52

Automatic Grammar Induction from Semantic Parsing

by

Debajit Ghosh

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1998

Author .....
Department of Electrical Engineering and Computer Science
May 21, 1998

Certified by .....
Dr. James R. Glass
Principal Research Scientist
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Automatic Grammar Induction from Semantic Parsing

by

Debajit Ghosh

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 1998, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, we investigate an approach for grammar induction that relies on semantics to drive the automatic learning of syntax rules. Specifically, we develop a semantic parser, which parses utterances based on "meaning," rather than syntax, by combining only words and phrases that satisfy a given set of semantic constraints. We subsequently extract a syntactic grammar from the resulting semantic-level phrases, trying various approaches to generalize this grammar. We evaluate the learned grammar utilizing two sets of experiments, restricting our test sets to semantically valid utterances. First, we use the grammar to parse new utterances from the same domain in which it was learned; the learned grammar covers 98% of the utterances handled by the semantic constraints. Second, we parse utterances from a new domain, assessing the portability of the grammar. Here, the grammar covers 85% of the semantically valid utterances. Semantic parsing proves to be a very powerful and useful mechanism, independent of syntax, providing an utterance's "meaning representation" directly. Furthermore, our experiments illustrate that this technique has potential for automatically developing portable grammars, making the task of moving an understanding system to a new domain easier.

Company Supervisor: Dr. David Goddeau
Title: Research Staff, DIGITAL Cambridge Research Laboratory

Thesis Supervisor: Dr. James R. Glass
Title: Principal Research Scientist

Acknowledgments

I am very grateful to the many people who have influenced this research and made this thesis possible. First and foremost, I would like to thank my thesis supervisors, Dr. David Goddeau and Dr. James Glass. DG provided me with an immeasurable amount of help, ideas, and encouragement, and he patiently listened to countless "quick questions" while I worked on this thesis. Jim provided invaluable guidance and advice, making sure I concentrated on the important ideas and "big picture" overall, as well as making sure I had the necessary resources and data. I cannot thank either of them enough.

I also want to thank DIGITAL's Cambridge Research Laboratory for granting the funding and support for this thesis research, as well as providing me with exciting research projects during the past few summers. Dr. Robert Iannucci, the director of DEC CRL, always ensured that other interns and myself were always provided with intellectually stimulating projects. Likewise, the administrative and system staff furnished me with the necessary resources during each summer. I have also thoroughly enjoyed interacting and conversing with the research scientists in this lab.

I would also like to thank MIT's Spoken Language Systems group for providing me with the data I needed for my experiments. I enjoyed getting advice from and conversing with the staff and students in that group.

During the past few years, my supervisors have helped me considerably. Chris Weikart taught me many valuable skills regarding system design and development during my first two summers. Dr. David Goddeau provided great supervision during this past summer and while I worked on my thesis, introducing me to natural language. Dr. William Goldenthal brought me into the VI-A program at DEC CRL and opened my eyes to the fascinating field of speech recognition, while inspiring me to challenge myself continually.

Finally, I want to thank my friends and family, in particular my parents, for all of their support during the past five years at MIT, especially during this last year. The continual support of my parents made everything I have done possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Natural language processing represents a fundamental component of many speech recognition and language understanding systems. As shown in Figure 1-1, the output of a speech recognizer may serve as the input of the natural language unit. Given an application domain or context and a corresponding grammar, this natural language unit generates a meaning representation for its input text; the unit might also feed back into the recognizer to provide constraints on what words are recognized. An interpreter evaluates the output meaning representation and processes it accordingly (i.e., retrieving requested data from a database). A dialog manager can use this evaluation to generate output (such as generating a spoken response, presenting requested data, or posing a query asking the user for more information if necessary); it can also provide feedback to the system, changing the current context to improve the recognition and understanding of words that make "sense" in this context.

This thesis investigates a novel approach for making it easier to define the natural language component. Specifically, we examine a way to learn the grammar used in natural language in an automatic fashion. After providing the purpose and motivation for this research, this chapter describes our research focus, stating our goals and outlining the various components and stages of our work.

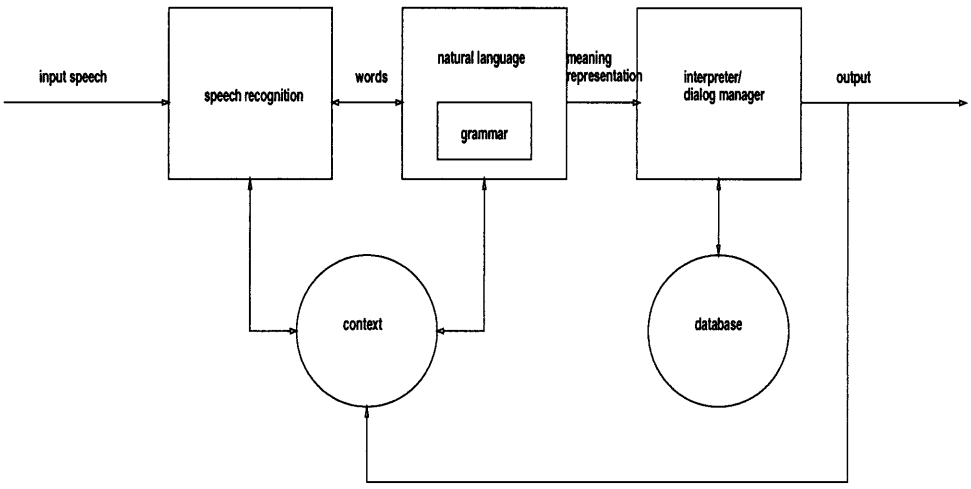


Figure 1-1: Major components of a speech recognition and understanding system. This figure displays the various components of a speech recognition and understanding system. The grammar, as part of the natural language unit, can constrain the output of the recognizer and generate meaning representations for input utterances, using the current context. An interpreter and dialog manager can then generate the corresponding output, perhaps retrieving requested information from a database.

1.1 Purpose and Motivation

Natural language processing has traditionally aspired to provide models for the way humans communicate and has attempted to produce mechanisms for "understanding" input sequences of words from a corpus of text or the output of a speech recognition system. Of course, other uses exist for these models, such as providing constraints on the output of a speech recognition engine. Thus, one can easily imagine the need for robust, useful models and mechanisms in this field.

Syntactic grammars represent a central component of these systems, providing useful constraints and structure. This structure can help guide the recognition search or direct a parser in language understanding mechanisms. A parsing system may utilize a hand-written grammar to generate a syntax tree and subsequently compute some form of semantic, or meaning-based, representation of the nodes of the tree. Interpreters can then use this representation in understanding the sentence and in preparing responses to a query or command.

However, several problems exist with this syntax-centric approach to understanding. First, writing a grammar for a language is by no means trivial. Indeed, writing a grammar manually is rather time-consuming and laborious. Second, even a supposedly "complete" grammar may not be able to parse all sentences, and this lack of a complete parse for a sentence may prevent the system from computing its semantics, effectively causing it to *lose* information and understanding of its input. Indeed, quite often input, such as that from a spoken language system, strays from grammatical rules, making it difficult for syntax-based systems to parse and subsequently compute semantics for those sentences.

The field of grammar induction attempts to address some of these problems. By learning grammars automatically, grammar induction mechanisms can alleviate the tediousness of writing grammars by hand. In addition, many techniques entail a data-driven approach and can learn grammars based on the way people speak, rather than adhering to strict or formal rules of grammar.

However, many grammar induction techniques suffer from a variety of criticisms. They can involve a high degree of complexity, depend on supervision for training and learning, or lack portability across domains. Some techniques even generate illogical or unusable grammar rules.

Additionally, one must consider that language serves to convey "meaning," not syntax. Typically, syntax is at most of intermediate interest, as a guide for further analysis, such as semantic computation and interpretation. Humans can understand many agrammatical sentences without much problem (other than perhaps raising an eyebrow or correcting grammatical creativity). Accordingly, several techniques have been developed and implemented that attempt to allow for robust parsing when grammar rules fail to complete a syntactic parse. One can even imagine using semantics for the parsing itself, in order to provide this desired increase in robustness to complex or agrammatical inputs.

There has been some work in using semantics directly in accepting and understanding input utterances. However, this is a relatively new field, and thus, no truly ubiquitous approaches exist. Certainly, room exists for new approaches or variations

in attempting to accept or parse utterances semantically.

One such variation could actually couple semantic parsing with grammar induction, forming a *semantic* parse tree that could be used to guide the induction process. This could allow for the learning of more robust and logical grammar rules than an unguided technique. This thesis will attempt to assess exactly this issue of whether or not it is feasible to use semantic parsing to drive grammar induction techniques and determine the usefulness of the resulting grammar.

1.2 Research Focus

1.2.1 Goals

This thesis will focus on investigating the feasibility and utility of semantic parsing as a tool for grammar induction. Specifically, we will examine the questions of what kind of syntactic rules can be learned from semantic parsing, as well as how well these rules perform in syntactic parsing experiments. In addition, we will examine and determine how useful these rules are in domains other than the one in which they are derived. Some previous work in semiautomatic or automatic grammar learning techniques seem to be restricted to their original domain, so this measurement of portability should provide us with a further assessment of this grammar induction technique and the resulting learned grammar.

1.2.2 Overview of Research

This thesis will begin by providing the relevant background information in Chapter 2. Chapter 3 presents the overall mechanism of semantic parsing and provides the details of our actual implementation. Next, Chapter 4 describes two related grammar induction techniques which we investigate. Chapter 5 discusses our experiments in syntactic parsing and portability, as well as the results of those experiments. Finally, Chapter 6 provides the conclusions we have drawn and suggests future work that can improve or extend the results of our research.

Chapter 2

Background

Our research investigates the development of a new grammar induction technique using semantics and semantic parsing as its basis. This chapter provides an overview of related research as well as other relevant background information. We first discuss several grammar induction techniques. Next, we loosely define semantics and review other works aimed at extracting semantics directly from an utterance. Finally, we conclude this chapter by describing how some existing systems parse sentences syntactically and subsequently compute semantics from the corresponding parse tree.

2.1 Grammar Induction

Grammar induction is an extensively researched field that attempts to derive and generalize the rules necessary to accept utterances from a specified language. Numerous techniques have been used with varying degrees of success in this field, many of which are examples of pattern recognition and learning. These include clustering-based techniques [4], domain-specific heuristic methods [14], artificial neural networks (ANNs) [5, 13], hidden Markov models (HMMs) [14], and Bayesian networks [11].

Much development and effort revolves around clustering-based approaches. These approaches begin with a simple, overly specific grammar (to a training set), which is iteratively improved through means of merging syntactic units based on a given distance metric. Indeed, some inference techniques begin with a grammar containing

all sentences in the training set and then cluster the syntactic units together until they obtain a satisfactory, generalized structure and grammar [4, 12].

One possible distance metric involves computing distributional similarity, which measures the divergence of the probability distributions of two word or phrasal units [3]. One technique uses this metric to derive a relatively uncomplicated grammar and subsequently acquires phrase structure automatically by using entropy or other metrics to guess which pairs of units can form a phrase [3]. Another metric uses error statistics (differences between learned models and training data) to achieve iterative improvement of the acquired grammar [14]. As one can imagine, many other metrics and variants exist; finding a good distance metric represents another interesting research problem.

Recently, researchers have begun to use artificial neural networks as the basis for some grammar induction approaches [5, 13]. While some of these depend on supervision in order to train the weights in the network, others are mostly or completely unsupervised. In one such unsupervised system, the networks are supplied with simple input sentences, words, and phrases, and after training, the system extracts a grammar from the weights connecting the nodes of the network [5]. However, the researchers behind that work admit that this grammar contains some illogical and unusable rules.

Some of these techniques depend on pre-tagged lexicons (denoting the parts of speech for each word), while others attempt to learn classes for words automatically [2, 12]. Unfortunately, while aesthetic and desirable, the latter, more automatic approach does not always produce results which port to new or different domains, as the learned word classes may only apply to words in one domain but not another. Also, since many grammar-learning and structure-acquisition techniques rely strictly on co-occurrence and distributional statistics, not linguistic cues and properties, these techniques can potentially acquire unportable or even illogical rules. Therefore, to overcome some of these criticisms, we consider using semantics to guide the grammar induction process, as we will describe shortly.

2.2 Semantics and Semantic Parsing

Semantics refer to any level of meaning representation, from a simple argument-structure to intricate and complex models of the world (or models of a specific domain). The actual level of detail contained in this representation depends on the implementation and the boundaries drawn between this intermediate language and the duties of the interpreter which evaluates it and generates a response. There are few standards for this intermediate representation; semantic representation is typically system and implementation-specific. Additionally, semantics themselves are inherently domain-specific, as some words often have different meanings in different contexts. Thus, performing any absolute comparison of semantics can be rather difficult. Nevertheless, we will describe some prior works that use some form of semantics directly in accepting a sentence.

Some work has been done in using transducers [14], a broad abstraction for a device which translates input from one language into output in another one (here, a semantic representation language); HMMs or ANNs can be used in the implementation of this abstract device. One HMM-based approach is trained using the utterances as input and pre-defined "meaning frames" as output; the resulting system can then fairly accurately create meaning frames for new test sentences [10].

An interesting ANN-based approach consists of a system which is simply supplied with the words in an input sentence and a semantic category into which the sentence falls [6]. The system then automatically computes associations (the weights) between various words and categories. The simple, one-layer version of this network actually has no dependencies on word order and only computes whether enough words which are associated with a certain class are present in the input utterance. The more complicated two-layer network does contain some type of phrase structure [7]; however, the meaning or utility of this type of semantic phrase is not entirely straightforward or apparent other than for use in performing simple semantic classification.

Therefore, neither of these approaches provides any semantic units or phrases which seem like they would be useful for driving grammatical inference techniques.

Other related approaches, however, stray from using semantics alone and instead use semantics to *complement* syntactic parsing. This allows for the simplification of some otherwise complex rules (i.e., rules with many different forms or optional components) and makes the system more robust to agrammatical inputs [1]. It is even suggested (though not tried or implemented) that this technique allows for learning of syntax from the semantics used to complete a parse. Accordingly, our thesis research attempts to develop a different technique for semantic parsing – one which can provide useful information for driving grammar induction systems.

2.3 Basic NLU System

2.3.1 Components and Stages

To extract the semantics of an utterance, many systems first perform a syntactic parse and generate a syntax tree for the utterance. Next, they compute the semantics of the different components of the syntax tree, working up to the root of the tree to generate its overall semantics. This section will describe each of the various aspects of such a system in more detail.

Syntactic Parsing A syntactic parsing system typically uses a set of context free grammar (CFG) rules, as shown in Figure 2-1, and through a series of rewrites, transforms a sentence into its syntactic structure. For example, suppose a domain consists of utterances such as *"cheapest flight from boston."* If the parts of speech for the component words in an utterance are supplied in a lexicon, or dictionary, the listed rules can be used to generate the parse tree shown in Figure 2-2. In this figure, one can observe *"cheapest"* combining with *"flight"* into a noun phrase, *"from"* combining with *"boston"* into a prepositional phrase, and those two phrases combining with each other into another noun phrase.

Semantics For understanding, a parsing system utilizes more than just parts of speech and syntactic rules. The system also needs an input set of semantics for words

Left-hand Side		Right-hand Side
NP	$\longrightarrow$	Adj N
PP	$\longrightarrow$	P Name
NP	$\longrightarrow$	NP PP

Figure 2-1: Examples of context free grammar rules.
Listing of the CFG rules necessary to parse *"cheapest flight from boston."*

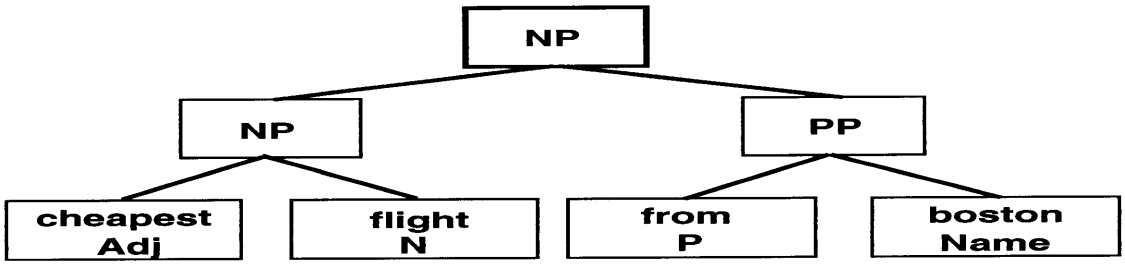


Figure 2-2: Syntactic parse tree for *"cheapest flight from boston."*
An illustration of the syntactic parse tree obtained using the CFG rules in Figure 2-1 for the utterance *"cheapest flight from boston."* *"cheapest"* and *"flight"* combine into a noun phrase, *"from"* and *"boston"* combine into a prepositional phrase, and those phrases combine into another noun phrase.

in the lexicon as well as a set of semantic functions associated with each syntactic rule. The semantics of most words and phrases can be represented by simple semantic frames; each frame consists of a head, denoting the main concept, and possibly a set of key-value pair modifiers further describing this concept, as shown in Figure 2-3. In addition, integers and strings can be used for simpler concepts. For example, numbers (or parts of the representation of numbers) can contain raw integers, like `42`. Also, very simple concepts, such as "`o_clock`," may be represented by a string rather than a frame (since there is no need for further information or modification). Typically, we define most words to have frame-based semantics, and sometimes those frames contain integer or string-based semantics within them.

In order to combine the semantics of each component in the parse tree, the system also needs a set of semantic functions. These functions are associated with specific syntactic rules, as shown in Figure 2-4. Each function takes a set of argument semantics and outputs a new set of semantics produced by combining these inputs. Specifically, it takes the semantics of each token of the associated syntax rule and considers them in the specified order. For example, **f (0 1)** dictates that the first token, at index 0, should be the first argument to the function f, and the second token, at index 1, should be the second argument; similarly, **f (1 0)** dictates that the second token, at index 1, should be the first argument, and the first token, at index 0, should be the second argument. Using this ordering, it generates a new set of semantics for the overall syntactic unit.

For example, we might associate the semantic function **add_topic (0 1)** with the syntax rule PP $\rightarrow$ P Name. Accordingly, to compute the semantics for the resulting prepositional phrase, the system needs to evaluate **add_topic** with the arguments of the semantics of the preposition (at index 0 in the rule) and the name (index 1), as shown in Figure 2-5. This evaluation results in adding the semantics of the name under the key of "`topic`" in the semantic frame for the preposition. Similarly, the function **add_pred**, associated with NP $\rightarrow$ Adj N, computes the semantics of the resulting noun phrase by combining the semantics of the noun (index 1) with those of the adjective (index 0), adding the semantics of the adjective as a predicate of the

Word/Phrase	Semantic Frame Representation
boston	{city :name ``boston''}
flight	{flight}
cheapest flight from boston	{flight :from { city :name ``boston''} :pred { cheap :type ``superlative''} }

Figure 2-3: Examples of semantic frames.
This figure displays several examples of meaning representations, in the form of semantic frames, for various words and phrases. Specifically, it shows the frames for the words *"boston"* and *"flight,"* as well as the phrase *"cheapest flight from boston,"* which would be computed during parsing. The semantics for *"boston"* represents a classic semantic frame, consisting of a head (i.e., "city") and key-value modifiers (i.e., the string "boston" under the key of "name").

Left-hand Side		Right-hand Side	Semantic Function
NP	$\longrightarrow$	Adj N	**add_pred (1 0)**
PP	$\longrightarrow$	P Name	**add_topic (0 1)**
NP	$\longrightarrow$	NP PP	**add_marked (0 1)**

Figure 2-4: Example of CFG with associated semantic functions.
The CFG rules that cover *"cheapest flight from boston,"* augmented with semantic functions. These functions are used by the parser to compute the semantics of each resulting non-terminal.

semantics of the noun. Finally, the overall semantics of the top level unit, another noun phrase, can be generated by evaluating **add_marked** with arguments of the semantics of the embedded noun phrase and prepositional phrase, respectively.

Constraints When computing meaning by combining the semantics of input syntactic tokens, such as adjectives, nouns, and phrases, one must consider that not all units can and should be combined. Only certain combinations "make sense"; thus, it is highly desirable to define a set of constraints that restrict which units can be combined meaningfully. A semantic function can check these constraints in order to determine if its arguments can combine, or if the function should instead not produce any semantics at all for a specific combination. For instance, while it is meaningful

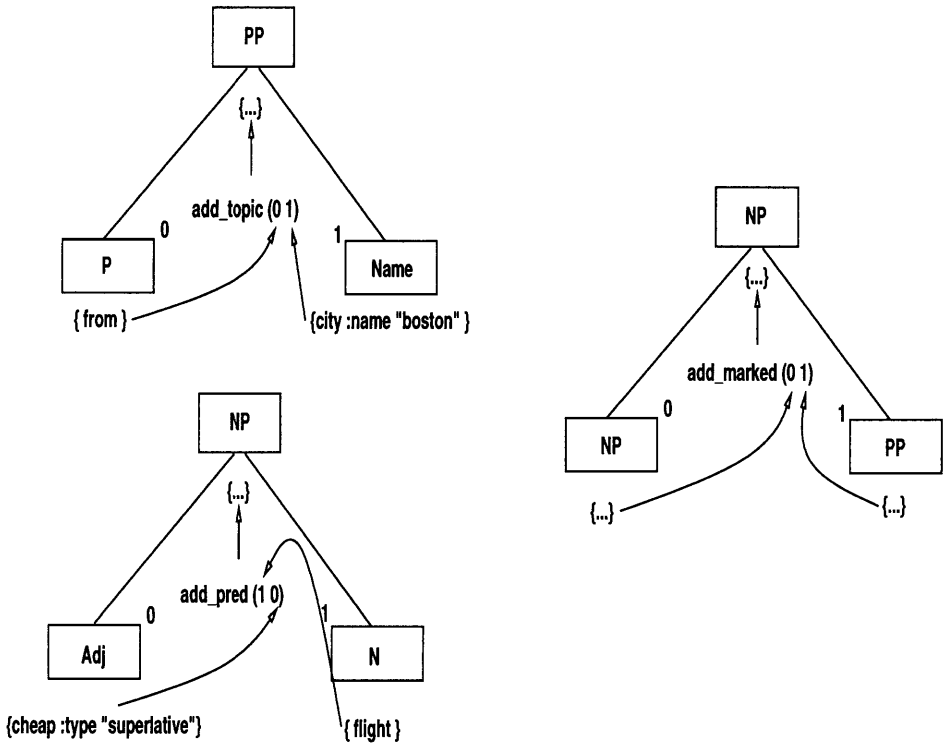


Figure 2-5: Examples of computing semantics (in a specified order) for parts of the syntax tree.

This figure displays how the parser uses semantic functions to compute the semantics of each non-terminal. For the prepositional phrase, the system combines the semantics of *"from"* and *"boston"* using the function **add_topic**, in that order. Likewise, the system combines the semantics of *"flight"* and *"cheapest"* using **add_pred**. Finally, the top-level noun phrase's semantics are computed by combining the semantics of the embedded noun phrase and the prepositional phrase, respectively, via **add_marked**.

Constraints for flight
{ flight :args ({ :type loc :role from } { :type loc :role to }) :parent ''obj'' } { cost :args ({ :type flight :role subj }) }

Figure 2-6: Examples of semantic constraints.
Some constraints used for utterances like *"cheapest flight from boston."* These constraints allow *"flight"* to accept arguments of a source, a destination, and a cost.

to combine the adjective *"cheapest"* with the noun *"flight,"* it may not be meaningful to combine *"soft"* with *"flight."* Semantic constraints can ensure that all computed semantics are reasonable and can be evaluated, rather than being nonsensical or uninterpretable, such as a *"soft flight."* These constraints can be denoted simply by a list of words and valid arguments that can modify those words. Figure 2-6 shows an example of possible constraints, stating that a flight can take arguments describing its source and its destination. In addition, other constraints can be used to denote that a cost can serve as a predicate (another type of argument) of flight.

2.3.2 Parser Implementation

We use an all-parses bottom-up chart parser for our various parsing experiments. This type of parser maintains a data structure called a chart [9] to perform book-keeping of partial results and keep track of matches between the input words and phrases and the defined syntactic rules. In addition, the parser also records its current position in each rule (usually denoted by a "dot" in output; the dot sits in front of the next component it tries to match). The chart stores matches as edges, which span zero or more words and contain the parts of speech of constituent words as well as a label denoting the represented rule. As each word in an utterance is being considered, the system looks up the part of speech for the word from its lexical entry, creates an edge containing just that word, and adds that edge to the chart. The parser also creates

active (not completed)	NP→Adj . N	
inactive (completed)	Adj (cheapest)	

active (not completed)	NP→Adj . N	
	NP→NP . PP	
inactive (completed)	Adj (cheapest)	
		N (flight)
	NP→Adj N .	

0 cheapest 1 flight 2

Figure 2-7: Snapshot of the chart during parsing.
This figure displays the state of the chart during two stages of the parsing process for the phrase *"cheapest flight."* This data structure records the parser's current position in each rule with a "." and separately maintains lists of complete and incomplete (rule-based) edges. The top chart shows the state after the parser accepts the word *"cheapest."* This adds an Adj to the list of complete edges as well as a partially formed NP containing this initial Adj component. Each of these edges begin at position 0 of the utterance and end at position 1, since only the initial word has been processed. The bottom chart shows the state after the parser accepts the word *"flight."* This adds a N from position 1 to position 2 of the utterance. The addition of the N also completes the noun phrase rule, which spans position 0 to position 2. The newly created NP instantiates a new, incomplete edge consisting of the matched NP and a PP which it may match in the future (if the utterance were longer). Notice that the original edges from the top chart remain, allowing for alternate matches using those components.

edges for any rules that begin with the desired part of speech, placing those in the chart as well. The parser then attempts to combine each of these edges with others in order to advance or complete any existing edges that need that part of speech, as shown in Figure 2-7.

For example, when the word *"flight"* is being considered (after *"cheapest"* has already been considered and added to the chart), an edge for NP (added because *"cheapest"* is an adjective) can be completed, since the last component for NP → Adj N has been encountered. In addition, any rules beginning with N can be added as edges to the chart. The chart maintains separate lists of created edges (NP → Adj . N) and completed edges (NP → Adj N .).

Edges need not be restricted to contain just syntactic components and rules, however. One can imagine making edges more generalized, allowing them to contain

items other than rules. Indeed, in our implementation, edges simply contain a state, which allows the edge to contain words (items defined in the lexicon), rules, or other pieces of information, as described later in this thesis. This provides us with the flexibility we need to parse utterances without using syntax, while still utilizing the simplicity and power of the chart parsing mechanism.

Chapter 3

Semantic Parsing

Semantic parsing can effectively be summarized as parsing utterances based on meaning, only combining those words and phrases which are "meaningful" to combine. Instead of relying on a grammar to dictate which semantic functions should be applied, the grammar is eliminated altogether, and *all* semantic functions are tried, resulting in whatever combinations are allowed by the pre-defined semantic constraints. This chapter describes the exact mechanism that we employ to perform semantic parsing, as well as the component semantic functions and semantic constraints we define and use in this thesis. We also consider and discuss the semantics of sentence-level queries and statements. Finally, this chapter describes the domains we use for our semantic parsing experiments as well as the results of those experiments.

3.1 Mechanism

We can implement semantic parsing in a simple bottom-up chart parser, employing the same general mechanisms used for syntactic parsing. However, instead of determining whether or not rule-based edges can combine based on syntax, the system needs to make *semantic* edges which allow for edge combinations based on meaning. In other words, we modify edges so they contain semantics, not syntactic rules; this is fairly straightforward given the implementation described in the previous chapter. Next, when attempting to combine two edges, instead of checking if those edges can

combine to advance or complete a rule, the parser can attempt to combine them semantically, by trying all defined semantic functions with arguments of the semantics stored in the edges. Any combinations that result in valid semantics being computed by the semantic functions and constraints can be considered a successful combination, and the parser can add a new edge to the chart as normal.

In order to avoid the influence of syntax on semantic parsing, this method of parsing should be word order independent. When considering combining the semantics of two words or phrases, the order of those components should be ignored. Alternatively, one can be more efficient and approximate word order independence by maintaining adjacency constraints and trying to combine two edges two different ways: one with the semantics of the first edge as the first argument to a semantic function, and one with the semantics of the second edge as the first argument, as shown in Figure 3-1. In other words, the parser attempts combining two edges, A and B, by trying to compute $f(sem(A), sem(B))$ as well as $f(sem(B), sem(A))$, for all semantic functions f. This effectively allows the system to ignore word order within adjacent words and phrases.

Because no grammar is used to restrict which semantic functions are tried, the defined constraints become very important and central to semantic parsing. In fact, writing these constraints becomes another engineering task. However, producing these constraints manually tends to be more simple and straightforward than producing syntax rules manually. More significantly, in order to do any kind of language understanding, one has to write these constraints anyway; thus, this involves no extra work.

One can also make use of hierarchy and inheritance to simplify the process of writing these constraints. By declaring certain semantic concepts to be "children" of other concepts, any valid arguments of the ancestors of a word can be considered valid arguments of the word itself. For example, if "`flight`" is declared to be a child of "`object`," then any valid arguments of "`object`" are automatically valid arguments of "`flight`." This drastically reduces the number of constraints that need to be written.

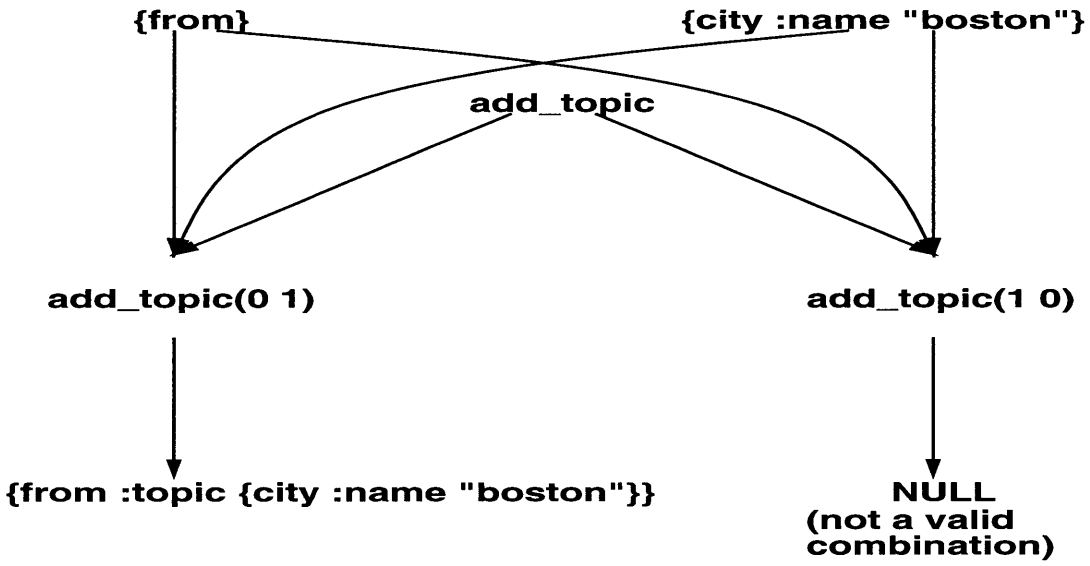


Figure 3-1: Semantic parsing ignores word order when combining two words or phrases.

When the semantic parser attempts to combine two words or phrases with semantic function f (for all f), it attempts to combine its inputs in both orders. For example, when trying to combine *"from"* and *"boston"* using **add_topic**, the system computes **add_topic (0 1)** as well as **add_topic (1 0)** for those inputs. Here, only one of these combinations results in valid output semantics.

3.2 Semantic Functions

Many different semantic functions can be used to create, represent, and combine the various concepts in a natural language system. Our system consists of functions primarily grouped into "phrasal" and "special" categories. The "phrasal" functions capture and help compute common concepts, such as the notion of certain words containing topics, agents, and other attributes. In addition, "special" functions span one or more domains and handle semantic concepts, such as dates and times, which are unique to those domains. Given the semantics of each of its inputs, each function verifies that it can combine its input semantics according to the defined constraints and subsequently attempts to create a single, new set of semantics based on their combination. If no meaningful combination is possible, the function accordingly returns "null" semantics.

The "phrasal" category of semantic functions consist of **add_topic**, **add_topic2**, **add_agent**, **add_marked**, **add_pred**, **add_name**, and **add_nn_rel**. We will now describe each of the major functions in detail.

add_topic takes two frames as input and attempts to create a new frame with the second input added as the "`topic`" of the first. **add_topic2** is related, using the "`topic2`" field, for capturing the notion of an indirect topic or object. Examples include *"boston"* serving as a topic for *"from,"* or *"me"* serving as the indirect object ("`topic2`") for *"show."*

add_agent adds its second input under the "`agent`" field of its first input. The utterance *"I want a flight"* provides a good example of an agent, with the word *"I"* serving as the agent of *"want."*

add_marked simplifies the format of its input somewhat, searching its second input for a "`topic`" field and its corresponding value. If it finds such a field, it adds the corresponding value directly to the first argument, using the name of the second

argument's head as the new key. This is useful for simplifying frame combinations for common concepts, such as that of a source and destination (*"from"*, *"to"*), with flights. For example, when **add_marked** combines {`flight`} with {`from :topic {city :name ''boston''}`}, it removes the city from the "`topic`" field and places it under the new "`from`" field of flight, resulting in the elegant meaning representation of {`flight :from {city :name boston}`}.

add_pred adds the semantics of its second input as a predicate ("functions" that can be applied to other words or concepts, such as a cost description being applied to a flight) of the first input. The relationship between words like *"cheapest"* and *"flight"* provides a good example of this notion.

add_name adds the second argument under the "`name`" field of the first; this is typically used for adding specific names for otherwise generic concepts. For example, we treat *"american airlines"* as an *"airline"* further modified by the name *"american."*

add_nn_rel handles combinations of objects, such as *"butter knife,"* where two words, which can serve as objects independently, can combine to form a new or different concept. It adds the second argument under the "`nnrel`" field of the first.

Of course, each of these combinations is restricted by the constraints which dictate whether or not one word can serve as a topic for another word, and so forth. Table 3.1 lists and summarizes all of the "phrasal" functions we used in our research.

The "special" functions compute less routine types of semantics. This category includes functions for making numbers, years, dates, and times. The **make_number** function takes two numbers and attempts to combine them into a larger number, semantically. The **make_year** function examines whether or not a combination of two numbers can form a reasonable number representing a year. **make_date** attempts to form a frame for the concept of a date in several different forms, by checking and combining numbers, years, days, and months. Similarly, **make_time** checks for

Semantic Function	Example Inputs	Output Semantics
add_topic	{a},{b}	{a :topic {b}}
add_topic2	{a},{b}	{a :topic2 {b}}
add_agent	{a},{b}	{a :agent {b}}
add_marked	{a},{b :topic {c}}	{a :b {c}}
add_pred	{a},{b}	{a :pred {b}}
add_name	{a},{b}	{a :name {b}}
add_nn_rel	{a},{b}	{a :nnrel {b}}

Table 3.1: "Phrasal" semantic functions.

This table lists all of the "phrasal" semantic functions we used in our experiments. These functions handle concepts of adding topics, agents, predicates, and so forth, to other words and phrases.

certain types of numbers, as well as a possible "a_m" or "p_m" modifier, and attempts to create semantics for a time. Table 3.2 lists all of these functions and gives examples of how they work.

3.3 Constraints

As mentioned, the semantic constraints represent a critical component of semantic parsing. These constraints provide restrictions on the output of semantic parsing, dictating which combinations are meaningful and preventing illogical meaning representations from being considered or created. The structure of the constraints are tightly coupled with the form of the semantic functions themselves, denoting whether or not a certain concept can serve as a topic or as an agent (for use by **add_topic** and **add_agent**, respectively) of another concept, and so forth. The constraints are specified in a file which lists each of the major concepts in a given domain. Under each of these concepts, the constraints list a set of valid arguments, specifying the type of the argument, as well as the possible keys under which the argument may be attached.

The constraints typically specify this through the `type` and `role` arguments. The `type` argument ensures that only arguments of the specified type (or children of the

Semantic Function	Example Inputs	Output Semantics
make_number	{num :value 30 :type ``cardinal''}, {num :value 1 :type ``cardinal''}	{num :value 31 :type ``cardinal''}
	{num :value 30 :type ``cardinal''}, {num :value 1 :type ``ordinal''}	{num :value 31 :type ``ordinal''}
make_year	{num :value 19 :type ``cardinal''}, {num :value 98 :type ``cardinal''}	{year :value 1998 }
make_date	{date :month ``may''}, {num :value 31 :type ``ordinal''}	{date :month ``may'' :monthday 31}
	{date :month ``may'' :monthday 31}, {year :value 1998 }	{date :month ``may'' :monthday 31 :year 1998}
make_clocknum	{num :value 11 :type ``cardinal''}, {num :value 30 :type ``cardinal''}	{clocknum :hour 11 :minute 30 }
make_time	{num :value 3 :type ``cardinal''}, ``o_clock''	{time :hour 3 }
	{clocknum :hour 11 :minute 30 }, ``a_m''	{time :hour 11 :minute 30 :merid ``a_m''}

Table 3.2: "Special" semantic functions.

This table lists all of the "special" semantic functions we used in our experiments. These functions handle creating the semantics for numbers, dates, and times.

specified type) can be bound under the appropriate key. Similarly, the `role` argument determines what key (and hence what semantic function) can be used to combine a pair of semantics. For example, when **add_topic** tries to combine a pair of semantics, it checks the constraints of the first argument to see if it has an entry with a `role` of "`topic`" and a `type` of the same type as the second argument. Likewise, **add_agent** checks its first argument's constraints for an entry with a `role` of "`agent.`" Most of the other types of functions and constraints follow a similar pattern, with a few exceptions. In particular, **add_marked** checks the constraints of its first argument for an entry with a `role` of the head of its second argument and a `type` corresponding to the semantics bound under the key of "`topic`" in the second argument. **add_pred** searches the constraints of the second argument, checking if it has an entry with a `role` of "`subj`" and a `type` corresponding to the first argument (checking if the second argument can serve as a predicate for a subject of the specific type).

In addition to listing these argument entries, the constraints can also include lists of nn-modifiers. These nn-modifiers refer to words describing objects which can modify a target object, forming a new object or concept. The nn-modifiers for a word declare which other semantic concepts can be combined to form an object-object pair, such as *"butter"* and *"knife"* combining into *"butter knife."* Accordingly, the function **add_nn_rel** checks its first argument's constraints to see if the second argument is listed as a possible nn-modifier. Finally, the constraints for a word can also list a possible parent of the word to allow for hierarchical lookup in determining the validity of predicates and arguments. Examples of constraints for selected words are shown in Figure 3-2.

Thus, the constraints for *"flight"* allow **add_marked** to place a location under the key of "`from`" or "`to`" and allow **add_nn_rel** to modify the flight with airline or class objects (i.e., *"american airlines flight," "first class flight"*). Similarly, the constraints for *"from"* allow **add_topic** to give it a topic of a location. Finally, the constraints for *"cost"* allow **add_pred** to apply it to a flight, and the constraints for *"desire"* allow **add_agent** to give it a speaker as its agent (the one who desires something, as compared to the item being desired).

```
{ flight
   :args ( { :type loc :role from }
           { :type loc :role to } )
   :nmods ( ``airline'', ``class'' )
   :parent ``showable_obj''
}
{ cost
   :args ( { :type flight :role subj } )
}
{ from
   :args ( { :type loc :role topic } )
}
{ show
   :args ( { :type showable_obj :role topic }
           { :type speaker :role topic2 } )
   :parent ``action''
}
{ desire
   :args ( { :type showable_obj :role topic }
           { :type action :role topic }
           { :type speaker :role agent } )
   :parent ``action''
}
{ quant
   :args ( { :role :type obj :role subj } )
}
{ showable_obj
   :parent ``obj''
}
{ city
   :parent ``loc''
}
{ cheap
   :parent ``cost''
}
```

Figure 3-2: Semantic constraints for selected words.
The basic constraints needed for handling utterances like *"cheapest flight from boston"* or *"show me all flights from boston."* The arguments that each word in those utterances can accept are listed here. Also shown are examples of inheritance; for example, "`flight`" is listed as having a parent of "`obj`," allowing any argument of "`obj`" to modify "`flight`."

Semantic Function	Example Inputs	Output Semantics
wh_ques	which flights from boston { WhObj :pred { quant :wh ``which''} }, { flight ...}	{ wh_ques :topic { WhObj ...} :comp { flight ...} }
make_command	please show flights from boston { please }, { show :topic { flight ...} }	{ show :topic { flight ...} }
make_statement	i want flights from boston { desire :agent { speaker :who ``me''} }, { flight ...}	{ statement :topic { flight ...} :pred { desire ...} }

Table 3.3: "Sentence-Level" semantic functions.

Sentence-level functions try to combine the semantics of their input words and phrases to determine if the overall utterance refers to a query, statement, or command. This table lists all of the sentence-level functions we used in our experiments.

3.4 Sentence-Level Semantics

Sentence-level functions represent another class of functions, independent of the "phrasal" and "special" ones. These functions capture "sentence-level" semantic concepts, such as commands, queries, and statements. An interpreter or evaluator then handles each of these with the appropriate type of response for the given sentence-level category. Table 3.3 displays all of the sentence-level functions we use and provides examples of how they work.

Specifically, the **wh_ques** function examines its inputs, checks if one consists of or contains a *"who," "what," "when," "where," "why,"* or *"how"* concept, and sees if the other is compatible or can be an argument, as defined by the constraints (typically, it checks to see if the other argument is a type of object). If these requirements are satisfied, the function creates a new set of semantics with the head "wh_ques,"

indicating the utterance refers to a query. Similarly, the **make_command** function checks its argument to see if it consists of or contains a type of "action" (i.e., the argument is a subclass of "`action`"), but does *not* contain an "`agent`" field, when creating its output semantics. Finally, the **make_statement** function checks its arguments to see if one is an "action" with an "`agent`" field and the other is a compatible argument of the action. If this is the case, then the function creates output semantics with the head of "`statement`," containing the topic and the action.

Often, the sentence-level functions can use the number and type of arguments to determine if its inputs compose a query, command, or statement. Indeed, the **make_command** function looks for a single argument of type "action" (without an "`agent`") to determine if its input is an utterance representing a command. However, relying solely on this criteria will not work in all cases; some functions take the same number and similar types of arguments, and so other criteria, such as the presence of specific concepts or arguments, must also be used. Further, it is impossible to distinguish most "yes/no question" utterances from a statement; the system needs contextual information to make this distinction. Because we do not supply this contextual information and actually are not even interested in the exact distinction, given the scope and nature of our research, we do not include a function that captures the notion of a "yes/no question"; it would simply duplicate the work and effort of the existing **make_statement** function.

These sentence-level functions need to be able to capture basic or similar meaning of different utterances. For example, a user might pose one of the three equivalent queries, *"show me flights from boston,"* *"please show me flights from boston,"* or *"could you show me flights from boston,"* to retrieve the same information. At the sentence-level, the parser often encounters words which contribute nothing to the semantics of a command, query, or statement. The word *"please"* in the command *"please show me flights from boston"* makes the command more polite but actually contributes nothing to the overall meaning representation extracted from that utterance. An interesting aspect of semantic parsing is being able to define these words as "filler words." More precisely, these words can be defined to be subclasses (children) of "`filler`," so that

they still have their own semantics but can also be considered filler words for other purposes.

The parsing system can therefore attempt to compute sentence-level semantics by skipping over the fillers and passing the non-filler semantics of an utterance to the sentence-level functions, which then decide into which sentence category, if any, that utterance falls.

3.5 Log File

In addition to recording the generated meaning representation as it parses utterances, this system also computes and logs other information that may be of use in later processing. Specifically, it logs the parts of speech of the words that combine as well as the semantic function that allows for their combination.

Thus, the semantic parse of *"cheapest flight from boston to philadelphia"* contains the information displayed in Figure 3-3. One can see that *"cheapest"* (an adjective) and *"flight"* (a noun) combine together by the **add_pred** semantic function, and *"from"* (a preposition) and *"boston"* (a name) combine together by the **add_topic** function. In turn, these two units get bracketed together by **add_marked**, and that unit combines with *"to philadelphia"* by the same function.

The information recorded in semantic parsing also illustrates how the system ignores filler words. For example, the parse of *"please show me flights from boston to philadelphia"* computes its final semantics by skipping over "please" and using the computed semantics of *"show me flights from boston to philadelphia"* as the sentence's overall semantics.

3.6 Implementation Details

We implemented the program used for semantic parsing (shown in Figure 3-4) in C++ for the WIN32 platform. In fact, we used the same program for syntactic parsing experiments; the parsing mode can be set by a menu option. We chose the WIN32

utterance:	cheapest flight from boston to philadelphia
meaning representation:	{flight :from { city :name "boston"} :to { city :name "philadelphia"} :pred { cheap :type "superlative"} }
parts of speech:	[add_marked (0 1): [add_marked (0 1): [add_pred (1 0): Adj N] [add_topic (0 1): P Name]] [add_topic (0 1): P Name]]
utterance:	which are flights from boston to philadelphia
meaning representation:	{wh_ques ...}
parts of speech:	[wh_ques (0 2): WhObj Cop [...]]
utterance:	please show me flights from boston to philadelphia
meaning representation:	{show ...}
parts of speech:	[make_command (1): Please [...]]

Figure 3-3: Sample log file entries.
Examples of entries in the log file generated by the semantic parser. The parser records the utterance, the computed meaning representation, and the parts of speech of the words that combine during parsing. The listed entries include examples of sentences and sentence-level computations.

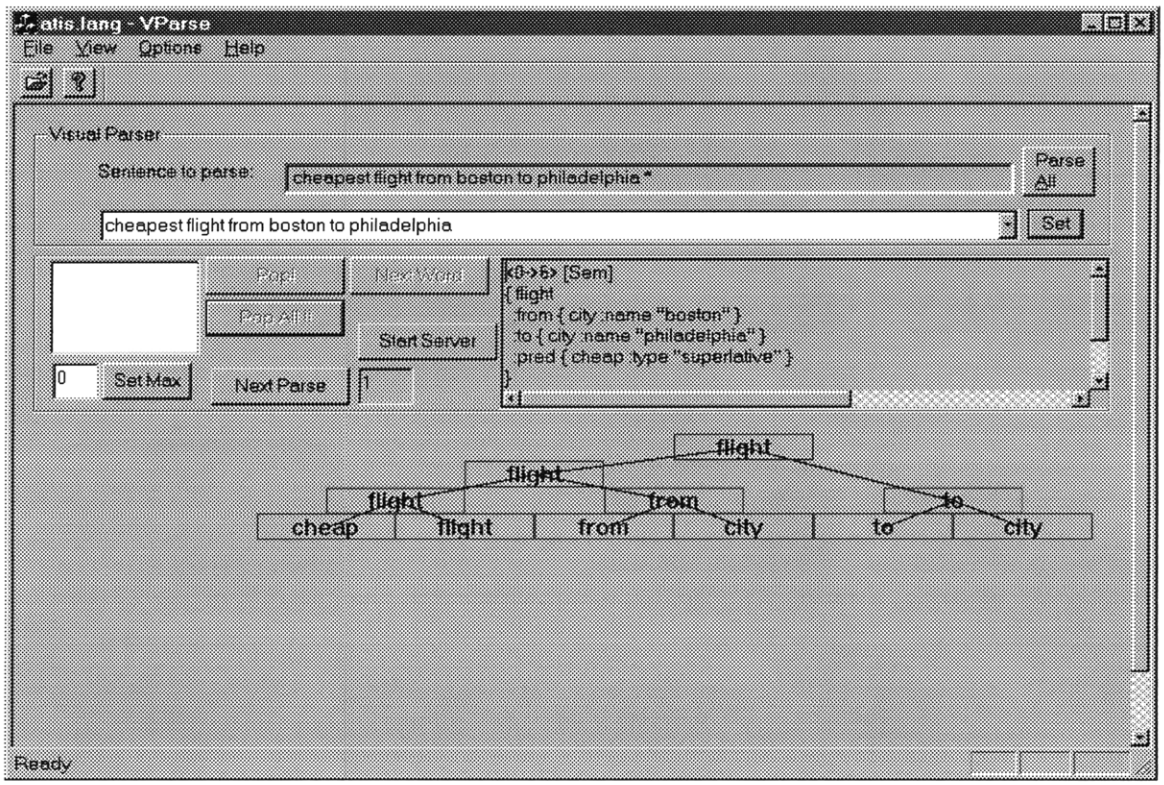


Figure 3-4: Screenshot of the semantic parsing system.

A screenshot of the actual program we use for our experiments. This program allows us to view the state of the parser, including the semantic parse tree and all of the computed meaning representations (if the system computes multiple parses). This program also serves as a syntactic parser and contains network server code, allowing for the remote execution of parsing experiments.

environment to allow us to create a GUI around the parser more easily, letting us view the parser state and step through parsing during the development stages of the research. To allow for other programs to control the parser remotely and interact across environments, we embedded a network server in the parser. This allowed for a user or a script to run parsing experiments remotely, get the results, and process them without having to interact through the GUI at all.

3.7 Domains and Experiments

The experiments in this thesis made use of two domains. Primarily, most of the testing and development utilized the ATIS [8] domain, a set of utterances involving airline travel queries and statements. In addition, some subsequent portability experiments utilized the Jupiter [15] domain, a set of utterances regarding information about the weather.

We divided these domains into different sets of utterances. From ATIS, we only used the "A" and "D" sets of utterances from the ATIS II and ATIS III collections. Speakers from these data were chosen at random, and all the utterances for a speaker were placed either into a training set, one of three development sets, or a test set. Out of a total of 7424 utterances, 3764 were placed into the ATIS TRAIN set and 1033 were placed in the ATIS TEST set; the remaining utterances were evenly divided into the ATIS DEV sets. From Jupiter, 1000 utterances were chosen at random and placed into a test set for the portability experiment. The grammar induction mechanism learned a grammar from the ATIS TRAIN set, refined the grammar on the ATIS DEV sets, and tested the final results on each of the TEST sets.

We defined a lexicon and set of semantic constraints for each of these domains and used them for our semantic parsing experiments. However, writing these constraints entailed defining an initial set of constraints, parsing a set of utterances, seeing what concepts were not handled, adding more constraints, reparsing the utterances, and so forth. As one can imagine, writing these constraints encompassed an engineering task in and of itself, so we chose to allocate some of our time and energy for subsequent

Utterance	Parse	Piece-count
A B C	[A B] C	2
A B C	[[A B] C]	1

Table 3.4: Illustration of piece-count measurement.
When the parser processes an utterance consisting of the words A, B, and C by combining just A and B, that utterance has a piece-count of two. If the parser also combines C with A and B, then the utterance is completely parsed and has a piece-count of one.

Set	Number of Utterances	Total Piece-count	Number of Complete Parses
ATIS TRAIN	3764	13515	1503
ATIS TEST	1033	3704	434
Jupiter TEST	1000	2436	636

Table 3.5: Overall semantic parsing results – coverage of sets.

The overall semantic parsing results for the TRAIN and TEST sets of ATIS as well as the TEST set of Jupiter. The experiments measured the piece-counts and coverage (number of complete parses) of the utterances in these sets.

experiments and portions of the thesis research, rather than just this engineering task. Therefore, the results of semantic parsing were by no means optimal. Rather, they served as a threshold and upper boundary against which results from later parsing experiments could be measured, as discussed later in this thesis. Overall, our ATIS lexicon and constraints contained 600 and 175 entries, respectively, including a list of 46 filler words. Similarly, our Jupiter lexicon and constraints contained 829 and 126 entries, respectively.

In analyzing the results of semantic parsing, we utilized and measured two primary criteria. First, we measured the coverage of the system, counting the number of utterances that were handled completely by the defined semantic constraints. Second, we quantified the fragmentation of the parses by counting the number of pieces in each utterance. As illustrated in Table 3.4, an utterance consisting of three words, A, B, and C, and generating a semantic parse in which just A and B combine, resulted in a piece-count of two for the parse; a complete parse had a piece-count of one.

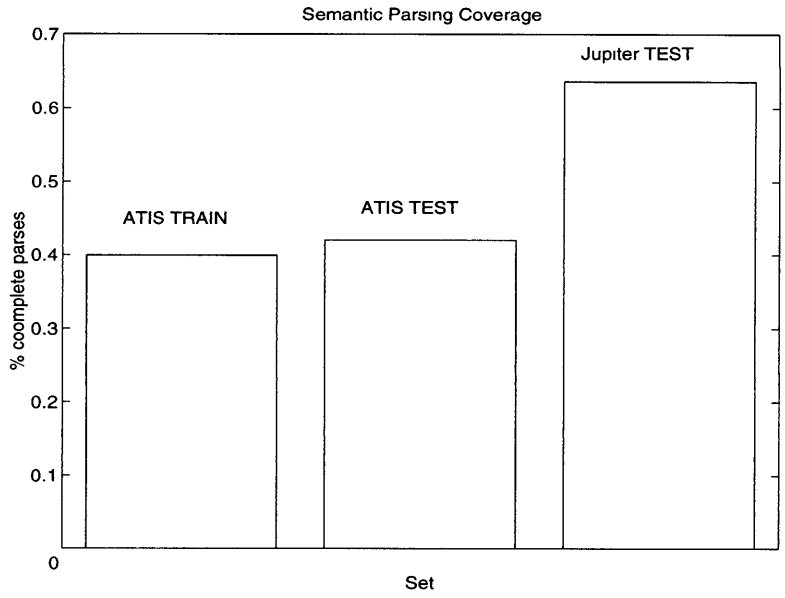


Figure 3-5: Overall semantic parsing results – coverage of sets.
The overall coverage of semantic parsing on the three sets. Semantic parsing covered about 40% of each of the ATIS sets and 64% of the Jupiter set.

Table 3.5 and Figures 3-5 and 3-6 display the results of our semantic parsing experiments as measured by the mentioned criteria. Semantic parsing covers about 40% of the utterances in two separate sets from the ATIS domain and 64% percent of the utterances in a set from the Jupiter domain. Utterances in Jupiter tend to be shorter than ones in ATIS; this observation likely represents the biggest contribution to this difference in coverage between domains. Finally, each of the parsed utterances has a low piece-count on average, and hence the total piece-count is also reasonably low. Indeed, a histogram of the piece-count measurement shows that the majority of utterances have a piece-count towards the low end of the spectrum, signifying an overall small amount of fragmentation, or alternatively close to complete coverage of the sets.

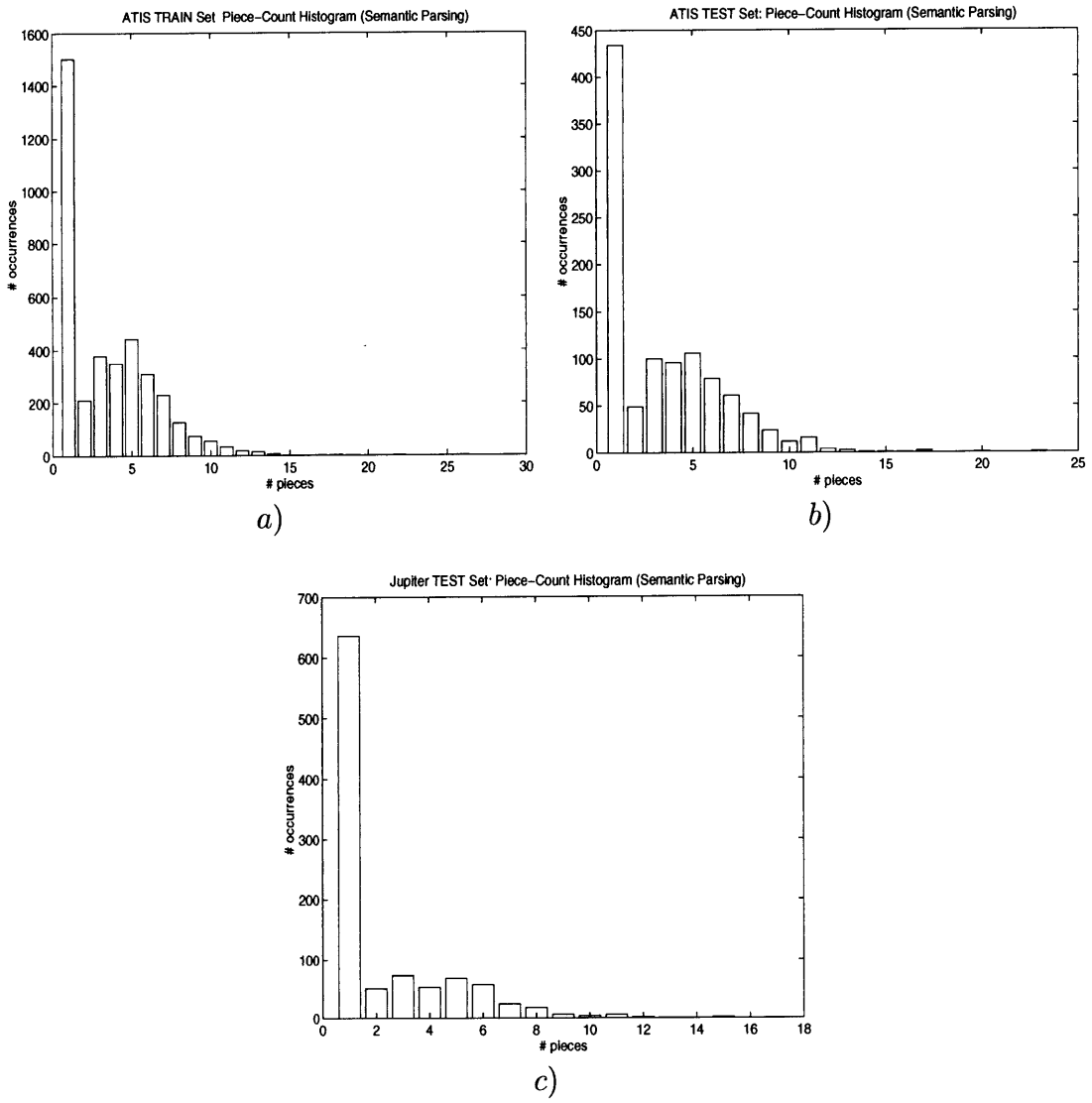


Figure 3-6: Overall semantic parsing results – histogram of piece-counts. These graphs display histograms of the piece-counts for utterances in the a) ATIS TRAIN, b) ATIS TEST, and c) Jupiter TEST sets. As shown, the piece-count for the majority of these utterances is relatively low.

Chapter 4

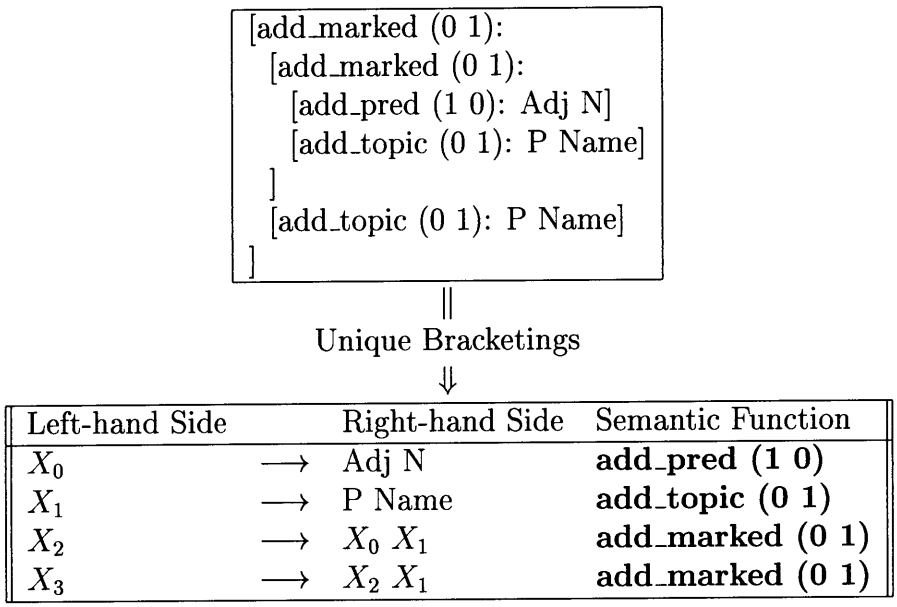
Grammar Induction

The inputs to semantic parsing effectively define *what* words and phrases can combine. Grammar induction mechanisms can use the output of semantic parsing and observe *how* things actually combine, generating the corresponding grammar. We explore two relatively simple techniques that interface with semantic parsing to generate a grammar. The first technique generates a new rule from every unique bracketing seen in the logs of the semantic parse of a training set and subsequently clusters these syntactic units together based on co-occurrence. The second technique attempts to make more use of the semantic phrasal information available in the logs in order to improve the efficiency of the mechanism and to make the learned grammar more compact. We now discuss each of these techniques in detail.

4.1 Unique Bracketings

4.1.1 Rule Extraction Phase

In this approach, the system generates a grammar by extracting one rule for every unique bracketing encountered in the semantic parse logs; Figure 4-1 illustrates this with an example. In the semantic parse shown (for *"cheapest flight from boston to philadelphia"*), a rule would be extracted for the bracketing of Adj and N with semantic function **add_pred (1 0)** and given a fresh non-terminal X_0. Similarly, the



Left-hand Side		Right-hand Side	Semantic Function
X_0	$\longrightarrow$	Adj N	**add_pred (1 0)**
X_1	$\longrightarrow$	P Name	**add_topic (0 1)**
X_2	$\longrightarrow$	X_0 X_1	**add_marked (0 1)**
X_3	$\longrightarrow$	X_2 X_1	**add_marked (0 1)**

Figure 4-1: Rules learned from unique bracketings in *"cheapest flight from boston to philadelphia."*
This figure shows the semantic parse of *"cheapest flight from boston to philadelphia"* and the resulting syntactic rules learned via the unique bracketings approach. From the Adj N combination, the system extracts the rule $X_0 \rightarrow$ Adj N. Similarly, it extracts the X_1 rule for the P Name combination and the X_2 rule for the combination of X_0 and X_1. Finally, the system creates an X_3 rule for the X_2 X_1 combination.

Left-hand Side		Right-hand Side	Semantic Function
X_0	$\longrightarrow$	Adj N	**add_pred (1 0)**
X_1	$\longrightarrow$	P Name	**add_topic (0 1)**
X_0	$\longrightarrow$	X_0 X_1	**add_marked (0 1)**

Figure 4-2: Example of merged rules.
The system merges X_0 and X_2 (from Figure 4-1) by replacing every instance of X_2 with X_0 and eliminating duplicate rules.

bracketing of P and Name would be extracted as another rule, X_1, and the combination of those two syntactic units would be recognized as another unique bracketing and extracted into another rule, X_2. Another instance of a P Name bracketing (*"to philadelphia"*) is encountered, but this is not unique, so no new rule needs to be generated for this observation. However, that instance of X_1 combines with X_2, and thus a new rule X_3 is extracted, completing the set of grammar rules that cover this utterance. This process is continued for every utterance in the semantic parse logs.

4.1.2 Merging Phase

As one can imagine, extracting a new rule for every unique bracketing seen can result in a very large grammar for all but the most trivial set of utterances. Furthermore, this grammar is highly specific to the training set. Therefore, to make the grammar more compact as well as more generalized, this system needs to perform clustering in order to merge the extracted syntactic units and rules.

Clustering techniques rely on distance metrics for choosing which units to combine at each iteration. However, as mentioned, developing this distance metric can easily grow into an independent research project; therefore, our research utilizes a fairly simple metric. Based on co-occurrence, our metric ranks every pair of possible merges by their co-occurrence counts. Co-occurrence describes when two syntactic units exist in the same context; in the previous example, X_0 and X_2 have a co-occurrence of one because they occur to the left of unit X_1 with the same semantic function **add_topic (0 1)**.

Each merge involves replacing every instance of one token with the companion token with which it combines. For example, if we choose to merge X_0 and X_2, we would derive the grammar shown in Figure 4-2. However, instead of simply merging the pair of tokens with the highest amount of co-occurrence, this system actually ranks all pairs of tokens by this measurement and tries the top n merges in syntactic parses of a development set. It then chooses the merge which results in the maximal reduction of the piece-count as the optimal merge for that iteration of the induction process. Essentially, this represents a greedy version of trying *all* possible merges in syntactic parses by restricting the parsing experiments to those which are likely to reduce the piece-count significantly (merging those pairs of units which display similarity because of co-occurrence).

Therefore, this metric does not blindly rely on co-occurrence statistics alone because there is no guarantee that co-occurrence can be used in creating meaningful and useful generalizations of a grammar. Instead, it tries to find the optimal merge based on which one reduces the piece-count the most. Since the piece-count measures fragmentation and captures how complete a set of rules are in terms of coverage, this measurement is meaningful and quite reasonable.

Figure 4-3 displays an overview of the induction process. First, the system performs a semantic parse of a training set of utterances. It extracts a syntactic grammar G_0 from this parse based on the unique bracketings approach. Next, it tries this grammar in a syntactic parse of a development set of utterances, counting the number of pieces in the resulting parses. After ranking all pairs of units by their co-occurrence counts in G_0, the system tries the top n merges in independent syntactic parsing experiments using the development set. The system chooses the merge which reduces the piece-count the most, along with the resulting grammar G_1, as the output of that iteration.

The next iteration begins by ranking all pairs of units in G_1 by their co-occurrence, trying the top n merges in syntactic parses, and again choosing the merge which reduces the piece-count the most to produce G_2. This process is repeated until the piece-count converges. We can then use the final output grammar in parsing experi-

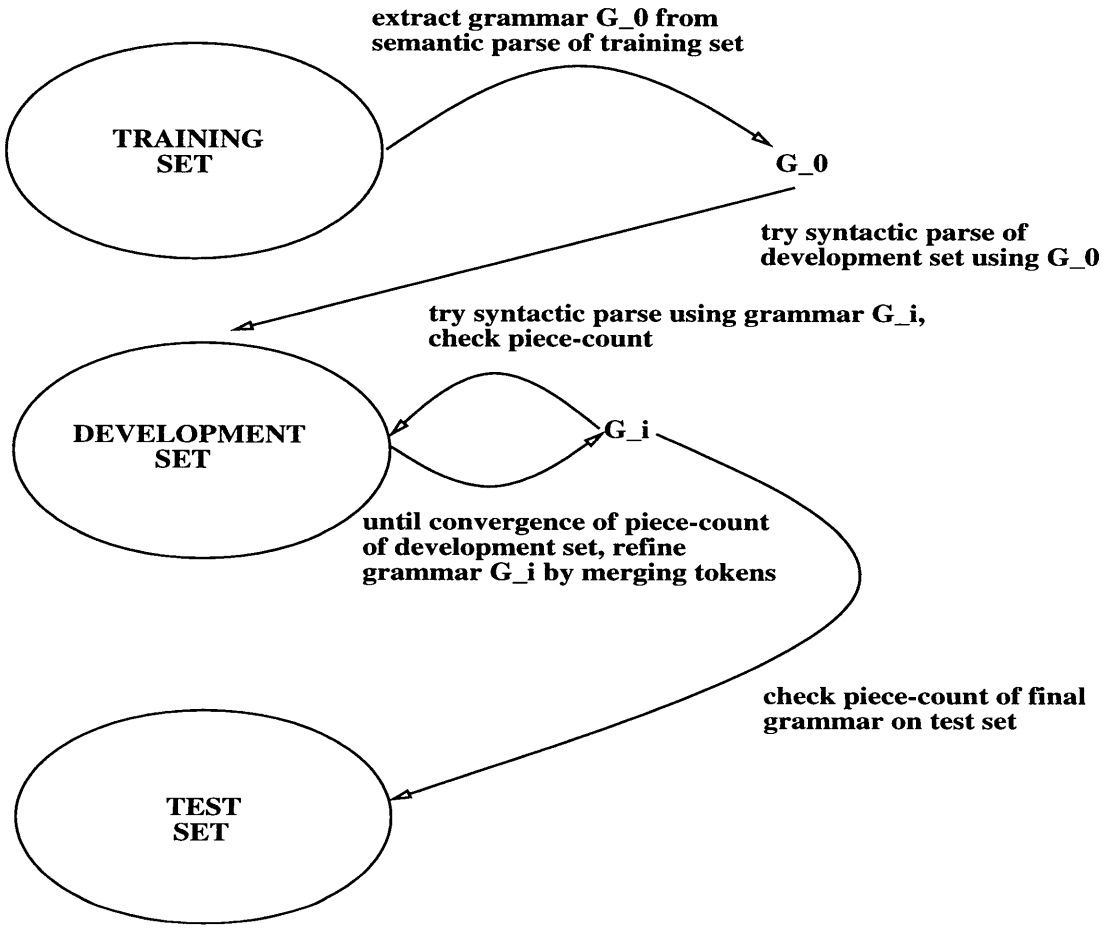


Figure 4-3: Overview of the induction process for the unique bracketings approach. This figure provides an overview for the induction process for unique bracketings. After the system semantically parses a training set, it extracts a grammar G_0 for that set using unique bracketings. It then tries using that grammar to perform a syntactic parse of a development set and obtains the corresponding piece-count. Next, the system attempts to create a grammar G_i by attempting n independent token merges of G_{i-1} and using the resulting grammars in syntactic parsing experiments, choosing the merge which reduces the piece-count the most. After each iteration, the system attempts to create a grammar G_{i+1}, repeating the entire process until the piece-count of the syntactic parse of the development set converges. The induction process concludes with a syntactic parsing experiment of a test set using the final grammar.

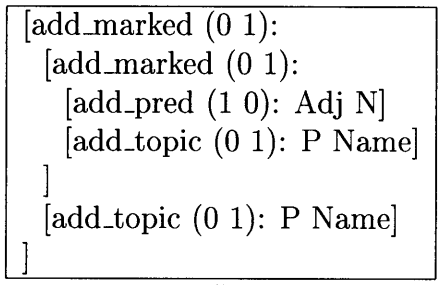
ments on a separate test set of utterances.

4.2 Semantic-Head Driven Induction

One potential flaw in the previous approach is that it does not make much use of the semantic-level information present in the semantic parse logs. The unique bracketings approach uses just the bracketings provided by semantic parsing, essentially ignoring the semantic phrasal information also embedded in the logs. Semantic-head driven induction (SHDI) represents another approach to the grammar induction process that does make use of this information. Essentially, SHDI builds on top of the unique bracketings approach and improves it by making more use of the information available.

Because of the nature of language, and accordingly the way we define semantic functions, the first argument to a function constitutes the head, or major concept, of the resulting semantic frame. For example, when *"cheapest"* and *"flight"* combine, *"flight"* is the first argument to **add_pred**, and the resulting semantic frame retains *"flight"* as its head. This can provide very useful information for grammar induction. Essentially, this observation recognizes how semantic-level phrases are constructed. One can therefore use these semantic-level phrases to influence the learning of syntactic structure. By using the part of speech of the first argument to a semantic function, the extraction mechanism can create syntactic phrases based on semantics and generate clean, readable rules directly from the parse logs, rather than assigning arbitrary non-terminal labels. Further, the extracted rules are often recursive in nature, eliminating the need for a subsequent merging phase; essentially, the learned grammar is pre-merged.

The example in Figure 4-4 illustrates this mechanism. Given the semantic parse for *"cheapest flight from boston to philadelphia,"* SHDI begins by recognizing the noun *"flight"* as the first argument to **add_pred** and extracts a syntactic rule N_0 for the Adj N combination. Similarly, it recognizes the preposition *"from"* as the head in the combination of *"from"* with *"boston"* and creates the P_0 rule and phrase type for this combination. Finally, upon encountering the combination of those two phrases, the



Semantic-Head Driven Induction

Left-hand Side		Right-hand Side	Semantic Function
N_0	$\longrightarrow$	Adj N	add_pred (1 0)
P_0	$\longrightarrow$	P Name	add_topic (0 1)
N_0	$\longrightarrow$	N_0 P_0	add_marked(0 1)

Figure 4-4: Rules learned in semantic-head driven induction from *"cheapest flight from boston to philadelphia."*
This figure shows the semantic parse of *"cheapest flight from boston to philadelphia"* and the resulting syntactic rules learned from the SHDI approach. From the Adj N combination, the system extracts the rule N_0 $\rightarrow$ Adj N. Similarly, it extracts the P_0 rule for the P Name combination and another N_0 rule for the combination of N_0 and P_0. Notice that the last rule is recursive, eliminating the need for merging.

technique observes that *"cheapest flight"* is the first argument to **add_marked** and therefore creates another instance of the N_0 (the derived part of speech for *"cheapest flight"*) rule corresponding to the N_0 P_0 combination. Interestingly enough, this last rule is extracted as two rules under the unique bracketings approach and represents a candidate for merging; SHDI learns the merged version directly.

4.3 Implementation Details

Because much of this grammar induction work involves parsing and scanning through logs from the semantic parsing stage, we utilize Perl to perform this task. The grammar induction code actually consists of a set of Perl scripts which read the semantic parse logs and record which pairs of tokens are bracketed together and the function that allows for their combination. These scripts subsequently generate a grammar containing some left-hand side token (according to the grammar induction technique used), the tokens for the right-hand side, the corresponding semantic function, and the count of how many times each rule (bracketing) is encountered in the logs. The scripts run on a DIGITAL UNIX system.

As previously mentioned, the first approach to induction requires tight coupling and interaction between the grammar induction scripts and the syntactic parser, since the induction process depends on the results of syntactic parsing experiments. To allow for this integration, this system utilizes network communication. The parser contains an embedded server, and the scripts contain client code and subroutines, allowing for communication with the parser server. In this manner, the scripts can dictate which utterances and grammar the parser should use for an experiment. As one script extracts a grammar and attempts a merge, it can obtain the piece-count for the grammar by invoking other scripts. One of these scripts directs the parser to run a syntactic parsing experiment, and another one computes the results (i.e., measures the piece-count for the parse using that grammar). Together, these scripts allow the induction mechanism to pick a grammar G_i for each stage of the induction.

The second approach, semantic-head driven induction, utilizes a set of scripts

which are heavily based upon the original induction scripts. However, the newer scripts do not use or need to use the network client code, since SHDI neither needs to run any syntactic parsing experiments nor needs to undergo a merging phase. Instead, these scripts simply use the semantic phrasal information to create or choose the appropriate left-hand side token.

For both types of induction, we want to avoid letting the order in which utterances appear in a set (typically alphabetical) influence the training process or misguide us in our interpretation of how many rules are being learned as more and more data are processed. To avoid this influence, the induction scripts randomize the order of the utterances in the semantic parse logs before processing them. In that manner, the scripts will not encounter all the utterances that start similarly (for example, *"what are the cheapest flights..."*) at once, but rather process them in a random order. Because of this randomization, the induction process is non-deterministic. Different trials result in the induction of different rules (and number of rules). However, these differences are slight; the fluctuation in the number of rules between trials is usually on the order of five rules or fewer.

4.4 Results of Induction

Using unique bracketings, our system generates an unmerged grammar containing 3785 rules learned from the ATIS TRAIN semantic parse; Figure 4-5 displays some examples of these rules. As this figure illustrates, several different rules correspond to similar phrase types; for example, X_0, X_{15}, X_{89}, X_{90}, X_{99}, X_{146}, X_{357}, and X_{358} all involve different types of noun phrases, and accordingly, several other groups of rules involve each of these tokens. These tokens exemplify prime candidates for merging.

Unfortunately, the mechanism for choosing merges proves to be far too slow to be usable. Each *iteration* requires n syntactic parsing experiments, and each of these syntactic parsing experiments takes an unreasonable amount of time to complete, especially given a grammar of this size. Therefore, we must rely on semantic-head driven induction (and its ability to produce a pre-merged grammar) as our grammar

Left-hand Side		Right-hand Side	Semantic Function	Count
X_0	$\longrightarrow$	Det N	**add_pred (1 0)**	1325
X_{15}	$\longrightarrow$	Adj N	**add_pred (1 0)**	276
X_{99}	$\longrightarrow$	N N	**add_nn_rel (1 0)**	121
X_3	$\longrightarrow$	P Name	**add_topic (0 1)**	4325
X_{89}	$\longrightarrow$	X_0 X_3	**add_marked (0 1)**	298
X_{146}	$\longrightarrow$	X_{99} X_3	**add_marked (0 1)**	23
X_{357}	$\longrightarrow$	X_{15} X_3	**add_marked (0 1)**	36
X_{90}	$\longrightarrow$	X_{89} X_3	**add_marked (0 1)**	265
X_{358}	$\longrightarrow$	X_{357} X_3	**add_marked (0 1)**	33
X_{2370}	$\longrightarrow$	Aux Pro X_{167}	**make_command (2)**	3
X_{2371}	$\longrightarrow$	Aux Pro X_{171}	**make_command (2)**	1
X_{2464}	$\longrightarrow$	Please X_{1028}	**make_command (1)**	1
X_{2465}	$\longrightarrow$	Please X_{1072}	**make_command (1)**	1
X_{2672}	$\longrightarrow$	WhObj Cop X_0	**wh_ques (0 2)**	12
X_{2773}	$\longrightarrow$	WhObj Cop X_{90}	**wh_ques (0 2)**	20

Figure 4-5: Examples of (unmerged) rules learned in unique bracketings.
This figure shows examples of rules learned in the unique bracketings approach. These rules are unmerged; hence, several rules correspond to similar syntactic concepts. For example, rules like X_0, X_{15}, X_{89}, and X_{99} all refer to noun phrases. Several other rules refer to each of these tokens, resulting in the learning of duplicate versions of similar rules.

induction mechanism.

Semantic-head driven induction has a number of advantages over the unique bracketings approach. First, because there is no need for merging, using the scripts and performing the actual grammar induction based on this approach is much quicker. Second, SHDI produces a very compact grammar which is faster to use because the syntactic parsing system can consider fewer rules when trying to parse an utterance. Third, because the actual parts of speech of semantic phrases are used in labeling the learned syntactic structure, the generated rules are actually quite readable, making examining and reasoning about them much more straightforward than in the original approach.

Figure 4-6 provides an interesting view of the learning process, plotting the number of utterances against the number of rules learned. As shown in this figure, the graph approaches an asymptote. Essentially, as the system parses more and more utterances,

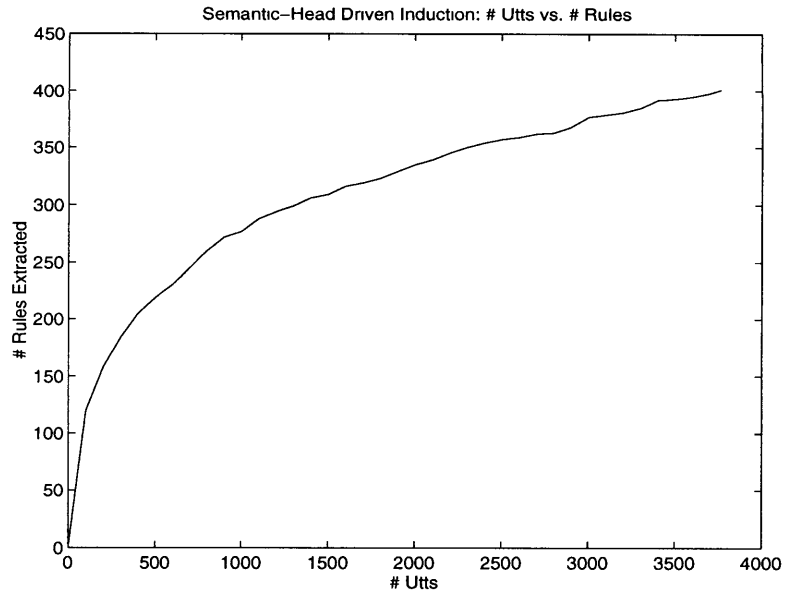


Figure 4-6: Number of utterances vs. number of rules learned in semantic-head driven induction.
This graph shows the number of utterances plotted against the number of rules learned in our SHDI experiments. As the system encounters more utterances, it learns and extracts fewer new rules, approaching an asymptote.

it encounters fewer previously unseen formations, and therefore does not need to create as many new rules.

Using SHDI, the system extracts 401 grammar rules. Some examples of rules actually learned by this process are shown in Figure 4-7. These rules are much more readable than the ones learned under the unique bracketings approach, and a single recursive rule can cover the same constructs that correspond to many distinct rules in an unmerged grammar.

Qualitatively, many of the rules learned under SHDI are quite reasonable, such as N_0 → N_0 P_0, or N_0 → N N. Other rules are a little more puzzling, such as P_0 → Name P in *addition* to the expected P_0 → P Name. Incomplete phrases and parses heavily contribute to this phenomenon. For example, in *"boston to philadelphia,"* both *"boston"* and *"to"* as well as *"philadelphia"* and *"to"* can combine. The system does not have enough information available to complete the parse and resolve which combination is correct, and thus the grammar induction script can encounter the *"boston to"* combination and generate a N_0 → Name P rule.

Left-hand Side		Right-hand Side	Semantic Function	Count
N_0	$\longrightarrow$	N_0 P_0	**add_marked (0 1)**	2400
N_0	$\longrightarrow$	N N	**add_nn_rel (1 0)**	150
N_0	$\longrightarrow$	Number_0 N	**add_pred (1 0)**	12
P_0	$\longrightarrow$	Name P	**add_topic (1 0)**	111
P_0	$\longrightarrow$	P Name	**add_topic (0 1)**	4241
Number_0	$\longrightarrow$	Number Number	**make_number (0 1)**	137
Number_0	$\longrightarrow$	Number Number	**make_number (1 0)**	132
SENT_0	$\longrightarrow$	Aux Pro V_0	**make_command (2)**	23
SENT_0	$\longrightarrow$	Please V_0	**make_command (1)**	98
SENT_0	$\longrightarrow$	WhObj Cop N_0	**wh_ques (0 2)**	223

Figure 4-7: Examples of rules learned in semantic-head driven induction.
Several of the rules our system learned using SHDI. Most rules are fairly readable and reasonable. Other rules are more confusing, such as P_0 $\rightarrow$ Name P; however, the system records how many times each rule appears in the data, demonstrating that this rule is not very common. Other rules, such as ones for handling numbers, are not as reasonable or dismissable from count information. Some sentence-level rules are also shown.

However, in addition to recording the rules that they learn and encounter, the grammar induction scripts also record counts of how many times a rule occurs in the training data. In examining these counts, one can see that the expected P Name combination clearly dominates the Name P combination. Thus, a statistical parser could assign that rule a lower probability, or just completely ignore it, and recover from the existence of this otherwise confusing pair of rules.

On the other hand, some constructs do give the semantically-based induction system some less avoidable or less recoverable problems. Specifically, it is difficult to distinguish numbers based solely on semantics, while ignoring syntactic cues such as order. For example, because the components of the number *"fifty two"* are considered in both orders (fifty two, two fifty), the system cannot ascertain which number is meant. Here, syntactic cues or pre-labeled examples are necessary, and thus numbers represent an example of where semantic-based grammar induction can suffer.

Finally, the system can also extract sentence-level syntactic rules. As shown, the rules for phrases like *"can you show me flights...,"* *"please show me flights...,"* and *"what are flights..."* are among those learned.

Chapter 5

Parsing and Portability

The next stage in the research assesses how well the learned grammar performs in syntactic parsing experiments. These experiments involve using the grammar in both the original domain in which it was learned (ATIS) as well as a new domain (Jupiter). This chapter describes the mechanisms used to conduct those experiments, the experiments themselves, and the overall results.

5.1 Modifications to the Syntactic Parser

We utilized an all-parses bottom-up chart parser for our syntactic parsing experiments. However, even using the grammar produced by semantic-head driven induction, the system performed too slowly for us to run (and re-run) all of our desired experiments in a reasonable amount of time. To address these concerns about speed and efficiency, we made some modifications and enhancements to the parser.

First, we added semantic filtering to the parsing mechanism. Instead of performing all semantic computation after completing a parse, the chart parser computed semantics of edges *as* it created them, and immediately filtered out any semantically invalid edges. Thus, instead of keeping track of edges which cannot possibly contribute to a semantically valid parse (because the edge itself had no meaning), the system did not even consider or maintain that edge. Alternatively, one could think of this enhancement as restricting our experiments to those utterances which

were semantically valid (and were handled completely by the defined semantics and constraints).

Another enhancement made to the system involved relaxing the definition of "equals" for two edges. Two syntactic edges often had slightly different structure but contained the same computed semantics, as shown in the *"cheapest flight from boston"* example in Figure 5-1. We therefore relaxed the notion of equals so that two edges with the same semantics were considered equivalent, based on the justification that an interpreter would perceive no difference in this set of potentially ambiguous edges and parses (and therefore maintaining a difference was unnecessary and even wasteful).

In addition, sometimes two edges would contain similar semantics with only slight differences, such as the locations where embedded arguments might be bound, as shown in the *"from boston fare to philadelphia"* example in Figure 5-1. Therefore, we also relaxed the notion of equals for semantics themselves, so the contents, not the locations of where arguments were bound, were considered. Again, we felt this was perfectly reasonable, as the two alternative meaning representations would evaluate to the same thing but slowed down the syntactic parsing, since the parser had to maintain and consider essentially duplicate edges in the chart data structure.

With these enhancements, the syntactic parser proved to be fast enough to run all of the desired experiments in a reasonable amount of time. Of course, because of semantic filtering, the syntactic parser could do no better than the semantic parser. If a phrase did not have valid semantics, the syntactic parser filtered it out, removing it from consideration. Thus, the results of semantic parsing served as an upper bound against which we could evaluate the results of our syntactic parsing experiments.

5.2 Parsing Experiments

The first set of parsing experiments involved assessing how well the learned grammar performed in the original domain. After doing a semantic parse of the training set of utterances from the ATIS domain and inducing the corresponding grammar, we

syntactic parse	meaning representation
edge: cheapest flight from boston	
[[Adj N] [P Name]]	{flight :from { city :name "boston"} :pred { cheap :type "superlative"} }
[Adj [N [P Name]]]	{flight :from { city :name "boston"} :pred { cheap :type "superlative"} }
edge: from boston fare to philadelphia	
[[[P Name] N] [P Name]]	{fare :from { city :name "boston"} :to { city :name "philadelphia"} }
[[P Name] [N [P Name]]]	{fare :to { city :name "philadelphia"} :from { city :name "boston"} }

Figure 5-1: Examples of edges with identical or similar semantics.
Illustration of how the same edge might be parsed different ways, resulting in different but similar (or even identical) meaning representations. Because an adjective can combine with either a noun or a noun phrase, several syntactic parses of *"cheapest flight from boston"* are possible; semantically, these parses are identical. Similarly, *"from boston fare to philadelphia"* can be parsed two different ways. However, the meaning representations are nearly identical, differing only in the location of where the "`from`" and "`to`" keys (and corresponding values) are bound.

tried the grammar on two sets of utterances from ATIS. First, as an initial check, we ran a syntactic parsing experiment in the original set of utterances in which the grammar was learned (ATIS TRAIN) to ensure we got the same results as before. In fact, we observed slightly improved results! The reason for this improvement related to sentence-level rules. Because sentence-level functions required the pre-definition of all filler words which it should ignore, a certain instance where a non-filler word actually contributed nothing to an utterance could not be handled semantically; the system could not combine the non-filler word with the rest of the utterance. However, a syntactic sentence-level rule existed where the non-filler slot was ignored (because that slot was typically occupied by a filler word), allowing the non-filler word to be treated as a filler in order to compute the overall semantics for the utterance. Thus, one extra utterance was completely covered that had not been covered in the semantic parsing stage.

Also, because sentence-level semantic functions were only applied over an entire parse, sentence fragments were never produced in semantic parsing, although an utterance might contain several embedded fragments or sentences. Their corresponding sentence-level syntactic counterparts, however, could be applied anywhere in the parse, since the parser treated them like any other grammar rule. Therefore, the syntactic parser handled sentence fragments and embedded sentences much more easily than the semantic parser, resulting in a lower piece-count for the syntactic parser. Overall, in the ATIS TRAIN domain, the syntactic coverage and piece-count results were very similar to the semantic coverage and piece-count, as expected.

Set	Number of Utterances	Total Piece-count	Number of Complete parses
ATIS TRAIN	3764	12342	1504
ATIS TEST	1033	3428	420

Table 5.1: ATIS syntactic parsing results – coverage of sets.

The overall syntactic parsing results for the ATIS TRAIN and TEST sets. The experiments measured the piece-counts and coverage (total number of complete parses) of the utterances in these sets.

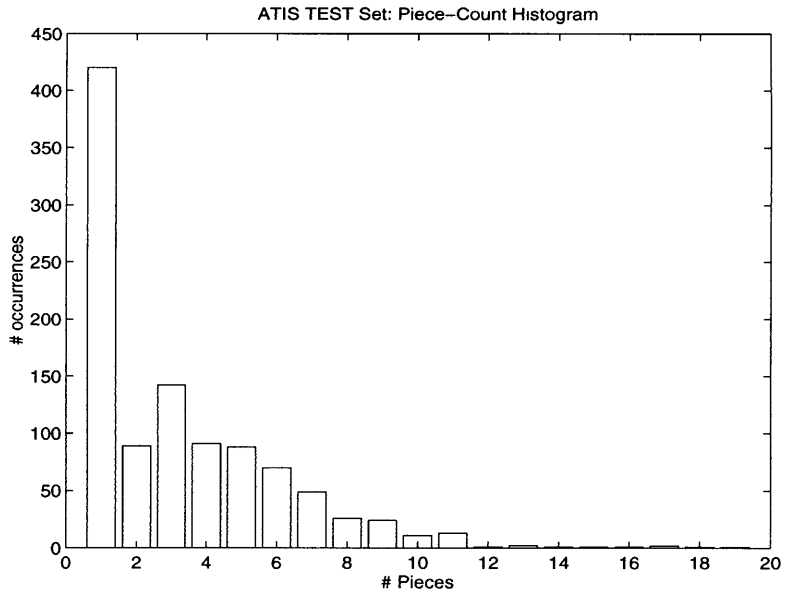


Figure 5-2: ATIS syntactic parsing results – histogram of piece-counts.
This graph shows the histogram of piece-counts for the utterances in the ATIS TEST set. As desired, the majority of utterances have a low piece-count.

Next, we ran the ATIS TRAIN-learned grammar on a separate set of test utterances from ATIS and achieved very promising results. The induced grammar covered 98% of the utterances that were covered semantically (or 41% overall), and this grammar reduced the piece-count for reasons similar to the ones already described. Thus, the grammar performed extremely well in handling utterances from the same domain in which it was learned. Table 5.1 shows the overall parsing results; in addition, Figure 5-2 shows a histogram of the piece-counts for the ATIS TEST set. The majority of the distribution was towards the low end of the spectrum, signifying that most utterances consisted of a small number of pieces and were close to being completely covered.

5.3 Portability Experiment

The final parsing experiment involved assessing how well the ATIS-trained grammar performed in a different *domain* altogether. We chose the Jupiter domain for this portability experiment. Jupiter, a weather information domain, primarily consisted of weather-related queries, such as *"what is the weather forecast for boston."* For this

Set	Number of Utterances	Total piece-count	Number of Complete Parses
Jupiter TEST	1000	2529	541

Table 5.2: Jupiter syntactic parsing results – coverage of set.

The overall syntactic parsing results for the TEST set of Jupiter. The experiments measured the piece-counts and coverage of this set.

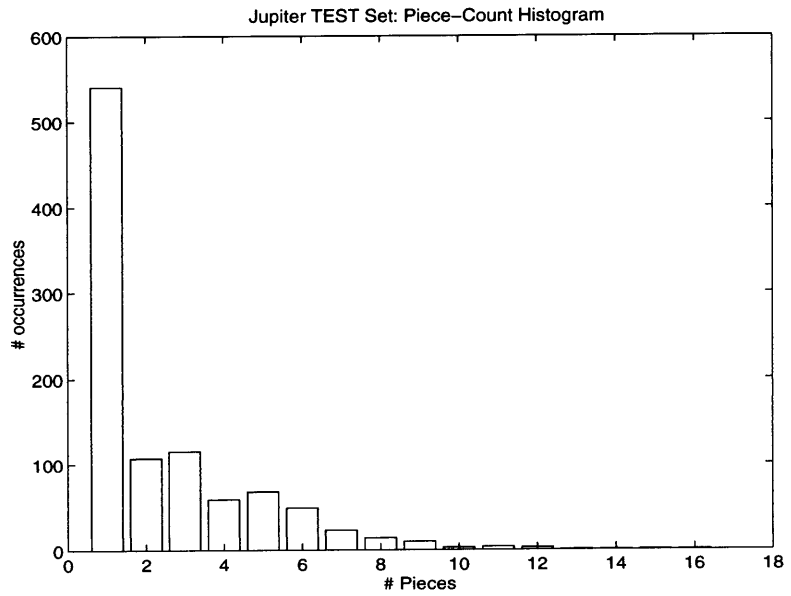


Figure 5-3: Jupiter syntactic parsing results – histogram of piece-counts.
This graph shows the histogram of piece-counts for the utterances in the Jupiter TEST set. Again, the majority of utterances have a low piece-count.

experiment, we defined a lexicon and set of semantics for Jupiter and attempted to parse a set of utterances using the grammar learned in the ATIS TRAIN domain.

As shown in Table 5.2 and Figure 5-3, the grammar performs reasonably well. It covers 85% of the semantically valid utterances (54% overall), so performance degrades somewhat compared to the in-domain parsing experiment. We have theories to explain this degradation in performance, however, and will address this issue shortly. Again, the piece-count histogram illustrates that the majority of utterances have a low piece-count, as desired.

5.4 Examples of Parses

Figures 5-4 and 5-5 provide some examples of utterances from the test sets of both the ATIS domain as well as the Jupiter domain. The induced grammar and corresponding semantic computation (from the syntactic parse) cover many utterances completely, resulting in useful semantic frame representations for those utterances.

However, even if the grammar cannot cover an entire utterance, all is not necessarily lost. The system computes and records the semantics for each partially parsed fragment of an utterance; one or more fragments may contain enough information to allow an interpreter to produce the appropriate response. This is especially the case when the unparsed portions of an utterance consist of irrelevant information.

Of course, the defined semantics and corresponding induced grammar do not handle or cover all utterances. This sometimes occurs because an utterance contains complex constructs, such as *"and,"* involves concepts outside the range of our defined semantics, or are actually meaningless.

5.5 Overall Results

Our experiments in semantic parsing resulted in decent parsing coverage, with 40% coverage of ATIS and 64% coverage of Jupiter. Had we spent more time and effort on the engineering task of writing semantic constraints, we could have improved each of these numbers. Instead, we allocated our resources towards developing grammar induction mechanisms and evaluating the learned grammar through the syntactic parsing experiments described earlier in this chapter.

Figure 5-6 summarizes these experiments by displaying two different views of their results. The first plot compares the absolute coverage of semantic parsing to that of syntactic parsing, displaying the identical results for the ATIS TRAIN set (trying a grammar in the exact set from which it was learned) and the nearly identical results in the separate ATIS TEST set, with about 40% coverage overall. It also displays the divergence in results in the Jupiter TEST set, with 64% coverage in semantic parsing

utterance:	show me flights on monday from philadelphia to boston after seven a_m
syntactic parse:	[[V Pro] [[[[N [P Name]] [P Name]] [P Name]] [P [Number Clock]]]]
meaning representation:	{show :topic2 {speaker :who "me"} :topic {flight :from {city :name "philadelphia"} :to {city :name "boston"} :pred {on :topic {weekday :name "monday"}} :pred {after :topic {time :hour 7 :merid "a_m"}} } }
utterance:	could you please tell me the cheapest fare from atlanta to boston
syntactic parse:	Aux Pro [Please [[V Pro] [Det [Sup [[N [P Name]] [P Name]]]]]]
meaning representation:	({can},{you}, {show :topic2 {speaker :who "me"} :topic {fare :from {city :name "atlanta"} :to {city :name "boston"} })
utterance:	show me round_trip fares between san_francisco and washington_d_c
syntactic parse:	[[V Pro] [Adj N]] PComp Name Conj Name
meaning representation:	({show :topic2 {speaker :who "me"} :topic {fare :pred { trip_type :name "round_trip"} }, {between}, {city :name "san_francisco"},{and}, {city :name "washington_d_c"})

Figure 5-4: Examples of parses from ATIS.
This displays examples of parses from ATIS. One parse is complete and contains a reasonable and useful meaning representation. Another parse is incomplete but contains enough useful information that would likely allow an interpreter to generate the appropriate response to this utterance. Finally, the third parse is incomplete and fails to capture the basic meaning of the sentence, due to the presence of *"and,"* which is not handled by our defined functions, constraints, or learned rules.

utterance:	what is the weather forecast for boston
syntactic parse:	[WhObj Cop [Det [[N N] [P Name]]]]
meaning representation:	{wh_ques :topic {WhObj :pred {quant :wh "what"}} :comp {forecast :for {city :name "boston"} :nnrel {weather} :pred {quant :sp "the"} } }
utterance:	i would like a weather forecast please
syntactic parse:	Pro Aux [[V [[Det N] N]] Please]
meaning representation:	({speaker :who "me"}, {will}, {desire :topic {forecast :nnrel {weather} :pred {quant :sp "a"} })
utterance:	is it raining in alaska
syntactic parse:	Cop Pro [Ving [P Name]]
meaning representation:	({be},{it}, {rain :in {state :name "alaska"}})
utterance:	can you give me the low and high temperatures in antarctica
syntactic parse:	[Aux Pro [V Pro]] Det Adj Conj [[Adj N] [P Name]]
meaning representation:	({show :topic2 {speaker :who "me"}}, {quant :sp "the"}, {low},{and}, {temperature :in {continent :name "antarctica"} :pred {high}})
utterance:	i would like to know the weather danny get off the phone in wyoming
syntactic parse:	Pro Aux V INF [V [Det N]] Unk V Unk Det Unk [P Name]
meaning representation:	(...)

Figure 5-5: Examples of parses from Jupiter.
This displays examples of parses from Jupiter. Again, one parse is complete and contains a reasonable and useful meaning representation. Other parses are incomplete and contain varying amounts of useful information. The final parse is completely meaningless, due to the speaker's embedding an unrelated utterance within the original query (the parser utilizes `Unk` for the part of speech for unknown words from this unrelated "domain").

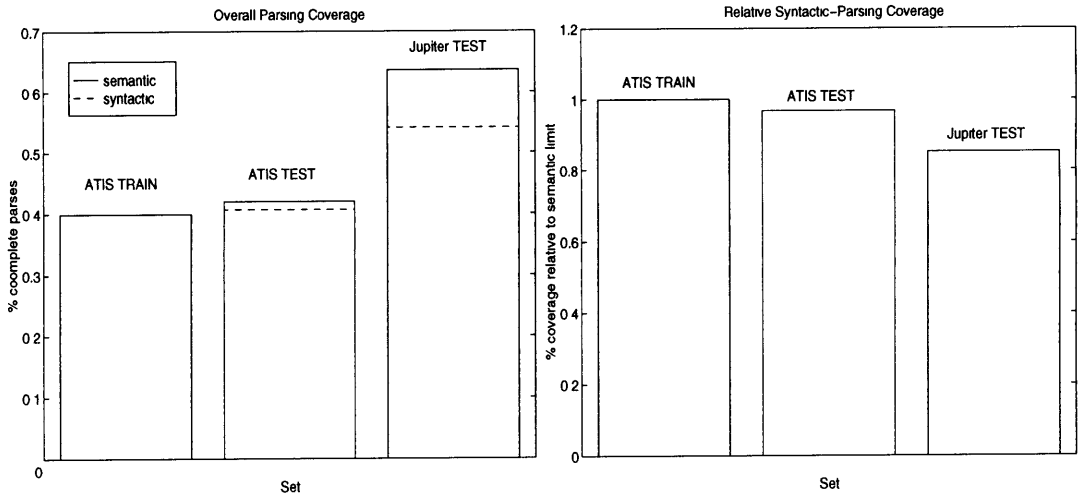


Figure 5-6: Overall parsing coverage.
These graphs display the overall parsing coverage of the ATIS TRAIN, ATIS TEST, and Jupiter TEST sets. The graph on the left compares the absolute coverage of semantic parsing and syntactic parsing. The system achieves 40% coverage of the ATIS sets for both semantic and syntactic parsing; it achieves 64% semantic coverage and 54% syntactic coverage of the Jupiter set. Because these experiments are effectively restricted to the semantically valid subsets of each of the ATIS TRAIN, ATIS TEST, and Jupiter TEST sets, syntactic parsing can do no better than semantic parsing. Accordingly, the graph on the right displays the results of syntactic parsing relative to the semantic parsing upper bound.

compared to 54% coverage in syntactic parsing. Because these results are restricted by semantic filtering, imposing the results of semantic parsing as an upper bound, the second plot displays the coverage of syntactic parsing relative to this semantic parsing upper bound. As shown in this figure, of the utterances that are completely handled by semantic parsing, the grammar covers 100% of those in ATIS TRAIN and 98% in ATIS TEST. Finally, it covers 85% of the utterances in Jupiter TEST that are covered semantically.

Why does the performance of the grammar degrade in the portability experiment? One must analyze the utterances and the types of utterances that were covered semantically but not syntactically in order to answer that question. In our analysis, we discovered that most of the degradation in performance occurred at the sentence-level, with several types of sentences in Jupiter not being covered by the syntactic rules learned in ATIS. Indeed, in analyzing the utterances in these two domains, one can easily see that each domain tends to have different kinds of sentences and sentence

types.

For example, while both domains contain quite a few *"what is ..."* type queries, Jupiter contains a large number of the more unique *"is it ..."* construct. It is quite common for a speaker to inquire, *"is it raining in boston?"* However, no similar types of queries exist in ATIS; the topics in ATIS simply do not compel users to pose those types of queries.

Therefore, we believe that the difference in performance in these different domains can be attributed to the different types of rules native to each domain. Because the system never encounters any *"is it ..."* constructs in ATIS, no such rules are learned in the grammar induction stage, and therefore the grammar loses coverage of those types of utterances in ATIS. The system does not account for these different types of sentences, preventing the sentence-level rules (and hence the overall grammar), from being as portable as one would desire.

Chapter 6

Conclusion

This chapter discusses our findings and conclusions from our research. First, we present a summary of our research and experiments. We then describe the achievements and contributions of this thesis work. Finally, we conclude with our ideas for future work involving further research and experiments which can improve these results and make the overall system more useful.

6.1 Summary

This thesis explored the feasibility of using semantics as a tool for learning grammars automatically. We first developed a semantic parser for parsing utterances based on meaning. Subsequently, we tried to learn a grammar from a semantic parse of a training set of utterances, using two techniques to extract grammar rules from bracketings of parts of speech in the semantic parse logs. The first induction technique, unique bracketings, attempted to learn a generalized grammar by clustering simple, specific rules learned from each unique construct from semantic parsing, and the second technique, semantic-head driven induction, used the semantic phrasal information in these logs to influence the learning and creation of the syntactic structure and rules. For reasons mentioned in this thesis, we settled on using the SHDI-learned grammar for our syntactic parsing experiments.

To assess the performance of this learned grammar, we conducted two parsing

experiments. We first measured the coverage of the grammar in new utterances from the same domain in which it was learned, achieving 98% coverage of the semantically valid subset of our test utterances. We then measured the coverage of this grammar in a new domain to judge the portability of this grammar. The grammar covered 85% of the semantically valid subset of the utterances from the portability experiment, making us optimistic that SHDI had potential for producing useful, portable grammars.

6.2 Contributions

We feel that this thesis research effectively demonstrates that one can indeed use semantics in the learning of syntactic grammar rules. The SHDI mechanism produces a readable and usable grammar, as ascertained by the experiments described in the previous chapter. Furthermore, we believe that our portability experiment illustrates that this learned grammar is not overly domain specific and has potential for producing grammars that work well across domains, reducing the amount of time needed to port a system to a new domain.

Our research contains two major contributions. First, semantic parsing proves to be a surprisingly fast and efficient mechanism for computing semantics from utterances directly. This simple technique admittedly has difficulty with certain concepts – namely syntactic ones like numbers. However, for many utterances, semantic parsing seems to be adequate for generating a meaning representation without relying on any syntactic grammar whatsoever. One can therefore imagine using semantic parsing or related techniques for natural language understanding (NLU); certainly, the feasibility of using just semantic parsing for NLU warrants consideration and further investigation.

Second, semantic-head driven induction also turns out to be an interesting grammar induction technique. As mentioned, it can extract a readable, recursive (premerged) grammar directly from the logs of semantic parsing; this grammar performs well in syntactic parsing and portability experiments. The ability to learn a pre-

merged grammar eliminates the need for further, potentially time-consuming clustering, and unlike many other techniques, this one produces very readable rules, making it easy to reason about the system and the rules themselves.

We feel that the developed mechanisms in this thesis can be of use in automatic speech recognition (ASR) systems. Of course, one can integrate the grammar and semantic computation with the natural language component of an ASR system. One can also utilize the learned grammar in the language modeling used in the recognition task itself, constraining the recognition search to syntactically and semantically valid utterances. This can be accomplished by using the grammar to develop n-grams, which are central to many speech recognition systems. In addition, this work might be useful in recognizers that utilize parsing to prune its list of hypothesized words and phrases. Perhaps semantic parsing itself can even be used in a recognizer for helping constrain the recognition search.

6.3 Future Work

We would like to explore four primary issues for possible future work in this research. First, we would like to improve the overall speed of the parsing mechanism through the use of statistics. Second, we want to utilize syntax for semantically difficult concepts. Third, we want to investigate the issue of portability by refining the sentence-level rules and the mechanism through which they are learned. Fourth, we would like to investigate whether or not one can apply the techniques and results of this thesis towards the automatic learning of semantic constraints. We will now discuss each of these issues in more detail.

6.3.1 Statistical Parsing

The biggest deficiency in this work involves the speed of using the grammar in the syntactic parser; this parsing is simply too slow. The slowness of the parsing actually prevents us from running the desirable experiment of syntactic parsing *without* semantic filtering, and we need to find a way to resolve this speed issue in order to

run those kinds of experiments and make further assessments about the power and usability of the grammar.

The speed problem may not lie in the grammar itself, but rather in the type of parser we use. Indeed, we could use a statistical parser which produces a best-first parse of an input utterance (as opposed to the *all*-parses mechanism we currently use, which computes all possible parses of an utterance) and improve the speed of the system dramatically. With the rule count information present in the SHDI-learned grammar, converting this grammar from a simple set of CFG rules to a set of stochastic CFG rules would be relatively easy and straightforward. With this change, running more and different kinds of syntactic parsing experiments would be more feasible. Furthermore, with these statistics, we could utilize other measurements, such as perplexity, to assess the usefulness of the grammar. Finally, we hope that the anticipated speed improvement will lead to a syntactic parsing system which is faster and more efficient than semantic parsing for extracting meaning representations from an utterance, because of the guidance provided by syntactic structure.

6.3.2 Use of Syntax

During this thesis research, we heavily focus our efforts on eliminating syntax altogether from the semantic-based system. Perhaps this approach is unnecessary and even too extreme. While using semantics alone works well overall, this system clearly cannot handle certain syntax-centric concepts such as numbers; numbers need a sense of ordering imposed upon them in order to be interpreted. While it is relatively easy to handle numbers syntactically, handling them semantically, while ignoring word order, proves to be rather difficult. Therefore, a system that integrates syntax and semantics, relying on seeded syntactic rules for handling semantically-difficult but syntactically-straightforward concepts, can get better overall performance and results. Such an integrated system may also be better for performing NLU as well, using syntax for syntactically-biased concepts, and relying on semantics and constraints for performing semantic parsing of other utterances as before.

6.3.3 Sentence-Level Rules

While the learned grammar performs quite well in covering new utterances from the same domain in which it was learned, the performance degrades somewhat in a new domain. Specifically, the major factor which keeps the learned grammar from being as portable as we desired involves sentence-level rules. Because different domains contain different types of sentences (such as the *"is it..."* construct), a grammar trained solely in one domain cannot learn the sentence-level rules needed to cover other domains. Therefore, we would like to improve the way these rules are learned. We can accomplish this by providing the system with example sentences and pre-labeled sentence types, allowing the system to learn sentence-level rules in a supervised fashion. Alternatively, we can more simply learn a robust set of sentence-level rules by training the system over *several* domains. By using a broader training set and providing enough data and samples of many different kinds of sentences, the system is less likely to encounter unseen sentence types and constructs in new test data from different domains.

6.3.4 Automatic Learning of Semantic Constraints

It would be very interesting to try to mirror the results of this thesis in trying to learn semantic constraints automatically. Porting a system to a new domain can be a rather difficult and time-consuming task, involving the development of a new grammar and the definition of a new set of semantics and constraints for the domain. This thesis simplifies this task somewhat by attempting to learn the grammar automatically and even demonstrates that this grammar can potentially be used in another domain (especially if the sentence-level rules can be improved). However, the engineering task of writing semantic constraints still remains. Therefore, it would be worthwhile to try to learn these constraints automatically.

This thesis research begins by using semantics and constraints to perform a semantic parse in one domain and learn a grammar from this parse. It then attempts to perform a syntactic parse in another domain using this grammar. One can therefore

imagine trying to use the syntactic parse (using a statistical parser to avoid the need for semantic filtering) in the new domain to deduce the correlation between different words and determine how often they occur together. This could lead to the ability to learn the semantic constraints for a domain automatically, much like the existing system learns syntactic rules automatically. Hence, such a system could effectively port itself across domains.

6.3.5 Other Possible Improvements

Of course, there are other improvements that we could make to the system. We could remove the adjacency constraints and let the system ignore word order altogether, letting it combine words and phrases regardless of where they appear in an utterance. This would be computationally complex, however, so pursuing this goal of completely ignoring word order requires investigation into finding a very efficient implementation.

Our research also indicates that the concept of ignoring filler words works well for handling sentence-level semantics. We can expand this so that filler words are ignored not just at the sentence-level, but also for any of the other semantic functions. This can certainly improve results and make the system more robust to disfluencies in speech.

However, we believe that the immediately useful and most significant improvements involve the four mentioned areas. By improving the speed of the parser through statistics and using syntax for semantically-difficult concepts, we hope to make semantic parsing and the resulting SHDI-learned rules more powerful and more useful. Finally, we are optimistic that broadening the training set to make the learned sentence-level rules more robust and attempting to learn constraints automatically can strengthen semantic parsing's and SHDI's potential for making the task of porting a system to new domains easier.

Bibliography

[1] R. Bobrow, R. Ingria, and D. Stallard. Syntactic/semantic coupling in the BBN DELPHI system. In *Proceedings DARPA Speech and Natural Language Workshop*, pages 311–315, Harriman, NY, February 1992.

[2] E. Brill, D. Magerman, M. Marcus, and B. Santorini. Deducing linguistic structure from the statistics of large corpora. In *Proceedings DARPA Speech and Natural Language Workshop*, pages 275–281, Hidden Valley, PA, June 1990.

[3] E. Brill and M. Marcus. Automatically acquiring phrase structure using distributional analysis. In *Proceedings DARPA Speech and Natural Language Workshop*, pages 155–159, Harriman, NY, February 1992.

[4] K-S Fu and T.L. Booth. Grammatical inference: Introduction and survey – Parts I and II. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-5 No. 1 and No. 4, January and July 1975.

[5] C.L. Giles, C.B. Miller, D. Chen, G.Z. Sun, H.H. Chen, and Y.C. Lee. Extracting and learning an unknown grammar with recurrent neural networks. *Advances in Neural Information Processing Systems*, 4:317–324, 1992.

[6] A.L. Gorin, S.E. Levinson, and A.N. Gertner. Adaptive acquisition of spoken language. In *Proceedings ICASSP*, pages 805–808, Toronto, Canada, April 1991.

[7] A.L. Gorin, S.E. Levinson, L.G. Miller, A.N. Gertner, A. Ljolje, and E.R. Goldman. On adaptive acquisition of language. In *Proceedings ICASSP*, pages 601–604, Albuquerque, NM, April 1990.

[8] L. Hirschman et al. Multi-site data collection for a spoken language system. In *Proceedings DARPA Speech and Natural Language Workshop*, pages 7–14, Harriman, NY, February 1992.

[9] M. Kay. Algorithm schemata and data structures in syntactic processing. In B. Grosz, K. Sparck Jones, and B. L. Webber, editors, *Readings in Natural Language Processing*, pages 35–70. Morgan Kaufmann Publishers, Inc., 1986.

[10] E. Levin and R. Pieraccini. Concept-based spontaneous speech understanding system. In *Proceedings Eurospeech*, pages 555–558, Madrid, Spain, September 1995.

[11] H. Lucke. Inference of stochastic context-free grammar rules from example data using the theory of Bayesian belief propagation. In *Proceedings Eurospeech*, pages 1195–1198, Berlin, Germany, September 1993.

[12] M. McCandless. Automatic acquisition of language models for speech recognition. Master's thesis, Massachusetts Institute of Technology, May 1994.

[13] M. Negishi. Grammar learning by a self-organizing network. *Advances in Neural Information Processing Systems*, 7:27–31, 1995.

[14] E. Vidal, F. Casacuberta, and P. Garcia. Grammatical inference and automatic speech recognition. *NATO ASI series. Series F, Computer and system sciences*, 147:174–191, 1995.

[15] V. Zue, S. Seneff, J. Glass, L. Hetherington, E. Hurley, H. Meng, C. Pao, J. Polifroni, R. Schloming, and P. Schmid. From interface to content: Translingual access and delivery of on-line information. In *Proceedings Eurospeech*, pages 2227–2230, Rhodes, Greece, September 1997.