's index to two words per syntactic class (Lewis, 1996). This size limit was not a hard and fast rule, but a parameter that could be affected by working memory.

Though Lewis' thesis is still cited as the foundational work for NL-Soar, development has been continued by Lewis and others. In 1994, Rubinoff and Lehman published a report on integrating NL-Soar into agents which model human behavior in specialized tasks (Rubinoff and Lehman, 1994). The first, NASA Test Director Soar (NTD-Soar), modeled the interaction of the NASA test

director with other engineers during preparation for shuttle take-off. The second, Tac-Air Soar, modeled jet pilots flying on various tactical missions and interacting with other aircraft. Both of these required language generation in addition to comprehension, made possible by the Soar cognitive architecture. The culmination of this research was using NL-Soar in the Department of Defense funded *Synthetic Theater of War* (STOW) in 1997, which was the first large-scale war simulation, involving thousands of agents cooperating through linguistic interaction (George et al., 1999).

Following the Lewis (2000) publication on how to specify a complete linguistic architecture, development and use of NL-Soar has been limited to the NL-Soar research group at Brigham Young University. Lewis has continued to work on cognitive modeling of language, including using another cognitive architecture, ACT-R (Vasishth and Lewis, 2004, 2006). One of the most significant changes to NL-Soar was its integration with WordNet, an electronic lexical database designed to model natural lexical relations (Fellbaum, 1998). WordNet contains some 91,000 concepts, giving glosses, examples and grammatical information. More than a simple dictionary, WordNet addresses issues like polysemy and synonymy and is explicitly designed upon psycholinguistic principles. Words are not stored according to spelling or sound, but by meaning, with sets of synonyms grouped together into synsets. In addition to containing words and relationships, WordNet relates words to concepts in two ways. First, it breaks each word down into a number of discrete senses. Second, it groups synonyms by those word senses (i.e. synset groupings) of different words that are nearly equivalent. WordNet helped NL-Soar solve its problem of lacking vocabulary, but also introduced the new problem of massive semantic/syntactic ambiguity given WordNet's size (Rytting and Lonsdale, 2001). However, NL-Soar scaled up to the increased vocabulary size and performed well on NLP tasks similar to word sense disambiguation (Rytting, 2000).

Chapter 3: XNL-Soar

XNL-Soar is a specialized implementation created within the Soar framework. The version of XNL-Soar described in this thesis builds on prior research (Rytting, 2000; Lonsdale, 2001; Lonsdale et al., 2008) based on what is discussed in section 2.3.4. In this thesis I describe the updates we have made to the operators creating syntactic structure and the improved coverage of syntactic phenomena. I also describe the addition of semantic structure building. In a recent update to the Soar architecture (Derbinsky and Laird, 2010), a semantic memory capability was added by others. This thesis also details our subsequent implementation of semantic memory in XNL-Soar and describes two experiments utilizing it.

The current instantiation of XNL-Soar implements 19 operators, which in turn involve 315 Soar productions. In section 3.2 below I explain each of these operators and how they work together to process language and create linguistic representations. Section 3.1 will provide details of the syntactic and semantic representations used by XNL-Soar.

3.1 Linguistic Formalisms

3.1.1 Syntax: Minimalism

Similar to the implementation by Lehman et al. (1991), XNL-Soar began with a 'middle-of-the-road' X-bar syntax as its syntactic representation (Jackendoff, 1977; Chomsky, 1986). In 2006, the development of NL-Soar was updated to XNL-Soar, principally to incorporate the use of Minimalist syntax (Lonsdale and Manookin, 2004; Lonsdale et al., 2006, 2008). Figure 3.1 illustrates the basic structural template for X-bar syntax. X-bar structure is organized around a head and its resulting projections. A head X (a lexical item) combines with one maximal projection, its complement, to form the intermediate projection, ("X-bar"), or X'. The X-bar combines with another maximal projection, its spec(ifier) to form the maximal projection, represented as XP. X can be

any syntactic category: either a lexical category V(erb), N(oun), A(djective), P(reposition); or a functional category such as I(nflection) or C(omplementizer). An adjunct is an optional part of a sentence that, if removed, will not affect the sentence except to delete unnecessary information. In XNL-Soar, both A and P can be adjoined to existing structure.

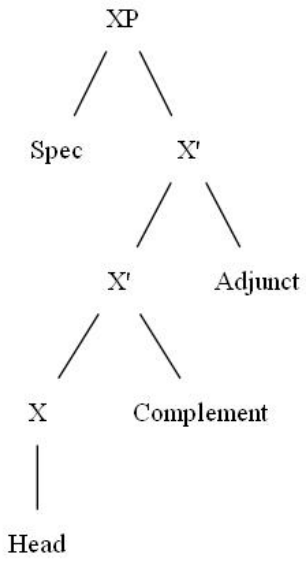

Figure 3.1 Basic X-Bar template

In our development of XNL-Soar, we use the Minimalist Program (MP) as the underlying theoretical foundation for our syntactic representation (Chomsky, 1995; Adger, 2003). The MP is one of the most widely accepted theories of syntax, though many skeptics remain. The MP provides a framework to contain grammatical phenemona and combine them as a syntactic tree. The MP builds on previous iterations of syntactic theory from the X-bar of Government and Binding (Chomsky, 1993) to Principles and Parameters (Chomsky and Lasnik, 1993), and is widely used in contemporary linguistics. Figure 3.2 defines the clausal and nominal hierarchy of projections given by the MP (Adger, 2003). It illustrates that the lowest node in the clausal hierarchy is the lexical node V. Above V the nodes are given precedence, indicated by >. Optional nodes are in parenthesis: (Neg)ative, (Perf)ective, (Prog)ressive and (Pass)ive. These nodes are not required to parse a sentence and are generated only if the contents of the sentence warrants their use. The nominal hierarchy represents the lowest node as the lexical N. It also provides the optional nominal

node (Poss)essive.

```
Clausal: C > T > (Neg) > (Perf) > (Prog) > (Pass) > v > V
Nominal: D > (Poss) > n > N
```

Figure 3.2 Hierarchy of Projections.

As an example, with the sentence '*The woman will see the man.*', Lasnik (2002) describes processing as beginning by combining the noun (N) '*man*' and the determiner (D) '*the*' to create a determiner phrase (DP) '*the man*'. The verb (V) '*see*' then combines with this DP to produce an intermediate projection, V-bar or V'. The DP '*the woman*' is created exactly as the previous DP and combines with the V-bar to produce a verb phrase (VP). Next, this VP merges with the inflection (I) node to create an I-bar node. Finally, the DP '*the woman*' attaches to the specifier(spec) position of the inflection phrase (IP) to generate the completed parse.

In response to Lasnik's (2002) summary of the MP, Edelman and Christiansen (2003) take issue with the MP and generative grammarians in general for not providing sufficient empirical evidence to support the claims they put forward: "Unfortunately these foundational issues have not been subjected to psychological investigations, in part because it is not clear how to turn the assumptions into testable hypotheses." They conclude, "it would be in the best interests of linguistics and of cognitive science in general if the linguists were to help psychologists like ourselves to formulate and sharpen the really important foundational questions, and to address them experimentally. This, we think, would help cognitive scientists take Minimalist syntax more seriously." These claims are perhaps biased, but the point is valid and clearly more research is needed to test the limits and capabilities of the MP. XNL-Soar is a way of gaining empirical evidence in support of or in opposition to the MP. Not only does XNL-Soar test the validity of the program, but it also illustrates *how* it works, because the inside of the model is observable. The productions in the Soar code are an explicit representation of how the MP can be mapped to operators and account for human sentence processing. This thesis seeks to establish a framework for testing the limits of the MP in a principled, computational manner.

Figure 3.3 contains a full example sentence and its matching syntactic stucture following the MP, similar to that described by Lasnik (2002). The DP '*my favorite cousin*' is created by combining the NP '*cousin*' as a complement of the DP with the root D '*my*'. The adjective '*favorite*', having been projected to an AdjP is adjoined at the N-bar level to complete the DP. The verb '*singing*' is projected to a VP and following Adger (2003), is projected to a verb shell. Because '*was*' is being used as an auxiliary verb in this sentence it is projected to a progressive phrase (ProgP). The main verb '*singing*' is attached as a complement at Prog' and an IP is subsequently projected. Once the DP is attached in the spec position of the IP the MP parse is completed.

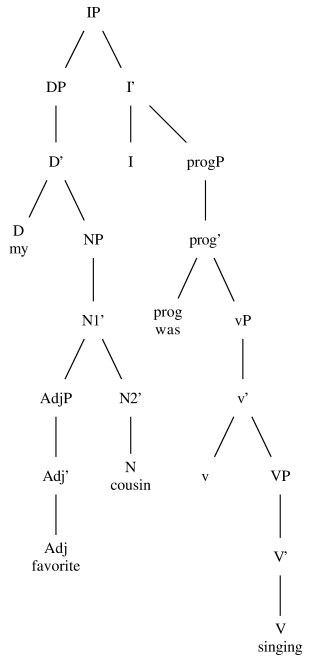

Figure 3.3 A syntactic tree following the MP for the sentence '*My favorite cousin was singing.*'

3.1.2 Semantics: Lexical Conceptual Structures

A lexical conceptual structure (LCS) is a compositional semantic abstraction with language-independent properties that transcend structural idiosyncrasies (Jackendoff, 1992). An LCS combines the semantics of individual lexical items to create a representation of an utterance through a combination

of semantic structure (specified by the shape of the graph and its structural primitives and fields) and semantic content (specified through constants). Figure 3.4 shows an example LCS structure for the sentence, '*A boy gave the book to his sister with the very ugly shovel.*'. The root of this structure is the verb '*give*' with a semantic category 'v-possession'. The verb links several different arguments with their labels: E, I, I2 and P. The external argument (E) contains the argument external to the verb (i.e. its subject). The internal argument (I) contains an argument internal to the verb, which in this case is the direct object, the book being given. The internal2 argument (I2) is the indirect object, the sister to whom the book is being given. Finally, the property argument (P) provides additional information about the predicate: in this example sentence, the manner in which the book was given (with a shovel), which is an internal argument of the instrumental property.

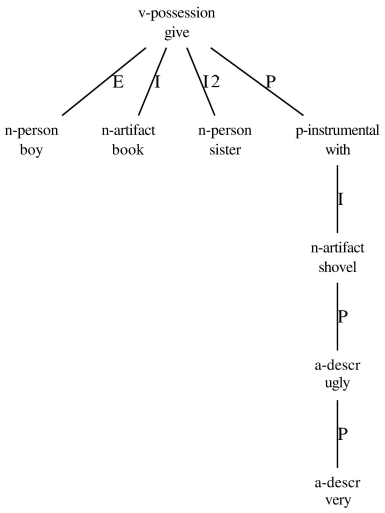

Figure 3.4 A lexical conceptual structure for one construal of the sentence '*A boy gave the book to his sister with the very ugly shovel.*'

Additionally, it is an ugly shovel, shown as another P of the shovel, and the ugliness has an intensifier 'very' represented as a property. Most importantly, the LCS in figure 3.4 is not ambiguous, in spite of the fact the sentence it illustrates is. The LCS illustrates one possible construal of the sentence, that in which the boy gives the book to his sister using the shovel. For the LCS to illustrate the other possible construal, in which the sister has a very ugly shovel, the P containing '*with*

the very ugly shovel' would be attached to the '*sister*' node instead.

3.2 Operators

Figure 3.5 provides a rough overview of how an XNL-Soar agent *might* apply its operators during processing. XNL-Soar's 19 operators can be grouped into several categories, the largest of which consists of 10 operators that create syntactic and semantic structure. The next largest grouping contains 4 operators that check a parse's validity and completion. When an agent is initialized, two operators fire, one to set up the linguistic formalisms and the other to interact with the environment, to input the words from the sentence to be parsed. The **getWord** operator performs all of the different manipulations in carrying out lexical access for each word; section 3.2.2 elaborates this process. Finally, there are two 'bookkeeping' operators which apply after any structure is built or undone. Each operator will be described in the remainder of section 3.2. It is important to remember that 3.5 is not a true representation of how XNL-Soar carries out a parse, but a simplified generalization to provide the reader with a point of reference for the following discussion on operators. Appendix A contains a hand-annotated trace of operator firings as they are in fact proposed and executed. Section 3.3's example sentence will illustrate the operator's function and relate it to specific operator cycles in appendix A.

3.2.1 Agent Initialization

Upon initialization of a new agent in the Soar architecture, the first XNL-Soar operator that fires is called **annotateHOP**. The purpose of this operator is to create working memory elements (WME's) in Soar that encode the hierarchy of projections (HoP) (Adger, 2003). This representation will be used by other syntactic structure building operators subsequently in the process of parsing an utterance. The **annotateHOP** operator involves only two productions, one to encode the clausal HoP and one to encode the nominal HoP.

The next operator that fires is **sentenceInputLink**. This operator matches the state of the agent after the HoP has been specified. This operator sets up an input buffer that allows the agent to

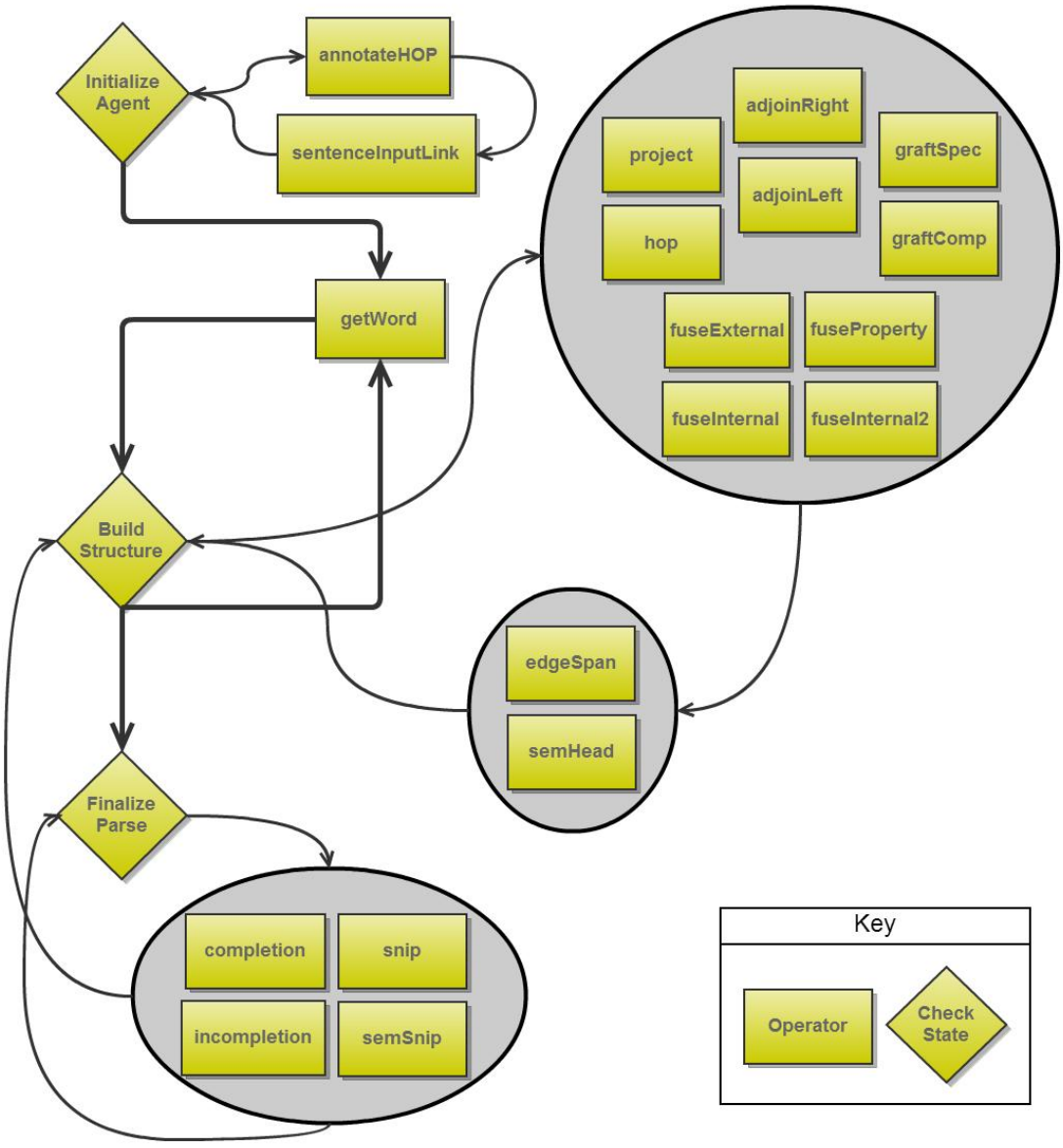

Figure 3.5 A rough generalization of processing in XNL-Soar

store words and create corresponding WME's. This operator involves 5 productions, one to create WME structures that make up the auditory buffer, 3 to create the WME's necessary to represent the word, and one to check for the end of an utterance (signaled in our textual input by a period or question mark).

3.2.2 Lexical Access

The next step for the agent is lexical access via an operator called **getWord**. This operator is considerably more complex than the previous two, involving 75 productions. First, the system copies a word from the input link into memory and creates the basic structures to annotate the word's lexical information. The system then determines whether the word being accessed is contained in WordNet (Fellbaum, 1998). WordNet is a very helpful lexicon, but does not contain information on certain types of words. If the word being processed is a pronoun, complementizer, punctuation, preposition, auxiliary verb, negative adverb, or conjunction the system creates a WME that flags the word as unavailable in WordNet. These same productions will also annotate the word with the applicable category (e.g. determiner or preposition). On the other hand, if the word is in WordNet, we retrieve the part of speech (POS) and create WME's. At this point some very basic morphological information is computed and stored and WME attributes for morphology are created. WordNet's data is then checked for sense information. For each POS attribute a WME substructure is created for each sense delineated in WordNet.

For example, the word '*have*' has a single noun sense (i.e. the haves and have-nots) but more than 20 verb senses. WordNet sentence frames are retrieved for each noun and verb sense. Additionally, we create a correspondence table labeling each frame as intransitive, transitive or ditransitive (see WordNet sentence frames).[1] WordNet also labels each sense with a semantic class (see WordNet semantic classes).[2] By using these two pieces of information instead of creating a WME for all 20 senses of '*have*' only the 12 unique pairings—called semantic classes—are generated. The most common verbal sense of '*have*' WordNet labels as v-possession. Sentence frames

[1] http://wordnet.princeton.edu/man/wninput.5WN.html#sect4

[2] http://wordnet.princeton.edu/man/lexnames.5WN.html#sect4

indicate that the verb can be either transitive or ditransitive.

By generating each word's POS sense attributes this way, some of the ambiguity introduced by using a comprehensive resource like WordNet is effectively reduced. Once WME's have been created for each word's POS and sense combinations, the system populates the assigners/recievers set (A/R set) with the possible syntactic relations. First outlined by Lewis (1993), this mechanism serves to constrain what would otherwise become a rapidly expanding list of possibilities.

After populating the A/R set, syntactic objects (synObj's) are generated. These synObj's are WME's that will become trees eventually comprising the utterance model (syntactic representation). A synObj is generated for each lexical category of a word. So with the word '*have*' three synObj's would be generated, one each for the noun, verb and perfective categories. At this point in access semantic objects (semObj's) are created. Like synObj's, semObj's are nodes that will build the situation model (semantic representation) of the input. A semObj is created for each unique semantic class of a word; in the case of a verb, each unique pairing of the verb's semantic class and sentence frame. Because this process generates so many more semObj's than synObj's one additional step is required. WordNet provides a rank for the semantic class of each sense of a word, so the highest-ranking class for each sense is given a WME labeling it the "current hypothesis." This label means that when generating structure in the situation model the semObj labeled as current will be selected over other possibilities.

After creating the synObj's and semObj's, syntactic and semantic profiles (synPro's and semPro's respectively) are created. One synPro is created for each category of a word. For a verb that is ambiguous between a tensed form and an infinitive, a production generates an additional synPro to account for the latter possibility. Every synPro has a category and a pointer to its corresponding synObj. The relationship between a synObj and a synPro can be one-to-one or one-to-many. Next, semPro's are created for each semantic class identified for the word in WordNet. Like synPro's, semPro's are created with pointers to their corresponding semObj. The relationship between semObj and semPro can also be one-to-one or one-to-many. After creating both semPro and synPro we point the semObj to their corresponding synObj and vice versa. The addition of these pointers

creates a syntax-semantics interface that allows information to be passed from one to the other. In this thesis the path is in only one direction: syntax informs semantics. However, the creation of the interface will allow further investigation and research relaxing this constraint. The final step of **getWord** is to create a WME on the word flagging access as completed.

Once lexical access is complete, the system continues parsing the sentence. As outlined earlier in section 2.3.3, the decision phase of the decision cycle involves preferences that determine which operator will fire next. In the application phase after the **getWord** operator fires, another **getWord** operator is proposed to accept the next word available on the input link. In addition, after lexical access is complete for a word, one or more **project** operator(s) are possible and thus proposed. Because multiple operators are proposed, Soar now has to determine which will fire first. This situation necessitates the need of special productions that Soar uses to compute operator preferences. When both **getWord** and **project** are possible, Soar prefers projecting current structure over getting a new word. This preference arises because XNL-Soar adopts an aggressive structure building strategy. This means that once a word's lexical access is complete, all structure that can be built will be built as soon as possible. As processing of a sentence proceeds and more operators become available, these search control productions become increasingly important.

3.2.3 Structure Building

Nine operators build structure in either the utterance or situation model (figure 3.5). The first of these is the **project** operator, which fires when lexical access has completed and an unprojected synObj is available. **Project** performs three actions: it creates two more nodes (the bar level and phrase level), links those nodes together, updates the A/R set by removing the original node and adding the new possible attachment sites, and adding the new structure as a synObj in Soar's memory state. Finally it annotates the synObj as having been **project**ed (only one synObj is **project**ed at a time).

After creating a new synObj representing a maximal projection for a given word, the next operator to be proposed is **HOP**, related to the Hierarchy of Projections (HoP). If the recently

projected synObj is a lexical category like N(oun) or V(erb), then **HOP** will match and generate more synObj's to create another level of structure. In the case of N (or V), **HOP** will first create 3 new synObj's for a shell projection. The noun and verb shells are a layer of linguistic structure described by Adger (2003). They are not particularly relevant to our discussion here, but are reflected in our structures to conform strictly to the MP. These nP (or vP) shell synObj's will be linked together and the NP (or VP) synObj created by the **project** operator will be linked to n' (or v') as a complement. If a noun was not preceded by a determiner (or is the first word input) then **HOP** will also create a determiner phrase projection, again consisting of three linked synObj's and attaching the noun (now with two levels of structure) as a complement to the determiner phrase. **HOP**'s preferences are that it will continue to create structure as long as it possibly can, for nouns no further than a determiner phrase and for verbs no further than a tense phrase. Additionally, as the **HOP** is creating new structure, the A/R set is updated (by adding and removing nodes) to maintain as possible attachment sites only those nodes that are still available. Search control preferences for **project** are such that it will not **project** any word while **HOP** could still possibly create more structure for a different word (or the same word with a different sense). This preference is in place to ensure that each structure is completed as much as possible before continuing on to the next possible structure building scenario. Finally it is important to note that XNL-Soar has several search control preferences implemented to rank any syntactic operator higher than any semantic one. This means that all available syntactic structure will be built and processed before any semantic structure operator will match and fire, assuring the precedence of syntax over semantics necessary for the interpretive approach.

After building some structure, Soar has to consider how to begin joining those pieces together; the first operator available for that is **graftComp**. This operator will match the state if there are two synObj's available: one having an X-bar (X') level node available in the assigner set and the other having a corresponding XP node available in the receiver set. **GraftComp** will then attach the two synObj's, one as a complement to the other, and update the A/R set accordingly. **GraftComp** has some important search control considerations. First, it is of lower priority than either **project** or

HOP, as it would be problematic to begin attaching structures if they were not complete. When multiple synObj's are available in the receiver set to be attached as a complement, the HoP WME is consulted and the lowest synObj is preferred. This preference is in place to ensure that any possible ambiguities are explored correctly. Furthermore, the synObj's available may belong to different words. If so, another preference is applied to prefer the word that is nearest to the attachment site. The final preference used by **graftComp** applies when the synObj's contain more than one lexical item; it selects the 'longest' synObj (the one containing more lexical items) for attachment.

The **graftSpec** operator is very similar to **graftComp**, except that it attaches one synObj as a specifier to the other, updating the A/R set accordingly. The preferences are also very similar, preferring closer and longer possible constituents, as well as lower members per the HoP.

Certain types of structures, like ditransitive sentences, require a specialized version of **graftComp** and **graftSpec** in order to place both objects at the same time. These operators match only when the available synObj's meet specific requirements (e.g. '*The lawyers faxed criminals indictments*').

AdjoinRight is another syntactic structure building operator, but it is more limited in scope. **AdjoinRight** only applies if the synObj being considered for attachment is a prepositional phrase or a complementizer phrase. It does not use the A/R set to determine if the attachment is allowed. This is because unlike other constructions (e.g. center-embedding), adjunction does not have a limit (e.g. '*That big ugly crazy blue dog*'). Currently only noun and verb shells or complementizer phrases can 'assign' an adjunction, while the synObj to be adjoined must be available as a complement in the receiver set. Furthermore, the **AdjoinRight** operator does not simply link existing structures. Instead, it must first sever an existing link (or destroy a node) and then insert itself, thereby creating two new links to do so. Additionally it updates the A/R set by removing the adjoined synObj from the receivers. The structures linked by the **AdjoinRight** operator are attached to the right. Figure 3.6 shows the sentence '*Hyenas burped quickly.*' first as two separate constituents, CP and AdvP, and then with the AdvP adjoined at the vP shell.

AdjoinLeft is very similar to **AdjoinRight**, but links structures to the left. Currently only

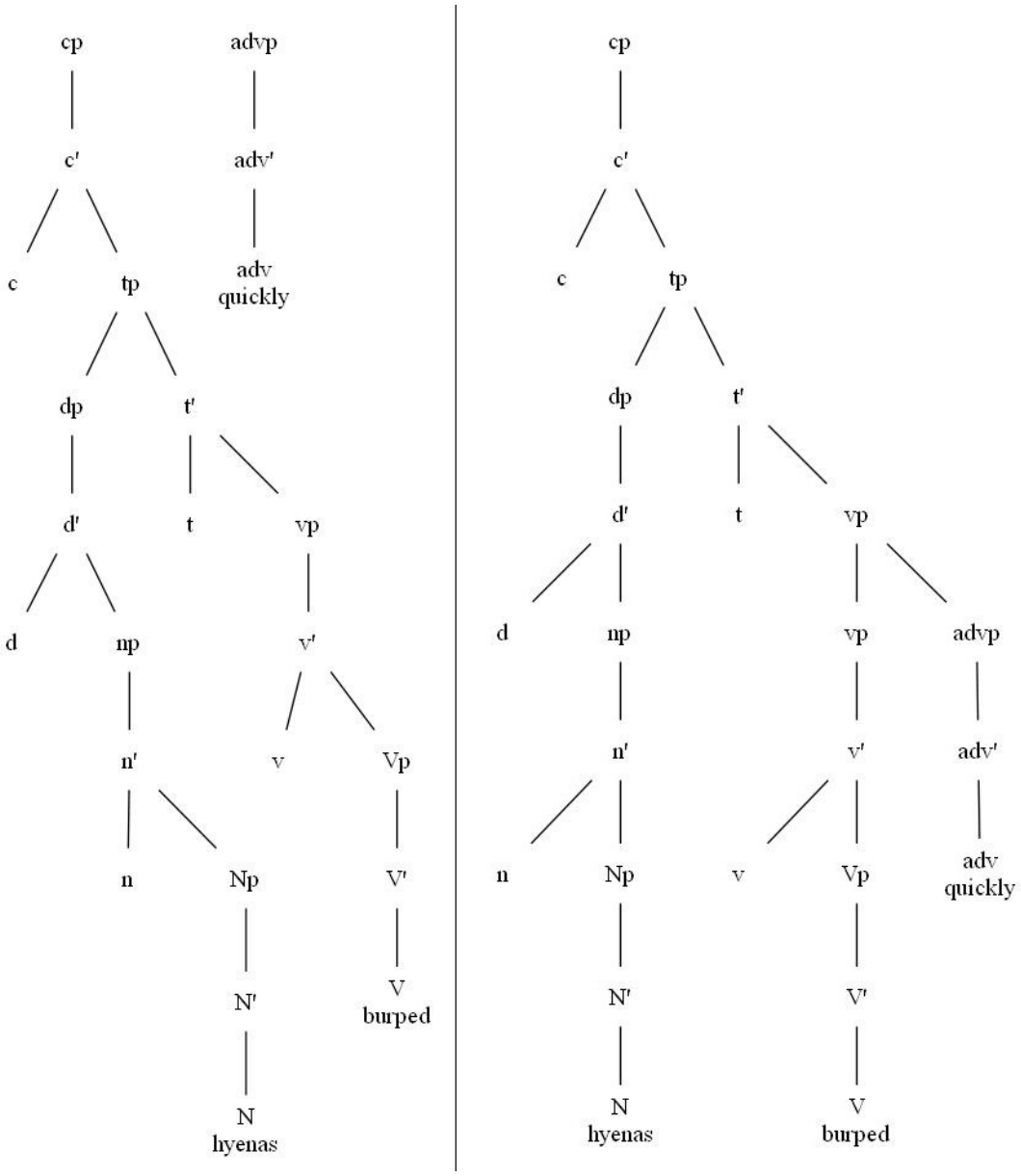

Figure 3.6 MP tree before and after adjoining a constituent.

adjective phrases and adverb phrases are available to be left-adjoined. Noun and verb shells as well as adjective and adverb phrases can 'assign' left adjunction (e.g. '*The big fat blue dog*').

Search control constrains the adjoin operators so that they are the last choice; all other structural options must be exhausted first. Finally, as was the case for all syntactic structure building, adjoin prefers closest attachment and the longest constituents. It should be noted that relative clauses are not part of the syntactic phenomena covered in this instantiation of XNL-Soar.

Fuse is the structure building operator for semantics. Semantic structure is generated when licensed by the semantic A/R set, i.e. when available semObj's include a matching assigner and receiver. The **fuse** operator actually subsumes 4 seperate operators, one each to create an external, internal, internal2, and property structure. Finally, recall that semantic structure building is dependent on the syntactic structure. For a **fuse** to be proposed a **compGraft**, **specGraft** or **adjoin** operator needs to have fired previously in the syntax.

3.2.4 Parse Finalization

Once all the words available on the input link have been accessed, the synObj's are checked for a tree that contains all those words. If no tree is found containing all the words then an **incompletion** operator fires; otherwise, a **completion** operator fires. Search control preferences ensure that these operators fire only after all other possible operators have been ruled out. This preference is important because once either of these operators fires, the agent terminates and no further processing occurs.

Instead of creating structure, the purpose of the snip operators is to undo structure by disconnecting existing links between nodes (Pritchett, 1992). XNL-Soar has a **synSnip** for syntax and **semSnip** for semantics. An important condition for **synSnip** to match is that an incomplete operator must already have fired. For syntax, the majority of the cases where **synSnip** matches involve ambiguity in the verbal complex. As a sentence is processed, it is possible that an ambiguous verb causes two (or more) structures to be created. As processing continues, these two structures both end up containing pieces of the correct parse. Once an **incomplete** is proposed (because all of

the words are not in a single tree), a snip can now occur, removing the modal as a main verb and linking the correct structures.

Figure 3.7 shows the application of the snip operator. The left side of 3.7 shows the vP and VP '*have*' verbal complex as a complement of the TP, meaning it is the main verb in the clause. After the sentence processing has received the verb '*expectorated*' and attached its vP and VP verbal complex as a main verb under the perfP projection of '*have*' (meaning it is an auxiliary or modal verb), the **semSnip** will fire to remove what it has now determined to be an erroneous interpretation of '*have*' as a main verb. After **semSnip** has fired, the main verb interpretation of '*have*' is marked with a "snipped" WME so that the graft operators will only attempt to attach the perfP '*have expectorated*' into the TP to complete the parse.

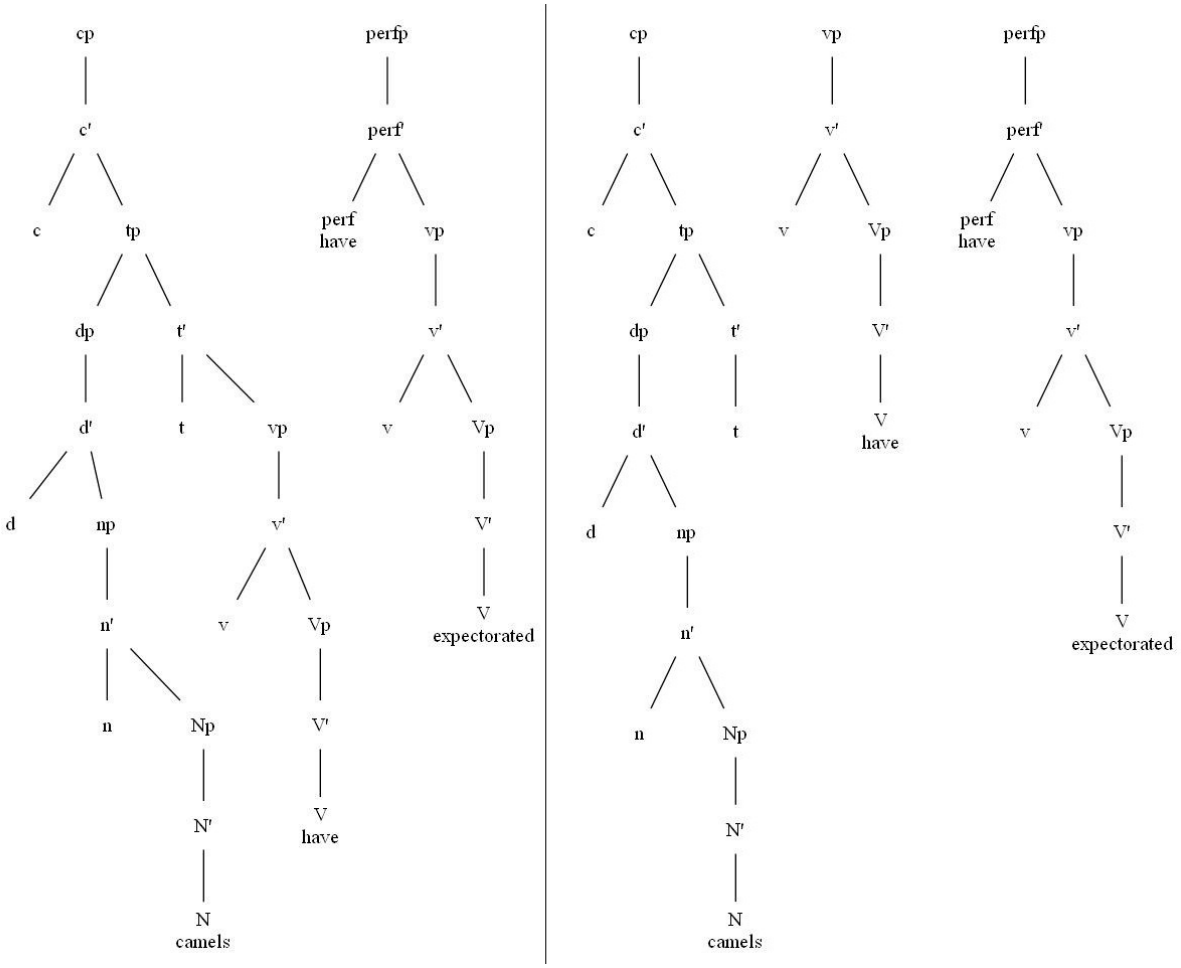

Figure 3.7 '*Camels have expectorated.*' shown before and after application of snip operator.

If the sentence has multiple modals (e.g '*The prices have been going up.*'), multiple snips are proposed and executed. Snips also update the A/R set and record which nodes have been snipped and are thus available again. This updating allows the same **graftComp** and **graftSpec** operators to match and correctly link the available structures, while the snipped label ensures that the system doesn't re-attach snipped synObj's to their former positions.

A precondition for **semSnip** is that a **synSnip** has occurred. Using the category and surface form of the snipped synObj, the system, (using synPro's and semPro's), selects the matching semObj. **SemSnip** then snips that semObj, again marking it as "snipped" and updating the semantic A/R set, allowing **fuse** to propose new semantic structure.

3.2.5 Bookkeeping

EdgeSpan and **semHead** are the last two operators to describe in the XNL-Soar system. Both of these operators are 'bookkeeping' mechanisms. Each synObj in the system originates as a lexical item having a WME attribute for the word's index in the sentence. This index is also recorded as a left and right edge on each synObj. As synObj's get built up and contain more than one word, the resulting synObj's edges are created by copying the leftmost and rightmost edges of the subsumed synObj's. **EdgeSpan** runs after every structure building operator, or snip, to ensure that edges are percolated up through every level of new structure. **SemHead** serves a similar function to percolate up an attribute containing the semantic head through levels of structure that have no semantics (e.g. determiners), and fires after every **fuse** or **semSnip** operator.

3.3 An Example Sentence

In this section, an example sentence (3.1) is presented with each operator's results represented graphically as MP parse trees or LCS's. Appendix A shows a hand-annotated trace of the operator firings.

(3.1) '*The lawyers pursue convictions.*'

Figure 3.8 shows the DP maximal projection created for '*the*' after access and the **project** operator (cycle 8 in appendix A). No semantics are represented for determiners, so XNL-Soar proceeds to the next word. Figure 3.9 shows the NP maximal projection created for '*lawyers*' again via the **project** operator (cycle 10 in appendix A).

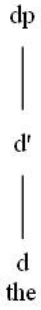

Figure 3.8 An Example Sentence: MP tree after first application of **getWord** and **project** (cycles 7 and 8 in appendix A).

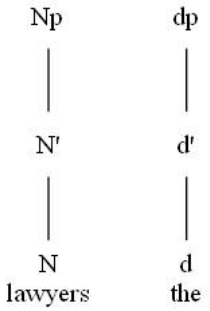

Figure 3.9 An Example Sentence: MP tree after second application of **project** (cycle 10 in appendix A).

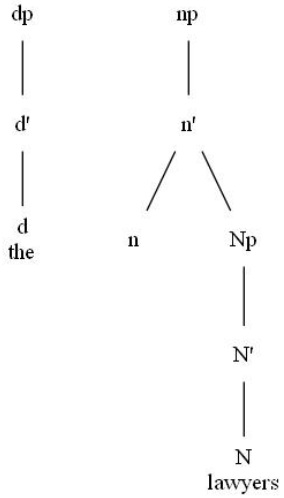

Figure 3.10 An Example Sentence: MP tree after application of **HOP** to lexical item '*lawyers*' (cycle 11 in appendix A).

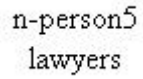

Figure 3.11 An Example Sentence: LCS after second application of **getWord** on lexical item '*lawyers*' (cycle 9 in appendix A).

The **HOP** operator now fires, as shown in Figure 3.10 (cycle 11 in appendix A), adding an nP shell to the NP maximal projection. Figure 3.11 shows the semObj created during access for lexical

item '*lawyers*' (cycle 9 in appendix A). Figure 3.12 shows the effect of **graftComp** attaching the nP '*lawyers*' to the DP '*the*' (cycle 12 in appendix A) as well as **getWord** and **project** on the verb '*pursue*' (cycles 12 and 13 in appendix A).

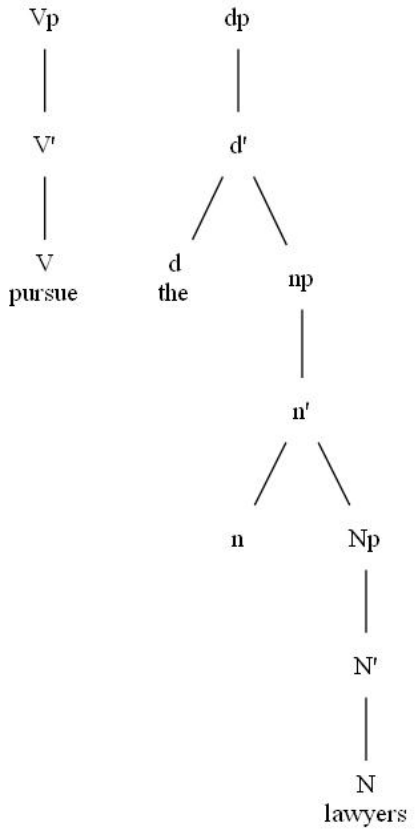

Figure 3.12 An Example Sentence: MP tree after application of **graftComp** to lexical items '*the*' and '*lawyers*' as well as the third application of **project** (cycles 12 and 14 in appendix A).

Figure 3.13 shows the utterance after applying the **HOP** operator to '*pursue*' adding a vP shell and TP maximal projection (cycle 15 and 16 in appendix A). The same figure shows the result of having **graftSpec** attach the DP '*the lawyers*' into the TP (cycle 17 in appendix A). Finally, figure 3.14 shows **HOP** having applied again adding a CP projection above the TP (cycle 18 in appendix A). Figure 3.15 shows the 3 new semObj's created by **getWord** for the verb '*pursue*': one for each of the senses identified in WordNet. Here XNL-Soar builds some semantic structure using **fuseExternal** to attach the semObj '*lawyers*' to the semObj '*pursue*' marked as the current

hypothesis, the result of which is shown in figure 3.16 (cycle 19 in appendix A).

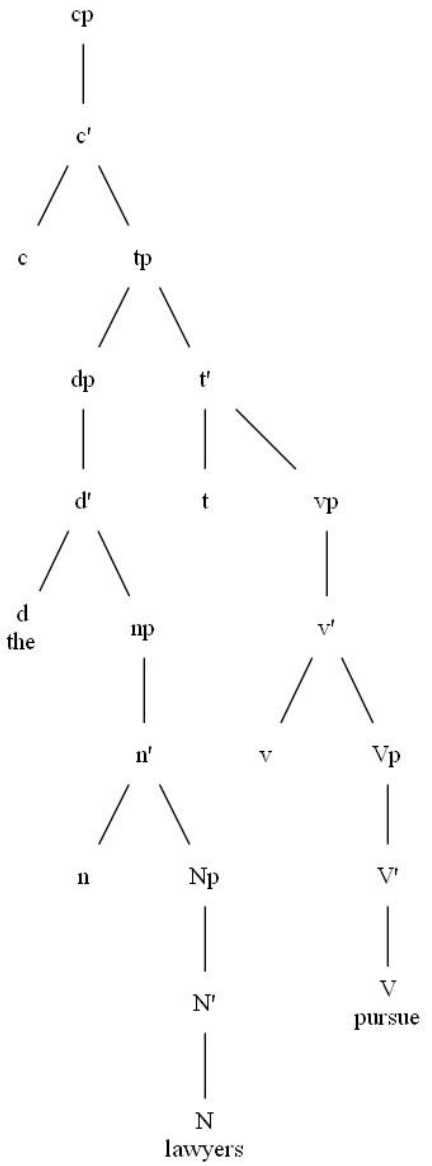

Figure 3.13 An Example Sentence: MP tree after application of **HOP** to lexical item '*pursue*' and **graftSpec** to DP maximal projection '*The lawyers*' and TP maximal projection '*pursue*' (cycles 15, 16 and 17 in appendix A).

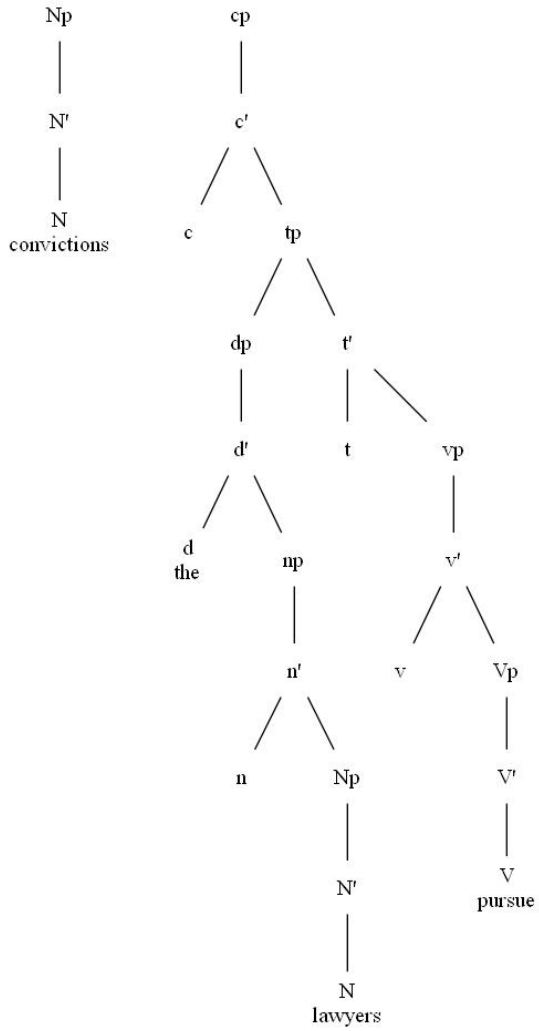

Figure 3.14 An Example Sentence: MP tree after fourth application of **project** on lexical item '*convictions*' (cycle 21 in appendix A).

Figure 3.14 shows the noun '*convictions*' after **getWord** and **project** have applied (cycles 20 and 21 in appendix A). Figure 3.17 shows the result of **HOP** on '*convictions*' adding a DP maxi-

mal projection (cycles 22 and 23 in appendix A). Figure 3.18 shows the result of the final syntactic operator in our example, **graftComp**,which attaches the DP '*convictions*' to the TP '*the laywers pursue*' (cycle 24 in appendix A). Figure 3.19 shows the final piece of semantic structure building using the **fuseInternal** operator to attach '*convictions*' to the verb '*pursue*' (cycle 25 in appendix A).

Figure 3.15 An Example Sentence: LCS after third application of **getWord** on lexical item '*pursue*' (cycle 13 in appendix A).

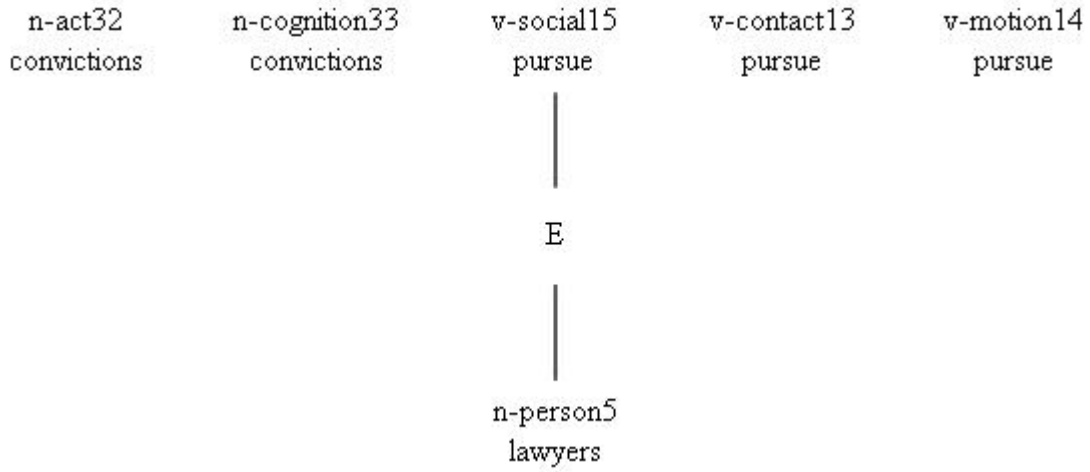

Figure 3.16 An Example Sentence: LCS after fourth application of **getWord** on lexical item '*convictions*' and first application of **fuse** (cycles 19 and 20 in appendix A).

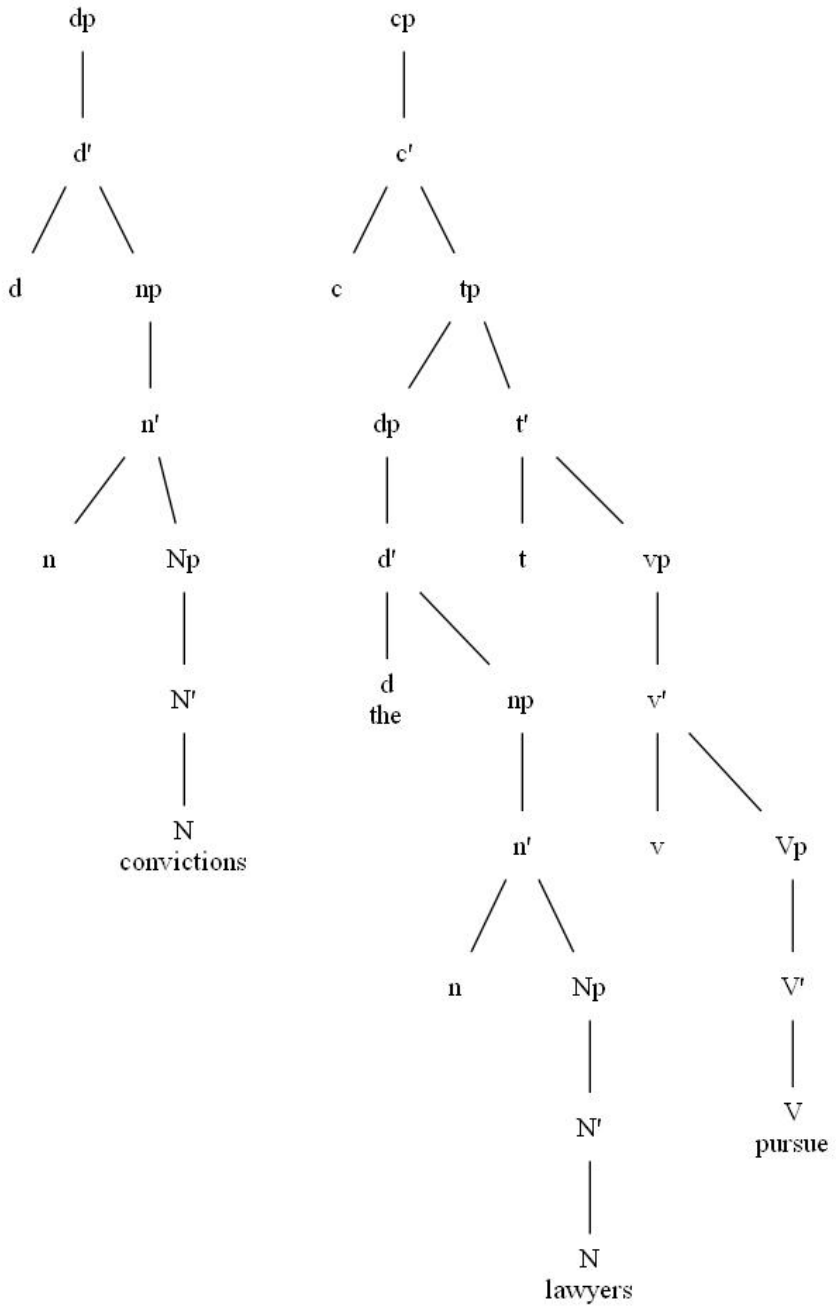

Figure 3.17 An Example Sentence: MP tree after two applications of **HOP** on lexical item '*convictions*' (cycles 22 and 23 in appendix A).

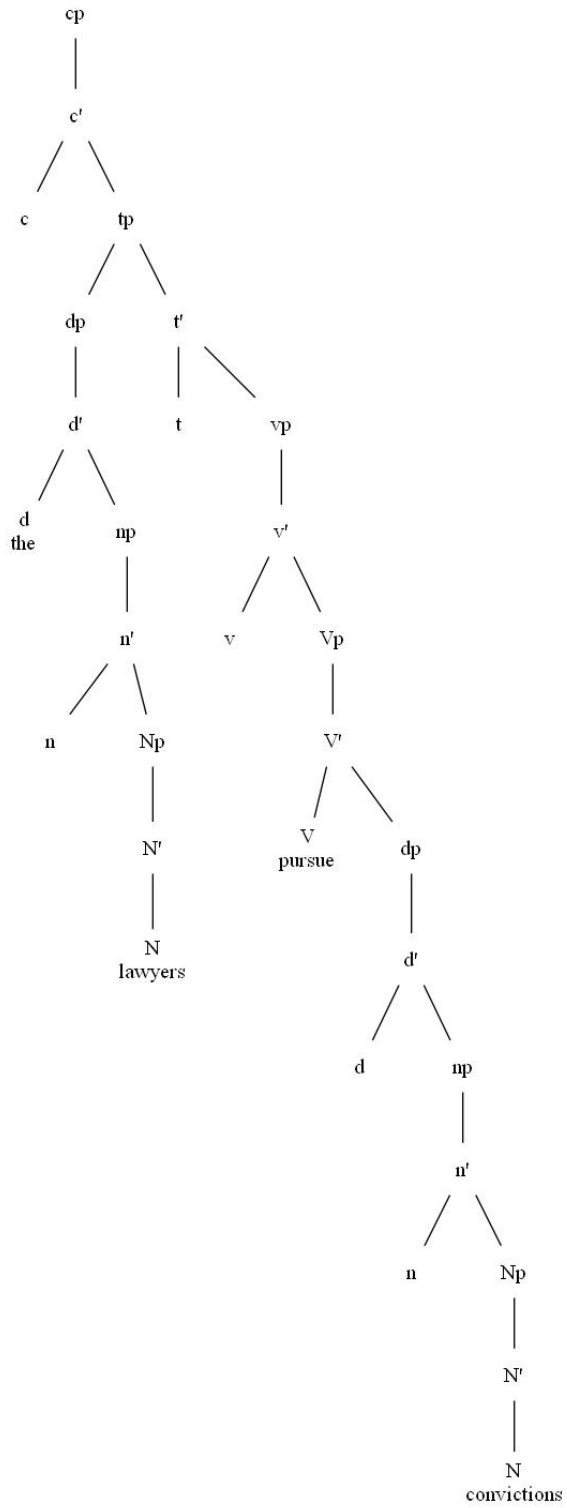

Figure 3.18 An Example Sentence: MP tree after second application of **graftComp** (cycle 24 in appendix A).

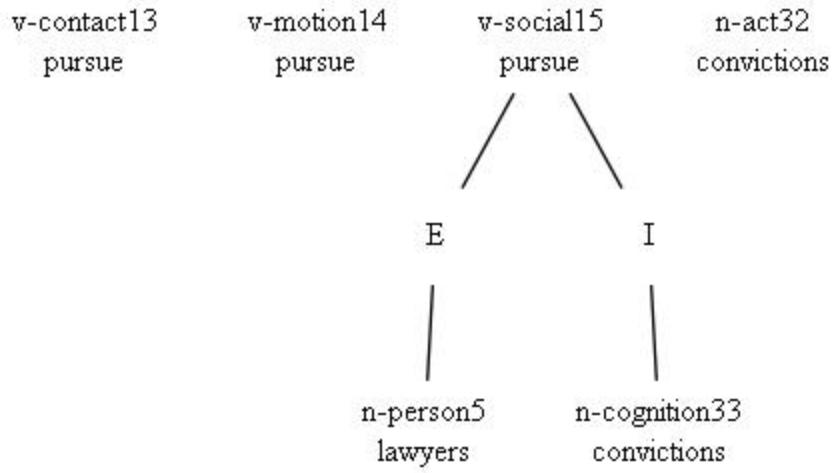

Figure 3.19 An Example Sentence: LCS after second application of **fuse** (cycle 25 in appendix A).

This is just one sentence to illustrate some of the operators and parsing capabilities of XNL-Soar. Several hundred sentences have been parsed successfully using this system. Because XNL-Soar has the ability to parse large numbers of sentences, we have carried out several experiments to measure the system's performance. Chapter 4 describes several of these experimental results.

Chapter 4: Empirical Validation

XNL-Soar differs in many important respects from other state-of-the-art natural language parsers (e.g. Charniak, 2000; Nivre and Hall, 2005; De Marneffe et al., 2006). The XNL-Soar system is not as efficient or robust as industrial-grade parsers and is not designed to parse by any order of magnitude faster than humans. On the contrary, it is meant to process syntax in a manner that closely mimics human performance. Currently, the system is not as versatile as humans in extracting meaning from ill-formed input. Work also remains to be done on canonical structures, including treatment of coordinated structures, compounds, relative clauses and certain types of adjunction. XNL-Soar uses no explicitly coded phrase-structure grammar (context-free or otherwise). Instead, permanent (long-term) memory contains rules that specify low-level constraints and principles that license or prohibit particular types of syntactic linkages. Additionally, XNL-Soar does not perform operations like lookahead; instead, processing at any given point is entirely predicated upon the current word and previously encountered ones. The current model has limitations: it is not capable, for example, of leveraging priming effects or collocational frequency statistics to perform next-word prediction. Finally, XNL-Soar does not employ data structures typical of many parsers. For example, it does not use a chart to exhaustively enumerate all possible partial parses; rather, the pursuit of constituency and combination is built into a singular ongoing hypothesis which is either pursued to completion or locally reanalyzed. Ambiguity packing is not performed; instead, complementary alternatives (in the limited instances where they are permitted) are separately represented in working memory, entailing some cognitive cost.

While standardized methods for parser evaluation exist, none seem wholly applicable to evaluating parsers claiming to be psychologically plausible. Traditional techniques offer little in the way of a gold standard for relating a given set of sentences with precise measurements quantifying human perception, attention, memory limits, processing time course, or other cognitive phenomena. Still, even if cross-system evaluations seem elusive at the current time, it is possible to quantify

performance of the XNL-Soar system in many different ways: in terms of syntactic and semantic constituency, lexical complexity, ambiguity resolution, word sense disambiguation, resource usage, versatility across various language processing modalities, and learning.

This chapter will focus on two measures to evaluate performance: resource usage and comparison to human/corpus data. Section 4.1.1 provides results quantifying the memory use of two distinct processing strategies. In sections 4.1.2 and 4.1.3 we give the details of two experiments using semantic memory to provide XNL-Soar with a knowledge source to simulate human linguistic decision making.

This thesis shows that XNL-Soar, as currently instantiated, resolves ambiguities common in language using strategies and resources including: reanalysis via snip operators, use of data-driven techniques with annotated corpora, and complex part-of-speech and word sense processing based on WordNet. This will be done by providing the results of the two experiments that quantify this hybrid system's improvement over baseline performance.

4.1 Experiments

4.1.1 Incremental Processing and Resource Usage

In developing XNL-Soar, it was necessary to consider how the syntactic mechanisms might support incremental processing, and the cost of doing so. Development focused on assessing how two different parsing control strategies supported parsing in as incremental a fashion as possible. The study of this question involved running sentences through the system and subsequently computing various statistical profiling processes to measure processing load.

The project/attach strategy

The first strategy retains some of the assumptions of the original GB-based theory (e.g. Lewis 1993). For example:

- Lexical categories are projected as completely as possible as soon as possible. Zero-level nodes are projected to XP nodes via one operator.

- No functional projections are built unless they are lexically licensed (except v).
- Structures are extended via the hierarchy of projections as soon as possible.
- Attaching complements and specifiers into pre-existing structure is performed as a last resort, and only when licensed.
- New words are not attended to until all possible structure has been built incrementally.

Thus, in processing a simple intransitive sentence, the agent posits structure as soon as possible, completing the subject's NP and then DP structure before the verb is encountered. Once the verb is attended to, it is projected up to a TP per information provided via operators that consult the clausal hierarchy of projections. The subject is then attached into the specifier position of the TP node.

Processing is similar for transitive verbs, with the additional step involving attaching the direct object into the complement position of the V-bar node. For ditransitive verbs, the first object is attached into the specifier of the VP node as soon as it is completed; the second object is attached into the complement of the V-bar node, similarly to how it is done for transitive verbs.

The bottom-up merge strategy

This strategy follows more closely the assumptions laid out in a recent text on Minimalist syntactic analysis (Adger, 2003). In particular:

- Only as much structure is projected as can be done at any given stage.
- No functional projections are built unless they are lexically licensed (except v).
- There are separate operators for projecting nodes at the intermediate (X-bar) and phrasal (XP) levels.
- There are separate operators for performing First Merge (incorporating complements) and Second Merge (incorporating specifiers).

- Projection to XP is only possible when licensed.
- Merge can only occur when licensed via features that need to be checked and deleted.
- New words are not attended to until all possible structure has been built incrementally.

In this case, the agent projects only as much structure as possible, one node at a time, as licensed. Intransitive verbs are constructed in a fashion similar to the previously discussed strategy. However with transitive verbs, the V-bar node is not built until the direct object's structure has been completed. Similarly, the TP node is not constructed until the subject can be combined into the specifier position of a T-bar root node. More interestingly, ditransitive instances require that no V-bar node can be constructed until the second object has been completed. Only then can it undergo First Merge to combine with the lexical verb. At this point, the first object combines with the V-bar node (i.e. in its specifier) via Second Merge to create a VP. Semantic information is also loaded for all three of the sentence types, forming the core for LCS construction.

Results

Figure 4.1 shows memory usage over time (measured in operator cycles) for the three canonical types of sentences (intransitive, transitive, and ditransitive) using the project/attach strategy. The intransitive sentence has the least amount of structure (both syntactically and semantically), and therefore the parse is completed in fewer decision cycles and using fewer WMEs than either the transitive or ditransitive sentences.

Though only representing a core set of possibilities for sentences, this experiment shows that the two strategies entail different amounts of resource usage, which can be quantified via statistical profiling in XNL-Soar. Figure 4.2 shows the results of statistical profiling for each strategy. Both graphs show the quantity of productions (rules) used by the strategies, the number of working memory changes, the maximum size of working memory used, as well as the mean size of working memory for both the project/merge (pm) and bottom-up (bu) strategies. The bottom-up strategy required substantially more resource usage, especially for ditransitive constructions. This is because verb phrasal structure must be held in abeyance until both internal arguments are completed.

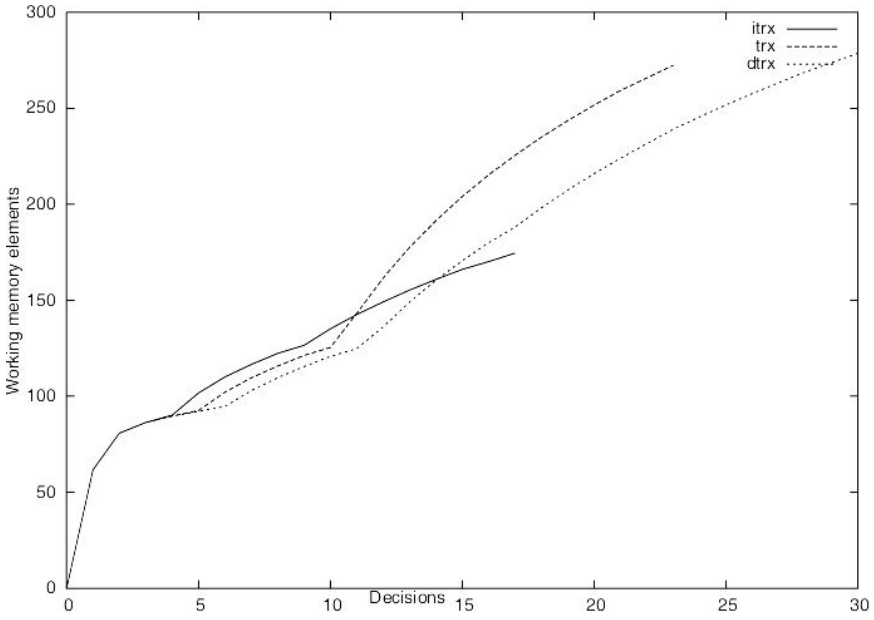

Figure 4.1 Working memory usage across operator cycles for intransitive, transitive, and ditransitive sentences.

A post-hoc analysis of the processing statistics showed that almost all of the changes in working memory are due to lexical access and the data retrieved at that time. Note that the project/merge parsing strategy that has been adopted as the default for the remaining experiments, a decision based in part on the results shown here.

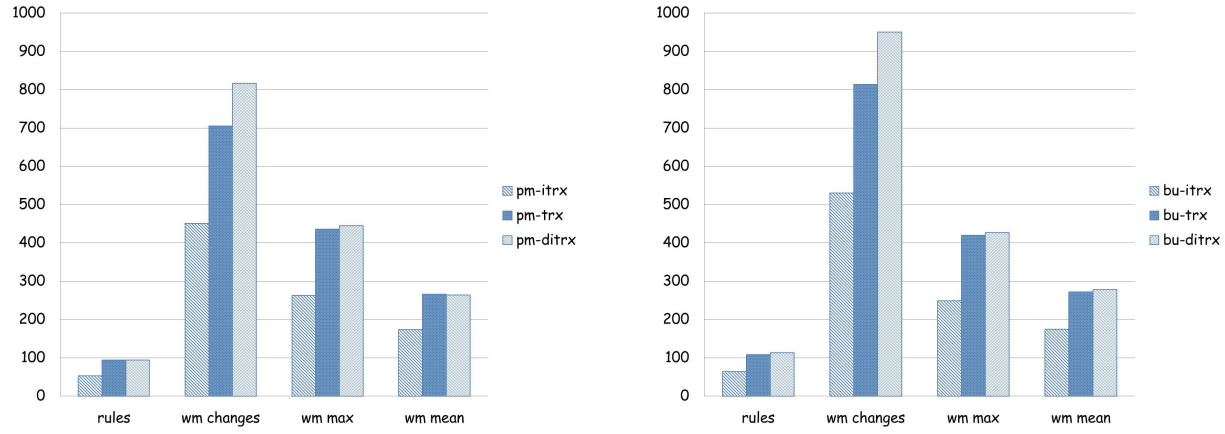

Figure 4.2 Statistical profiling for project/merge (pm) and bottom-up (bu) strategies.

4.1.2 Prepositional Phrase Attachment Resolution

The issue of prepositional phrase attachment (PP attachment) is an important and widely studied problem in natural language processing. Determining syntactic PP attachment is even problematic for humans, as different attachment sites lead to multiple semantic interpretations (e.g. '*I saw the man with the telescope.*'). Psycholinguistic research shows that human strategies for resolving PP-attachment ambiguities include the use of argument relations in the sentence (Schelstraete, 1996; Schuetze and Gibson, 1999), prosodic cues (Schafer, 1998; Straub, 1998), lexical constraints (Boland and Boehm-Jernigan, 1998), and context (Ferstl, 1994). Others have demonstrated that lexical bias and contextual information have a strong effect (Spivey-Knowlton and Sedivy, 1995). XNL-Soar, as a rule-based symbolic system, has traditionally inferred PP-attachments primarily using lexical subcategorization information (i.e. WordNet verb frames).

Though the system has been capable of handling relatively complex constructions, a large class of PP-attachment scenarios could not be processed by early versions of the system. In particular, problems arose when the attachment decision was determined, not by subcategorization information on the matrix verb, but rather by lexical semantic information contained in the oblique object of the preposition. This leads to familiar structural ambiguities, which are sometimes ambiguous even for humans:

(4.1) '*I saw the man with a beard/telescope.*'

Here attachment is determined by the PP object ('*beard*' or '*telescope*'), not by the subcategorization of '*saw*'. Hence, subcategorization information alone is insufficient to make PP-attachment decisions in many contexts; another approach was needed by NL-Soar to deal with ambiguities of this type.

Past approaches to PP-attachment disambiguation have focused on either statistical methods (Hindle and Rooth, 1993; Ratnaparkhi et al., 1994; Collins and Brooks, 1995) or rule-based methods (Brill and Resnik, 1994). The statistical approaches generally involve mining large annotated corpora and treebanks for determining the probability of an unknown attachment, usually based on

the environment's lexical content. For '*I saw the man with the telescope.*', a stochastic parser might assert an 84% probability for attachment to the verb '*saw*' and a 16% probability for attachment to the noun '*man*'. Since, for both of these tasks, exemplar-based methods seem more appropriate, this thesis describes their integration within the framework of XNL-Soar.

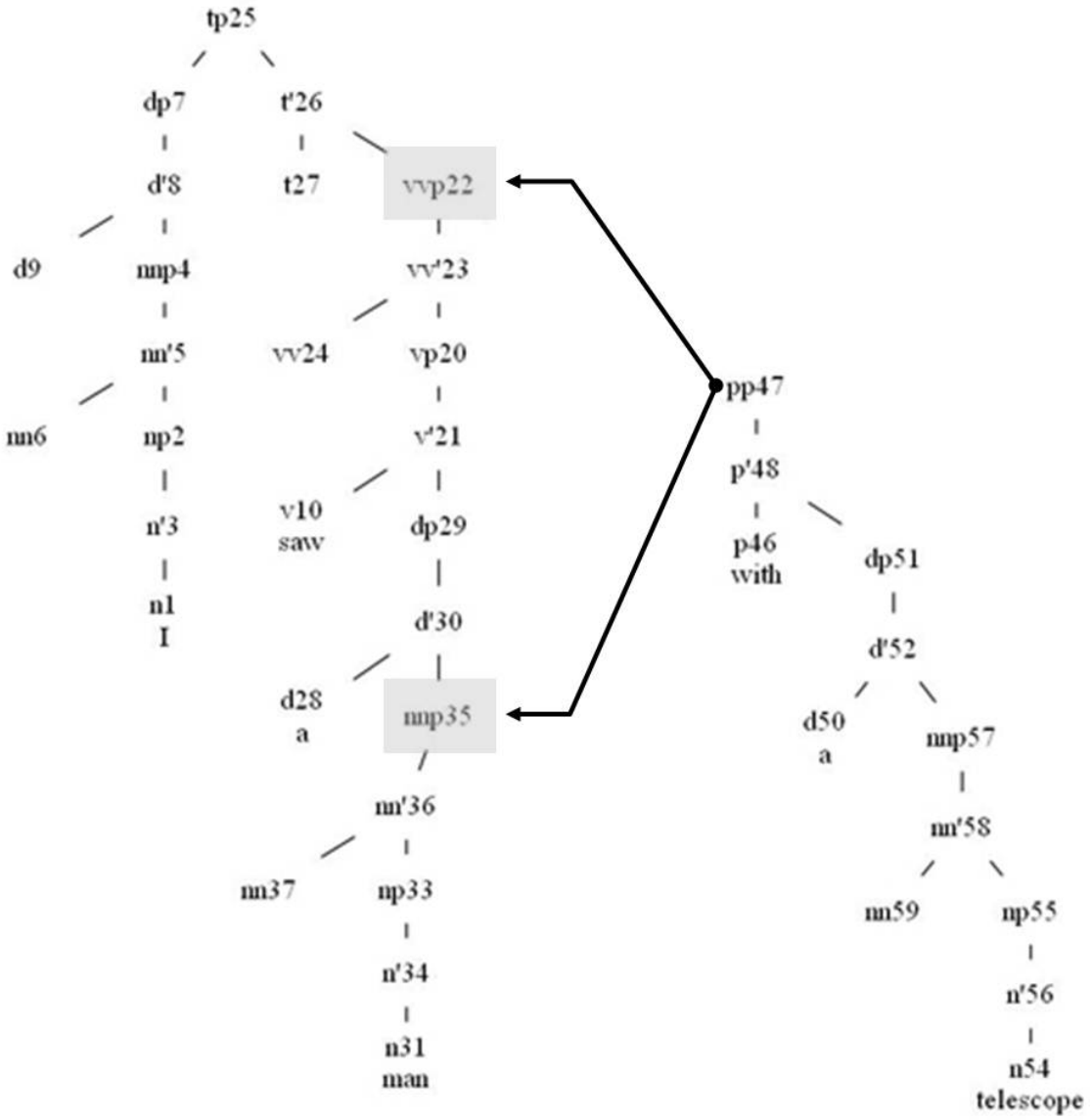

Figure 4.3 Possible attachment sites for prepositional phrase from example 4.1.

Addressing the PP attachement issue has involved infrastructural adaptations to the core Soar architecture. One of these adaptations was integrating semantic memory, a new module that stores

long-term learned (or at least declarative) knowledge and assures retrieval via queries from Soar productions based on exact or partial matches. This involved populating the semantic memory module with prior knowledge about appropriate PP attachments where ambiguity is possible. Similar to Lonsdale and Manookin (2004), we mined PP attachment decisions from the Brill/Resnik PP-attachment corpus (12,266 examples) derived from the Wall Street Journal. Sample instances are:

```
N, improve protection of
V, hurting themselves by
```

meaning (respectively) that:

- "of" should attach to the noun (nnP) "protection" when the latter is the object of the verb "improve"
- "by" should attach to the verb (vvP) "hurting" when the latter's object is "themselves"

With this instance base installed in semantic memory, the system can query it either directly or at various levels of abstraction for guidance in PP attachment decisions. Consider the sentence: '*The prosecutor faxed the indictment to the judge.*' XNL-Soar incrementally parses the sentence until the preposition '*to*' is encountered. At this point there is no certain way for the system to decide whether to attach the preposition '*to*' to the noun (nnP) or to the verb (vvP): possible strategies include random choice, minimal attachment, late closure, right association, analogy, probabilistic reasoning, etc. Table 4.1 explains the assumptions of each of the strategies we used in this experiment at precisely this decision point in processing.

Results

Figure 4.4 shows how the six different strategies described in table 4.1 performed in XNL-Soar and compares performance using the Brill/Resnik corpus of sentences that exhibit this ambiguity. The right association strategy performs best for PP's that should attach to the noun phrase (nnP) because

PP attachment strategies	
Right Association	Always attach the PP to the rightmost available position in the sentence Guaranteed 100% completion Will always adjoin to the noun (nnP) in our sentences
Random Attachment	Select PP attachment site randomly from those available No principled basis for site selection Often results in incompletion because of untenable hypotheses point of departure, pseudo-baseline
Semantic Memory	Load exemplar base into SMem At moment of PP attachment indecision, query SMem using the noun, verb, and preposition Query will return closest match with exemplar base SMem has three possible responses; N, V, or failure If successful, use SMem answer If failure (i.e. no matching exemplar), default to random attachment
3 sub-strategies: • Semantic Memory+Verb • Semantic Memory+Noun • Semantic Memory+	Augment SMem exemplar set: • Use V hypernyms • Use N hypernyms • Use both V and N hypernyms The addition of hypernyms generalizes exemplar base

Table 4.1 Prepositional phrase attachment strategies.

the are always (in our exemplar set), the rightmost attachment site available. This artificially inflates the results for the right association strategy and noun phrases (nnP), but it serves as a useful point of departure in our analysis. The random attachment performs very poorly for both noun (nnP) and verb (vvP) attachment sites. The semantic memory strategy improves performance substantially for noun (nnP) attachment; however the semantic memory strategy was not nearly as successful for verb (vvP) attachment. A possible explanation for this is the paucity of data available to use in our exemplar set. Adding the hypernyms of verbs and nouns in the semantic memory plus strategies did improve performance, albeit only slightly.

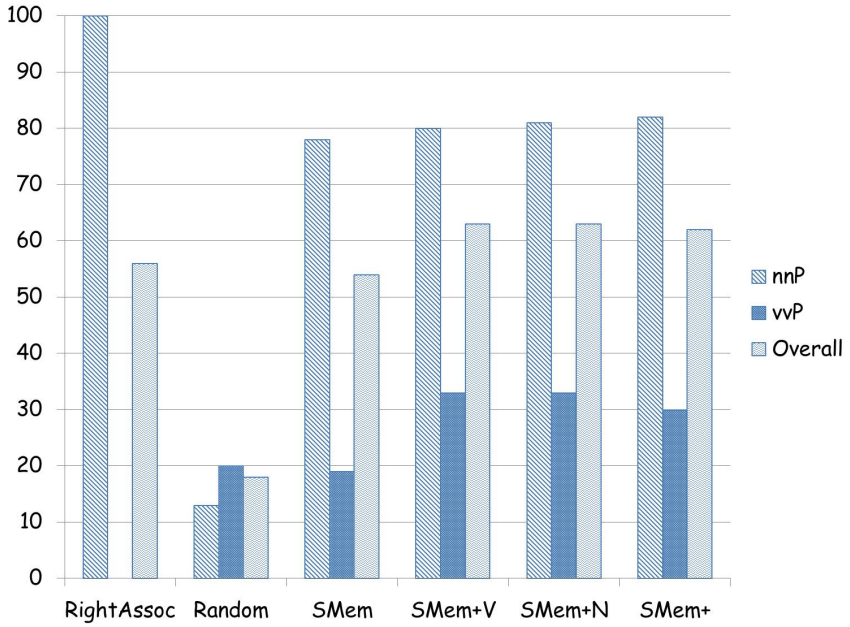

Figure 4.4 PP attachment results by strategy (as percentage).

This experiment shows that by accessing semantic memory at a crucial stage in the parsing process, we are able to use prior knowledge to guide attachment decisions in a way that supports XNL-Soar's incremental parsing. It is likely that the corpus-based approach we use here in connection with semantic memory to resolve ambiguity could also be extended to other types of syntactic constructions. Our third experiment, involving progressive vs copula+gerund disambiguation also uses semantic memory to carry out structural disambiguation.

4.1.3 Progressive vs. Copula+Gerund Disambiguation

Another syntactic ambiguity involves where a phrase or other syntactic structure's interaction with other structures in the sentence may lead to meaning differences. Sentences like those shown in examples 4.2 and 4.3 illustrate the potential structural ambiguity between a progressive and a copula+gerund. In the first sentence, the word '*was*' serves as an auxiliary verb, signaling that the main verb '*singing*' has progressive aspect. In the second sentence, '*was*' serves as a form of the verb "to be", which is often called a copula or linking verb. In these cases there is little trouble for humans in distinguishing between the function of these verbs, but more difficult cases exist. On the other hand, the distinction is difficult for computers to disambiguate correctly.

(4.2) '*My favorite cousin was singing.*'

(4.3) '*My favorite activity was singing.*'

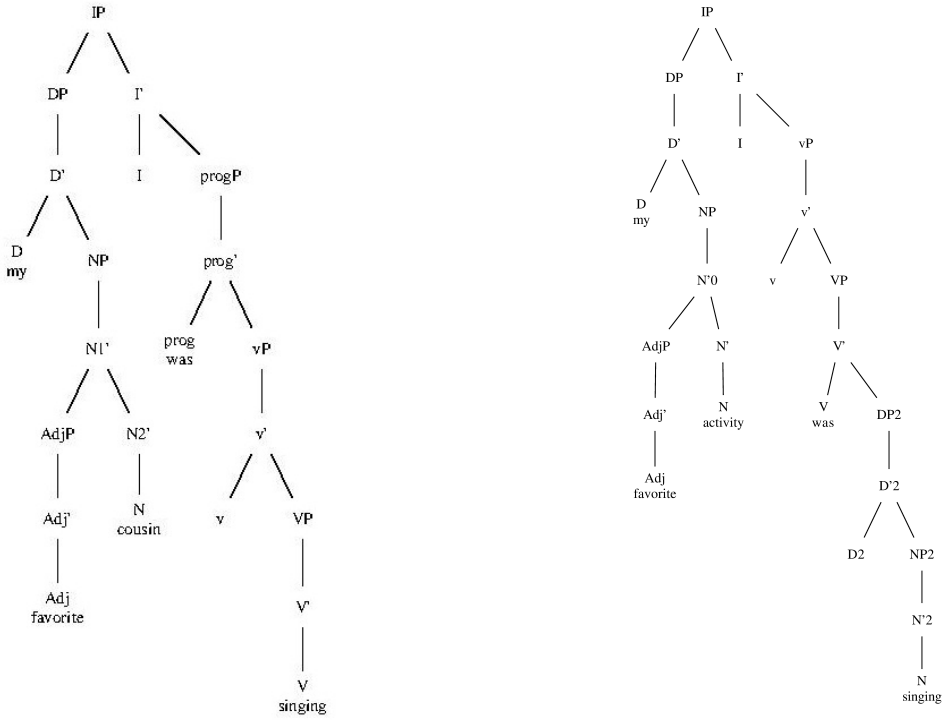

Figure 4.5 Parse trees for examples 4.2 and 4.3.

The use of XNL-Soar's new syntactic model has implications for the progressive versus copula+gerund ambiguity discussed previously. We assume an explicit articulation of the various auxiliary verbs for English in the form of separate X-bar projections for progressive, perfective, and passive items (Adger, 2003; Lonsdale, 2006). Figure 4.5 shows the respective parses for the progressive construction (which uses the progressive projection) and the gerund construction (that uses a main verb's projection). Note that in either case the verb in the Prog node or the V node is some form the the verb "be" (i.e. "is," "were," "are," "be," etc.). For this analysis, the potential ambiguity with passive constructions, which also use this auxiliary but whose main verb is encoded as a past participle, was ignored.

The corpus

In order to ground this work in actual language use, data was collected from available corpora and treebank resources. We resolved to first consult treebanks, because of their extensive syntactic annotation. We then used the tgrep2 tool to retrieve relevant sentences from the Penn Treebank (Marcus et al., 1994). It was readily apparent that examples of the progressive construction were numerous (over 4500 sentences); see Figure 4.6 for the relevant tgrep2 queries.

```
S<(VP</^VB/<(VP<VBG))
S<(VP<(/^VB/ . (NP-PRD<NP<<,VBG)))
```

Figure 4.6 Queries to retrieve the two sentence types: progressive (top) and copula+gerund (bottom).

```
One reason is mounting competition from new Japanese car plants .
The main reason for the production decline is shrinking output of light crude .
Mr.  Leinberger is managing partner of a real-estate advisory firm .
Conference committees are breeding grounds for mischief .
```

Figure 4.7 Sample non-progressive sentences collected from the Penn Treebank.

On the other hand, the amount of copula+gerund constructions harvested from PTB was surprisingly meager, resulting in only a handful of instances, such as those in Figure 4.7. In fact, each

of these is arguably not even of the copula+gerund type since the "-ing" word is in fact a present participle filling the role of an attributive adjective to another noun.

Since copula+gerund examples were in such short supply in PTB, three other corpora were consulted to find and extract sentences with this type of construction:

- The BYU interface to the British National Corpus (Davies, 2004), originally created in the late 1980's by Oxford University Press, containing 100 million words of English.
- The Corpus of Contemporary American English (COCA) (Davies, 2009) is the largest freely-available corpus of English, and the only large and balanced corpus of American English. The corpus contains more than 410 million words of text and is equally divided among spoken language, fiction, popular magazines, newspapers, and academic texts, and includes 20 million words each year from 1990-2010.
- The Corpus of Historical American English (COHA) (Davies, 2010) is the largest structured corpus of historical English consisting of 400 million words from texts of 1810-2009.

Though these corpora do not have full treebank-style syntactic parses, their words are tagged for part of speech (POS) and have interfaces that support querying for limited sequences of tags, words, and other similar information. Finding sentences with progressive tenses was straightforward; there were tens of thousands of such instances. Again, the copula+gerund examples were not nearly as prevalent.

Part of this is due to POS tagging errors which, for progressives and gerunds, are particularly common. For example, in the COCA, a sequence of noun plus copula+gerund is coded as:

```
[n*] [vb*] *ing.[n*]
```

but it erroneously returns sequences such as "heart is pounding," "storm is brewing," "people are spending," and so forth due to POS tagging errors. This demonstrates the difficulty of processing this type of construction and the limited state of the art in addressing this syntactic ambiguity type.

We also excluded sentences where other factors complicated the syntactic analysis and disambiguation strategy; in particular, those listed below were left for future investigation:

- potential ambiguity exists between contracted auxiliaries and the possessive:
  The court's reasoning that the two jurisdictions are sovereign...
- multi-word expressions and fixed phrases mask syntactic structure:
  Chances are homebuilding will require...
- the post-copula gerund forms the initial part of a nominal compound:
  On the floor were sleeping bags, knapsacks, ...
- the subject is a pronoun and hence determining reference is difficult:
  ...some were counseling caution...
  ...others are shuffling...
- the mood is interrogative:
  In which country was fencing originated?

Figure 4.8 shows some copula+gerund sentences which have been matched and extracted from the corpora discussed above. In total, about 80 copula+gerund sentences were gathered and, naturally, thousands of progressives.

In most cases, resolution of the ambiguity is fairly straightforward for humans: most are clearly progressive or clearly copula+gerund. However, some sentences illustrate that indeterminism does exist, especially at the sentence level. An example like 4.4 could be construed as involving either construction, depending on wider context.

(4.4) '*His business is advertising.*'

Executing disambiguation decisions

As mentioned above, the corpus collection process revealed a large discrepancy in the ratio of progressive sentences to copula+gerund ones. Hence, from a global perspective, a rational choice

```
Rosemary's hobbies are gardening, walking, ...
The charges are kidnapping, torture, ...
Examples of such activities are mountaineering, rock climbing, ...
In tundra and taiga the natural resources are fishing and hunting.
Included in our amusements are gardening, riding, shooting, ...
Common treatments have been counseling and psychotherapy.
Part of the issue is timing.
Bowling is bowling, and I know what it takes.
The central fact is that cloning is cloning is cloning.
His passion is fishing.
The bottleneck is training and finance.
Betting is gambling.
My talent is gambling.
Our theme was gardening ...
My passion was teaching.
His one good distraction was gardening.
His main business was ranching.
His favorite recreations were gardening and fishing.
```

Figure 4.8 Example sentences of the copula+gerund type mined from several corpora and annotated.

and most productive baseline would be to assume that all such sentences would be of the progressive type. However, since the parser is incremental, when processing either example 4.2 or 4.3 and receiving the word '*was*', the system doesn't know which construction will emerge as subsequent words are encountered. Short of doing lookahead the default processing step is to assume that '*was*' is a main verb since, generally in English, copula sentences outnumber sentences with progressive forms.

In some circumstances, the next word, the -ing word, is unambiguous. If it's an unambiguous noun, the system will link in the noun as the object of the main verb (including copula) and the sentence terminates with a successful parse. This case is uninteresting for the work reported here and will not be discussed further. Similarly, if the incoming -ing word is unambiguously a verb, the system recognizes the inconsistency with the auxiliary's temporary assumption, and "snips" the main verb node from the tree, thus repairing the construction by replacing it with the progressive

auxiliary Prog node. At that point, the -ing form can be incorporated into the tree as its complement, and the sentence terminates successfully with a progressive reading. Again, since there is no ambiguity in the -ing word, the case is not directly interesting to us for this paper.

Crucially, it is our source for lexical knowledge, WordNet, that determines whether a given -ing form is (un)ambiguous, having a noun reading, a verb reading, or both. While an excellent resource, WordNet is not entirely consistent in this determination: '*engineering*', for example, is only listed as a noun, though the conversion-derived verb is common, as in '*They are engineering a solution to the problem.*' On the other hand, it does list both verbal and nominal lemmas for '*kidnapping*'.

Note that WordNet appropriately lists '*golfing*' and '*fishing*' as both verbal and nominal in nature. When our system parses sentences like '*His hobby was fishing.*', it will parse correctly using the default strategy, given the appropriate assumption that '*was*' is a main verb. However, the system would parse a sentence like '*The president was golfing.*' in exactly the same way: the result would be an incorrect copula construction, instead of the correct progressive construction. What was needed was a way for the agent to encapsulate and access information that would help identify, when confronted with an incoming ambiguous -ing word, when to proceed with one parse versus the other.

In order to process copula+gerund constructions, though, we need to rely on further information. We tested several different solutions to this problem, all involving knowledge sources that the XNL-Soar agent could consult at the time of indeterminism.

Evaluation and Results

In this section, we sketch each of the five disambiguation approaches and discuss results from implementing them in the XNL-Soar system.

First, since progressive sentences out-number copula+gerund ones (both in random English running text and in our training corpus), a reasonable baseline disambiguation approach would be to simply always assume a progressive. This strategy, called *forceverb*, obviously ignores and cannot process any ambiguous copula+gerund constructions. This constitutes the default strategy

for our agent, as outlined above.

Second, if we want to be sure to process all copula+gerund constructions correctly, we could reverse the processing mentioned in the previous point and ignore all possible progressive constructions. We call this *forcenoun*. Given the relative paucity of this construction, this would be worse than the baseline, but is included for the purpose of comparison.

Third, we could have the agent enact a random strategy for choosing between the two alternatives. The result would be that many instances of each category would be missed. Here, we simply use a 50/50 random choice since we don't know *a priori* what the constitution of the future unseen test corpus would be.

Fourth, we converted all of the corpus examples into data that could be loaded into semantic memory (Smem). This allows a resource for the agent to query when deciding which alternative to pursue. In particular, given $word_1$ (the subject) and $word_2$ (the -ing word), if a direct match is found, the copula+gerund construal is preferred over the progressive one. If no match is found, the agent tries for a partial match by abstracting away from the two lexical items themselves and querying instead based on their WordNet hierarchical-semantic categories. Thus, if semantic memory contains a match for '*activity*' and '*reading*' (such as from the sentence '*My favorite activity was reading.*'), but the agent is processing a sentence like '*My favorite pastime was reading.*', a partial match will result since '*activity*' and '*pastime*' are hierarchically close.

Fifth, the agent can enact a semantic-based resolution strategy. A similarity metric is generated for the subject/-ing word pair using the Java WordNet::Similarity implementation.[1] The program provides similarity measurements given by several algorithms, so based on our corpus collection, we determined the best metric for our disambiguation procedure. The Resnik measurement (Resnik, 1995) correlated most highly with the training data, namely with the function:

1.81 + 3.93*Math.cos(13.2*resnik)/resnik

where a result ≥ 1.73 indicates a copula+gerund, and otherwise a progressive. The mean absolute error on a validation set we evaluated was .093. This function was then encoded in the agent as a

[1] http://www.sussex.ac.uk/Users/drh21/

means of identifying copula+gerund instances not yet present in semantic memory.

For the test set we had annotators collect all of the relevant sentences of both types from an independent resource: the WordNet example glosses. They also removed adjuncts and other material not crucial to the determination of the attachment site in incremental parsing for the ambiguity we address in this paper. This resulted in a set of 92 sentences, 54 of which involved progressives and 38 of which involved copula+gerund constructions.

Using the XNL-Soar system on this corpus of test sentences, we employed each of the five strategies outlined above. The XNL-Soar system experienced four parsing failures due to words missing from WordNet, and hence not available to the system for subsequent processing: '*telecommuting*' and '*face-painting*' as verbs, for example.

Strategy	Correct	Incorrect
forceverb	52	36
forcenoun	36	52
random(avg.)	44.0	43.7
Smem	46	42
Resnik	68	20

Table 4.2 XNL-Soar performance on test set of sentences (sample size=92, failed=4).

Table 4.2 provides the results for each of the five strategies. For example, the *forceverb*, or baseline strategy, correctly predicted 52 of 88 attachments (the number of verbs in our exemplars). The *forcenoun* results correctly predicted only 36 of the 88 attachments (the number of nouns in our exemplars). The *random* strategy performed as expected, predicting attachments correctly about half the time. The Smem strategy correctly predicted 46 of of 88 attachment sites. Finally, the Resnik metric correctly predicted 68 of 88 attachments, well above the baseline.

Conclusion

The *forceverb* and *forcenoun* results are not surprising, given the discussion above, nor is the *random* result (averaged over 5 runs).

We expected better performance by Smem, but as it turned out, there was no lexical and very

little semantic overlap between the training and test corpora. In retrospect, this is not too surprising given the data sparseness issue and modest size of the corpora. With more extensive training data, we expect this result to improve.

The Resnik metric handily outperformed all other approaches. For example, the test data contained a few tautological sentences, like '*Bowling is bowling.*' and '*Beekeeping is beekeeping.*' where the measure detected best-case semantic similarity (namely identity), correctly licensing the copula.

We suspected, and our experiments have illustrated, that the progressive versus copula+gerund ambiguity poses a challenge to an agent encountering either structure during incremental parsing. Various strategies are possible and even fairly successful, though they are dependent on corpus-based knowledge sources which, as we have seen, are scanty, error-prone, and (like all hand-annotated data) effort-intensive to develop. This work investigated several strategies in isolation, but it should be possible to integrate all of them into XNL-Soar. Some meta-strategy would be necessary to mediate the conflicting proposals that would arise.

Though we have concentrated only on the progressive versus copula+gerund ambiguity, related difficulties abound, especially when semantics is also implicated, as it is in our agent. We have mentioned earlier several phenomena remaining which are beyond the scope of this paper, but nevertheless must be addressed to assure robust treatment.

There are still other closely related constructions. Consider, for example, the following sentence:

'*Twenty minutes after they've arrived, chicken is frying, lamb ragout is simmering,...*'

In this sentence, '*chicken*' is syntactically the subject of a progressive construction of the type we have been discussing. However at the same time, it is the semantic theme (or patient) of the verb. This syntax/semantics mismatch, often called an unaccusative or middle verb construction, is also implicated in our discussion here: constructions like these must be detected if the sentence is to be understood correctly. In the future we intend to use the techniques sketched in this thesis,

namely semantic memory and semantic distance, to tease apart these types of unaccusatives from active (i.e. unergative) predicates involving ambiguous -ing words.

4.2 Results

Experiment 1 showed that as sentence complexity increased, the amount of memory and rules necessary to parse it also increased. Further, it showed that different parsing strategies also influence the quantity of resources necessary to process a sentence. Experiment 2 showed that semantic memory provides the system with a knowledge source that, when accessed at a crucial decision point, resolves the structural ambiguity involved in prepositional phrase attachment. Experiment 3 used semantic memory to provide a knowledge source to disambiguate progressive versus copula+gerund constructions. It also showed that encoding semantic proximity (Resnik, 1995) into the XNL-Soar system provides usable metrics that successfully disambiguate progressive versus copula+gerund constructions well above baseline performance.

Chapter 5: Conclusion and Future Work

This thesis has outlined the development of the XNL-Soar system and its natural language processing capabilities. The current instantiation generates syntactic (MP) and semantic (LCS) representations. Additionally, XNL-Soar takes advantage of the newly available semantic memory to utilize other knowledge sources. My contributions to the system focused on the semantic processing and implementing the semantic memory module. I also participated in the efforts to extend the syntactic coverage of the system.

This thesis has shown that XNL-Soar processes lexical complexity and handles structural disambiguation in a state-of-the-art and principled manner. Empirical results provided statistical evidence that the system outperforms the baseline (random in section 4.1.2 and forceverb in section 4.1.3) with its hybrid cognitive architecture. It also demonstrated that resource usage increased proportionally to sentence complexity.

5.1 Future Work

Beyond these initial successes, much remains to be done to increase XNL-Soar's capabilities. Principal among these is the need to devise a strategy to test the semantic processing capabilities. Just as experiments 2 and 3 illustrated they system's syntactic capabilities, experiments to quantify semantic coverage should be carried out. Comparisons to other systems would be ideal, but similarly to the difficulty in finding systems with which to compare the syntactic coverage, there is even less work being done in semantic processing.

Another important area where this work should be extended is making better comparisons to human data. Lewis (1993) carried out work on garden paths, center embeddings, unproblematic ambiguities, and parser breakdowns. A close replication of his work would represent an important step in increasing syntactic coverage and comparison to available human observations.

Further investigations using semantic memory and other corpora constitutes another area for

further research. Encoding resources like the Brill/Resnik corpus and COCA from experiments 2 and 3 show that this approach can be successful.They further illustrate the need to identify other linguistic knowledge sources and how useful they can be in improving natural language processing. One of the most exciting of these opportunities is the availability of the newly released Annotated English Gigaword corpus (Napoles et al., 2012), containing 183 million English sentences richly annotated.

XNL-Soar's coverage of syntactic phenomena is one limitation of this work. Coverage of clausal constructions, distinguishing unergative from unaccusative, and more phenomena still needs production rules, and perhaps even new operators written for correct processing. More complete coverage of syntactic phenomena would allow us to experiment with more NLP problems. Enriching our implementation of the Minimalist Program, by adding feature checking and movement would also allow us to align our experimentation with active linguistic research.

Word sense disambiguation is finding the most appropriate meaning for a word in context. Word sense disambiguation and related tasks like named entity recognition, anaphora resolution and more constitute one of the most active areas of NLP research. The use of semantic memory to encode resources like SEMCOR (Miller et al., 1993) and others provides XNL-Soar with the opportunity to make a significant contribution to this research.

A larger undertaking is to more fully parameterize the syntactic and semantic processing as independent modules. Currently semantics is informed by, and always follows, syntax. Experimentation with semantics and syntax as independent systems, or of their effectiveness by interacting with each other would be an endeavor of great interest to the psycholinguistic community.

Finally, I echo Rytting (2000) and his sentiment that his thesis (and mine), "...does not, and cannot, describe a finished product - modeling [incremental sentence processing] is too large a task, even when parceled into smaller tasks, and Soar is a paradigm designed to integrate theories, not to further subdivide them. I do hope, however, that this thesis does suggest the possibilities of language modeling within the NL-Soar system, and provide a guide to further research."

Appendix A: Example Sentence

Operator Trace from XNL-Soar

The annotated trace below shows each operator cycle for the example sentence '*The lawyers pursue convictions.*' from section 3.3.

initialize agent
source (XNL-Soar_operators.soar)
Total: 315 productions sourced

run
annotateHOP setting nominal HoP
annotateHOP setting clausal HoP

1: ==>S: S3 (state no-change)
    proposing operator O1

2: O: O1 (sentenceInputLink)
    proposing operator O4

3: O: O4 (sentenceInputLink)
    proposing operator O5

4: O: O5 (sentenceInputLink)
    proposing operator O6

5: O: O6 (sentenceInputLink)
    proposing operator O7

6: O: O7 (sentenceInputLink)
    exhausted input
    proposing operator O8 to access lexitem '*the*'

7: O: O8 (getword)
    accessing word: '*the*'
    sidestepping wnet access for '*the*'
    Receivers: d the at X-bar
    Assigners: d the up at X-bar
    Assigners: d the up2 at X-bar
    creating synobj
    tagging access completed
    adding synobj pointer to profiles
    proposing operator O10 to access lexitem '*lawyers*'

8: O: O9 (project)
    projecting to dP

9: O: O10 (getword)
    accessing word: '*lawyers*'
    loading lemmas for '*lawyers*'
    loading senses for '*lawyers*'
    Receivers: n lawyers at X-bar
    Assigners: n lawyers up at X-bar
    Assigners: n lawyers up2 at X-bar
    creating synobj
    found an inflected noun
    creating semobj for '*lawyers*' w\n-person
    adding synobj pointer to profiles
    creating semProfile for '*lawyers*' w\n-person
    adding semobj pointer semprofiles
    selecting sense 1 for current hypothesis
    tagging access completed
    proposing operator O12 to access lexitem '*pursue*'

10: O: O11 (project)
    projecting to nP

11: O: O13 (hop)
    extending n to nn
    projecting nn to nnP
    proposing operator O14 to graftComp

12: O: O14 (graft)
    doing graftComp

13: O: O12 (getword)
    accessing word: '*pursue*'
    loading lemmas for '*pursue*'
    loading senses for '*pursue*'
    loading frames for '*pursue*'
    Receivers: v pursue at X-bar
    Assigners: v pursue up at X-bar
    Assigners: v pursue up2 at X-bar
    creating synobj
    found an uninflected verb
    creating semobj for '*pursue*' w\v-contact & transitive
    creating semobj for '*pursue*' w\v-motion & transitive
    creating semobj for '*pursue*' w\v-social & transitive
    adding synobj pointer to profiles
    adding synobj pointer to profiles
    creating semProfile for '*pursue*' w\v-contact
    creating semProfile for '*pursue*' w\v-motion
    creating semProfile for '*pursue*' w\v-social
    creating semProfile for '*pursue*' w\v-social
    adding semobj pointer to semprofiles
    adding semobj pointer to semprofiles
    adding semobj pointer to semprofiles
    selecting sense 1 for current hypothesis

adding semobj pointer to semprofiles
tagging access completed
proposing operator O17 access lexitem '*convictions*'

14: O: O16 (project)
projecting to vP

15: O: O18 (hop)
extending v to vv
projecting vv to vvP

16: O: O19 (hop)
extending vv to t
projecting t to tP
proposing operator O20 to graftSpec
proposing operator O38 to fuseExternal

17: O: O20 (graft)
doing graftSpec
proposing operator O44 to fuseExternal

18: O: O47 (hop)
extending t to c
projecting c to cP

19: O: O23 (fuse)
doing fuseExternal

20: O: O17 (getword)
accessing word: '*convictions*'
loading lemmas for '*convictions*'
loading senses for '*convictions*'
Receivers: n convictions at X-bar
Assigners: n convictions up at X-bar
Assigners: n convictions up2 at X-bar
creating synobj
found an inflected noun
creating semobj for '*convictions*' w\n-act
creating semobj for '*convictions*' w\n-cognition
adding synobj pointer to profiles
creating semProfile for '*convictions*' w\n-act
creating semProfile for '*convictions*' w\n-cognition
adding semobj pointer to semprofiles
selecting sense 1 for current hypothesis
adding semobj pointer S49 to semprofiles
tagging access completed
proposing operator O49 to access lexitem '.'

21: O: O48 (project)
projecting to nP

22: O: O50 (hop)
extending n to nn
projecting nn to nnP

23: O: O51 (hop)
extending nn to d
projecting d to dP
proposing operator O52 to graftComp

24: O: O52 (graft)
doing graftComp
proposing operator O53 to fuseInternal

25: O: O53 (fuse)
doing fuseInternal

26: O: O49 (getword)
accessing word: '.'
reached end of sentence

27: O: O55 (completion)
Interrupt received.
This agent halted.

Bibliography

Adger, David. 2003. Core Syntax: A Minimalist Approach. Oxford University Press.

Bader, Marcus, and Josef Bayer. 2006. Introducing the Human Sentence Processing Mechanism. Case and Linking in Language Comprehension, Studies in Theoretical Psycholinguistics, vol. 34, 19–47. Elsevier Science.

Ball, Jerry, Christopher Myers, Andrea Heiberg, Nancy Cooke, Michael Matessa, and Mary Freiman. 2009. The Synthetic Teammate Project. Proceedings of the 18th Annual Conference on Behavior Representation in Modeling and Simulation.

Bever, T. 1970. The Cognitive Basis for Linguistic Structure. Cognition and the Development of Language, ed. by J. R. Hayes. New York: Wiley.

Blache, Philippe, and Jean yves Morin. 1990. Bottom-up filtering: a parsing strategy for GPSG. Proceedings of the International Conference on Computational Linguistics.

Boland, Julie, and Heather Boehm-Jernigan. 1998. Lexical constraints and prepositional phrase attachment. Journal of Memory and Language 39.684–719.

Boston, Marisa F., John T. Hale, Shravan Vasishth, and Reinhold Kliegl. 2011. Parallel processing and sentence comprehension difficulty. Language and Cognitive Processes 26.301–349.

Brill, Eric, and Phil Resnik. 1994. A transformation-based approach to prepositional phrase attachment disambiguation. Proceedings of the Fifteenth International Conference on Computational Linguistics (COLING-1994).

Caramazza, Alfonso, Alessandro Laudanna, and Cristina Romani. 1988. Lexical access and inflectional morphology. Cognition 28.297–332.

Charniak, Eugene. 2000. A maximum-entropy-inspired parser. Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference, NAACL 2000, 132–139. Association for Computational Linguistics.

Chomsky, Noam. 1965. Aspects of the Theory of Syntax. MIT Press.

Chomsky, Noam. 1986. Barriers. MIT Press.

Chomsky, Noam. 1993. Lectures on Government and Binding: the Pisa lectures. Studies in Generative Grammar. Mouton de Gruyter.

Chomsky, Noam. 1995. The Minimalist Program. MIT Press.

Chomsky, Noam, and Harold Lasnik. 1993. The theory of principles and parameters. Syntax: Ein Internationales Handbuch Zeitgenossischer Forschung. An International Handbook of Contemporary Research, ed. by Joachim Jacobs, Arnim von Stechow, and Wolfgang Sternefeld. Walter de Gruyter.

Collins, Michael, and James Brooks. 1995. Prepositional Phrase Attachment through a Backed-Off Model. Proceedings of the Third Workshop on Very Large Corpora, 27–38.

Crocker, Matthew, and Thorsten Brants. 1999. Incremental probabilistic models of human linguistic performance. The 5th Conference on Architectures and Mechanisms for Language Processing. Edinburgh, U.K.

Davies, Mark. 2004. BYU-BNC: The British National Corpus. Available online at http://corpus.byu.edu/bnc.

Davies, Mark. 2009. The 385+ Million Word Corpus of Contemporary American English (1990-2008+): Design, Architecture, and Linguistic Insights. International Journal of Corpus Linguistics 14.159–90.

Davies, Mark. 2010. The Corpus of Historical American English (COHA): 400+ million words, 1810-2009. Available online at http://corpus.byu.edu/coha.

De Marneffe, Marie-catherine, Bill Maccartney, and Christopher D. Manning. 2006. Generating Typed Dependency Parses from Phrase Structure Parses. Proceedings of the fifth international conference on Language Resources and Evaluation, 449–454.

Derbinsky, Nate, and John E. Laird. 2010. Extending Soar with Dissociated Symbolic Memories. Proceedings of the 1st Symposium on Human Memory for Artificial Agents, 31–37. Leicester, UK.

Earley, Jay. 1983. An efficient context-free parsing algorithm. Communincations of the Association for Computing Machinery 26.57–61.

Edelman, Simon, and Morten H. Christiansen. 2003. How seriously should we take minimalist syntax? Trends in Cognitive Sciences 7.60–61.

Fellbaum, Christiane. 1998. WordNet: An electronic lexical database. MIT Press.

Ferstl, Evelyn. 1994. Context effects in syntactic ambiguity resolution: The location of prepositional phrase attachment. Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society, 295–300. Lawrence Erlbaum Associates.

George, Gary R., Richard A. Breitbach, Rebecca B. Brooks, Robert Steffes, Herbert H. Bell, and Cecil C. Bruhl. 1999. Synthetic forces in the Air Force command and control distributed mission training environment, current and future. Proceedings of the 8th Conference on Computer Generated Forces and Behavioral Representations, 319–331.

Gorrell, Paul. 1989. Establishing the loci of serial and parallel effects in syntactic processing. Journal of Psycholinguistic Research 18.61–73.

Gorrell, Paul. 1991. Subcategorization and sentence processing. Principle based parsing: Computation and psycholinguistics, Studies in Theoretical Pyscholinguistics. Elsevier Science.

Hale, John. 2004. The information-processing difficulty of incremental parsing. Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together, 58–65. Stroudsburg, PA: Association for Computational Linguistics.

Hindle, Donald, and Mats Rooth. 1993. Structural ambiguity and lexical relations. Computational Linguistics 19.103–120.

Hutchins, John. 1999. Retrospect and prospect in computer-based translation. Machine Translation Summit VII, 30–34.

Jackendoff, Ray. 1977. X-bar Syntax: A Study of Phrase Structure. MIT Press.

Jackendoff, Ray. 1992. Semantic Structures. MIT Press.

Jurafsky, Daniel. 1996. A probabilistic model of lexical and syntactic access and disambiguation. Cognitive Science 20.137–194.

Jurafsky, Daniel, and James H. Martin. 2000. Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition. Upper Saddle River, NJ: Prentice Hall PTR.

Knuth, Donald E. 1997. The art of computer programming, volume 1: Fundamental algorithms. Redwood City, CA: Addison Wesley Longman Publishing.

Laird, John E. 2008. Extending the Soar cognitive architecture. Proceedings of the Artificial General Intelligence Conference, 224–235. IOS Press.

Laird, John E. 2012. The Soar cognitive architecture. MIT Press.

Lasnik, Howard. 2002. The minimalist program in syntax. Trends in Cognitive Sciences 6.432–437.

Lehman, Jill Fain, Richard L. Lewis, and Allen Newell. 1991. Natural Language Comprehension in Soar: Spring 1991. CMU-CS-91-117. Department of Computer Science, Carnegie Mellon University.

Lewis, Richard L. 1992. Recent developments in the NL-Soar garden path theory. Tech. Rep. CMU-CS-92-141, Carnegie Mellon University.

Lewis, Richard L. 1993. An architecturally-based theory of human sentence comprehension. Carnegie Mellon University dissertation.

Lewis, Richard L. 1996. Interference in short-term memory: The magical number two (or three) in sentence processing. Journal of Psycholinguistic Research 25.93–115.

Lewis, Richard L. 1998. Reanalysis and limited repair parsing: leaping off the garden path. Reanalysis in sentence processing, ed. by Janet Dean Fodor and Fernanda Ferreira. Kluwer Academic Publishers.

Lewis, Richard L. 2000. Specifying architectures for language processing: Process, control, and memory in parsing and interpretation. Architectures and mechanisms for language processing, ed. by Matthew W. Crocker, Martin Pickering, and Charles Clifton Jr. Cambridge University Press.

Lonsdale, Deryle. 2001. An Operator-based Integration of Comprehension and Production. LACUS Forum XXVII, 123–132. Linguistic Association of Canada and the United States.

Lonsdale, Deryle. 2006. Learning in minimalism-based language modeling. Proceedings of the 28th Annual Meeting of the Cognitive Science Society, 26–51. Mahwah, NJ: Lawrence Erlbaum Associates.

Lonsdale, Deryle, Jamison Cooper-Leavitt, Warren Casbeer, Rebecca Madsen, and LaReina Hingson. 2008. An incremental minimalist parser. Unpublished manuscript.

Lonsdale, Deryle, LaReina Hingson, Jamison Leavitt-Cooper, Warren Casbeer, and Rebecca Madsen. 2006. XNL-Soar, Incremental Parsing, and the Minimalist Program. 19th Annual CUNY Conference on Human Sentence Processing.

Lonsdale, Deryle, and Michael Manookin. 2004. Combining Learning Approaches for Incremental On-line Parsing. Proceedings of the 6th International Conference on Conceptual Modeling, 160–165. Lawrence Erlbaum Associates.

Lonsdale, Deryle, and C. Anton Rytting. 2006. An operator-based account of semantic processing. Acquisition and Representation of Word Meaning: Theoretical and Computational Perspectives, ed. by Alessandro Lenci, Simonetta Montemagni, and Vito Pirrelli, Linguistica Computazionale, vol. XXII-XXIII, 117–137. Pisa/Roma: Istituti Editoriali e Poligrafici Internazionali.

Marcus, Mitchell, Grace Kim, Mary Ann Marcinkiewicz, Robert Macintyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The Penn Treebank: Annotating Predicate Argument Structure. In Proceedings of ARPA Human Language Technology Workshop, 114–119.

Melby, Alan, Alan Manning, and Leticia Klemetz. 2007. Quality in translation: A Lesson for the Study of Meaning. Linguistics and the Human Sciences .

Mel'čuk, Igor. 1988. Dependency Syntax: Theory and Practice. Albany, NY: State University of New York Press.

Miller, George A., Claudia Leacock, Randee Tengi, and Ross T. Bunker. 1993. A semantic concordance. Proceedings of the Workshop on Human Language Technology, 303–308. Association for Computational Linguistics.

Napoles, Courtney, Matthew R. Gormley, and Benjamin Van Durme. 2012. Annotated English Gigaword. Linguistic Data Consortium, Philadelphia.

Newell, Alan. 1988. Annotated models. Unpublished.

Newell, Alan. 1990. Unified theories of cognition. Harvard University Press.

Newell, Allen, Paul Rosenbloom, and John Anderson. 1981. Mechanisms of skill acquisition and the law of practice, 1–55. Erlbaum.

Newell, Allen, J. C. Shaw, and Herbert A. Simon. 1959. Report on a general problem-solving program. Congress of the International Federation for Information Processing, 256–264.

Newell, Allen, and Herbert A. Simon. 1976. Computer science as empirical inquiry: symbols and search. Communications of the Association for Computing Machinery 19.113–126.

Nivre, Joakim, and Johan Hall. 2005. Maltparser: A language-independent system for data-driven dependency parsing. In Proceedings of the Fourth Workshop on Treebanks and Linguistic Theories, 13–95.

Oaks, Dallin D. 2012. Structural Ambiguity in English: An Applied Grammatical Inventory. Bloomsbury.

Onifer, William, and David A. Swinney. 1981. Accessing lexical ambiguities during sentence comprehension: effects of frequency of meaning and contextual bias. Memory and Cognition 9.225–236.

Pritchett, Bradley. 1992. Grammatical competence and parsing performance. University of Chicago Press.

Ratnaparkhi, Adwait, Jeff Reynar, and Salim Roukos. 1994. A maximum entropy model for prepositional phrase attachment. Proceedings of the Human Language Technology Workshop, 250–255. Plainsboro, N.J.: ARPA.

Resnik, Philip. 1995. Using information content to evaluate semantic similarity in a taxonomy. Proceedings of the 14th International Joint Conference on Artificial Intelligence, 448–453. San Francisco, CA: Morgan Kaufmann Publishers.

Rubinoff, Robert, and Jill Fail Lehman. 1994. Natural language processing in an IFOR pilot. Collected Papers of the Soar/IFOR Project: Spring 1994. Carnegie Mellon University.

Rytting, C. Anton. 2000. Semantic Class Disambiguation in Natural Language Soar. Brigham Young Unviersity Honors Thesis.

Rytting, C. Anton, and Deryle Lonsdale. 2001. Integrating WordNet with NL-Soar. Proceedings of the Workshop on WordNet and other lexical resources: Applications, extensions, and customizations, 162–164. North American Association for Computational Linguistics.

Schafer, Amy. 1998. Prosodic parsing: The role of prosody in sentence comprehension. University of Massachusetts, Amherst dissertation.

Schelstraete, Marie. 1996. Definiteness and prepositional phrase attachment in sentence processing. Current Psychology of Cognition 15.463–486.

Schuetze, Carson, and Edward Gibson. 1999. Argumenthood and English prepositional phrase attachment. Journal of Memory and Language 40.409–431.

Simon, Herb. 1996. Computational Theories of Cognition. The Philosophy of Psychology, chap. 12, 93–142. London, England: SAGE Publications.

Spivey-Knowlton, Michael, and Julie Sedivy. 1995. Resolving attachment ambiguities with multiple constraints. Cognition 55.227–267.

Straub, Kathleen. 1998. The production of prosodic cues and their role in the comprehension of syntactically ambiguous sentences. University of Rochester dissertation.

Turing, Alan M. 1950. Computing machinery and intelligence. Mind LIX.433–460.

Vasishth, Shravan, and Richard L. Lewis. 2004. Modeling Sentence Processing in ACT-R. Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together. Association for Computational Linguistics.

Vasishth, Shravan, and Richard L. Lewis. 2006. Symbolic models of human sentence processing. Encyclopedia of Language and Linguistics, 410–419. Elsevier Science.

Yngve, Victor. 1955. Syntax and the Problem of Multiple Meaning. Machine Translation of Languages, ed. by William N. Locke and A. Donald Booth, 208–226. MIT Press.