Appears in *Proceedings of the ICML-97 Workshop on Automata Induction, Grammatical Inference, and Language Aquicision,* Nashville, TN, July 1997

Learning to Parse Natural Language Database Queries into Logical Form

Cynthia A. Thompson and **Raymond J. Mooney** and **Lappoon R. Tang**
Department of Computer Sciences
University of Texas
Austin, TX 78712-1188
(cthomp,mooney,rupert)@cs.utexas.edu

Abstract

For most natural language processing tasks, a parser that maps sentences into a semantic representation is significantly more useful than a grammar or automata that simply recognizes syntactically well-formed strings. This paper reviews our work on using inductive logic programming methods to learn deterministic shift-reduce parsers that translate natural language into a semantic representation. We focus on the task of mapping database queries directly into executable logical form. An overview of the system is presented followed by recent experimental results on corpora of Spanish geography queries and English job-search queries.

Introduction

Language learning is frequently interpreted as acquiring a recognizer, a procedure that returns "yes" or "no" to the question: "Is this string a syntactically well-formed sentence in the language?". However, a black-box recognizer is of limited use to a natural language processing system. A simple recognizer may be useful to a limited grammar checker or a speech recognizer that must choose between several word sequences as possible interpretations of an utterance. But, for most natural language tasks, sentences need to be transformed into some internal representation that is actually useful for tasks such as question answering, information extraction, summarizing, or translation.

Language learning has also been interpreted as acquiring a set of production rules (e.g. `S` $\rightarrow$ `NP VP`) that define a formal grammar that recognizes the positive strings. This is more useful than a black-box recognizer since it allows a standard syntactic parser to produce parse trees that may be useful for further processing. However, most natural language grammars assign multiple parses to sentences, most of which do not correspond to meaningful interpretations. For example, any standard syntactic grammar of English will produce an analysis of "The man ate the pasta with a fork" that attaches the prepositional phrase "with a fork" to "pasta" as well as to "ate" despite the fact that people generally do not consume eating utensils (i.e. compare "The man ate the pasta with the cheese"). In fact, any standard syntactic English grammar will produce more than 2^n parses of sentences ending in n prepositional phrases, most of which are usually spurious (Church & Patil 1982).

A truly useful parser would produce a unique or limited number of parses that correspond to the meaningful interpretations of a sentence that a human would actually consider. As a result, the emerging standard for judging a syntactic parser in computational linguistics is to measure its ability to produce a single best parse tree for a sentence that agrees with the parse tree assigned by a human judge (Periera & Shabes 1992; Brill 1993; Magerman 1995; Collins 1996; Goodman 1996). This approach has been facilitated by the construction of large *treebanks* of human-produced syntactic parse trees for thousands of sentences, such as the Penn Treebank (Marcus, Santorini, & Marcinkiewicz 1993), which consists primarily of analyses of sentences from the Wall Street Journal.

Although useful, syntactic analysis is only part of the larger problem of natural language understanding. In this paper, the term "parser" should be interpreted broadly as any system for mapping a natural language string into an internal representation that is useful for some ultimate task, such as answering questions, translating to another natural language, summarizing, etc.. Parsing can range from producing a syntactic parse tree to mapping a sentence into unambiguous logical form. Figure 1 shows examples of three types of parses, a syntactic parse of a sentence from the ATIS (Airline Travel Information System) corpus of the Penn treebank, a case-role (agent, patient, instrument) analysis of a simple sentence, and a logical form for a database query about U.S. geography. Given the appropriate database in logical form, the final form for the third example can be directly executed in Prolog to retrieve an answer to the given question.

Appears in *Proceedings of the ICML-97 Workshop on Automata Induction, Grammatical Inference, and Language Aquicision,* Nashville, TN, July 1997

Syntactic Parse Tree

Show me the flights that served lunch departing from San Francisco on April 25th.

```
s:[np:[*],
   vp:[show,
       np:[me],
       np:[np: [np:[the, flights],
                sbar:[that,
                      s:[np:[t],
                         vp:[served,
                             np:[lunch]]]]],
           vp:[departing,
               pp:[from,
                   np:[san, francisco]],
               pp:[on,
                   np:[april, '25th']]]]]]
```

Case-Role Analysis

The man ate the pasta with the fork.

```
[ate,agt:[man,det:the],pat:[pasta,det:the],
   inst:[fork,det:the]]
```

Executable Logical Form

What is the capital of the state with the largest population?

```
answer(C, (capital(S,C),
   largest(P,(state(S),population(S,P))))).
```

Figure 1: Examples of Several Types of Parses

Learning parsers or transducers that can produce semantic analyses such as case-role assignments or logical forms is significantly more useful for natural language processing than learning a syntactic recognizer or grammar. There has been some research using both neural networks and symbolic induction to learn parsers that produce case-role analyses (McClelland & Kawamoto 1986; Miikkulainen 1993; St. John & McClelland 1990; Zelle & Mooney 1993; Miikkulainen 1996). Even more useful is a parser that can map natural language queries into a logical form or a database query language (e.g. SQL) that can be immediately executed to retrieve an answer to the question.

Consequently, our recent research has focused on learning parsers that map natural language database queries into an executable logical form. The system we have developed, called CHILL[1] (Zelle 1995), uses *inductive logic programming* (ILP) (Muggleton 1992; Lavrač & Džeroski 1994) to learn a deterministic shift-reduce Prolog parser. CHILL has been demonstrated on learning parsers for each type of analysis represented in Figure 1. In particular, we have conducted experiments demonstrating that CHILL learns parsers for answering English queries for a database on U.S. geography that are more accurate than an existing hand-built system for this application (Zelle & Mooney 1996).[2] This paper presents an overview of our approach for learning parsers and presents new experimental results on Spanish geography queries and English job-search queries that demonstrate the robustness of the approach.

Overview of CHILL

A first approach to learning a parser in Prolog for the predicate `parse(Sentence, Representation)` might be to use traditional ILP techniques directly. In addition to the sheer complexity and unconstrained nature of such a task, there is the difficulty of obtaining negative examples and background knowledge with which to guide the learner. Instead, CHILL begins with a well-defined parsing framework, shift-reduce parsing, and uses ILP to learn control strategies within this framework. By doing so, we are mapping language learning into standard, traditional concept learning; we view grammar learning as learning to control the actions of a parser. Treating language acquisition as a control learning problem is not in itself a new idea, as discussed in the section on Related Work. However, CHILL is the first system to use ILP techniques that can directly induce over unbounded lists, stacks, and trees rather than less flexible propositional approaches.

The input to CHILL is a set of training instances consisting of sentences paired with the desired parses. The output is a shift-reduce parser in Prolog that maps sentences into parses. CHILL outputs a simple deterministic, shift-reduce parser. The current parse state is represented by the contents of the parse stack and the remaining portion of the input buffer (Tomita 1986).

Consider producing a case-role analysis (Fillmore 1968) of the sentence: "The man ate the pasta." Parsing begins with an empty stack and an input buffer containing the entire sentence. At each step of the parse, either a word is shifted from the front of the input buffer onto the stack, or the top two elements on the stack are popped and combined to form a new element that is pushed back onto the stack. The sequence of actions and stack states for our simple example is shown Figure 2. The action notation *(x label)*, indicates that the stack items are combined via the role *label* with the item from stack position x being the head. Choosing the correct action to apply at any given point in the deterministic parse requires a great

[1] Constructive Heuristics Induction for Language Learning

[2] Previous publications are available on line at `http://www.cs.utexas.edu/users/ml`.

Appears in *Proceedings of the ICML-97 Workshop on Automata Induction, Grammatical Inference, and Language Aquicision,* Nashville, TN, July 1997

Action	Stack Contents
	[]
(shift)	[the]
(shift)	[man, the]
(1 det)	[[man, det:the]]
(shift)	[ate, [man, det:the]]
(1 agt)	[[ate, agt:[man, det:the]]]
(shift)	[the, [ate, agt:[man, det:the]]]
(shift)	[pasta, the, [ate, agt:[man, det:the]]]
(1 det)	[[pasta, det:the], [ate, agt:[man, det:the]]]
(2 obj)	[[ate, obj:[pasta, det:the], agt:[man, det:the]]]

Figure 2: Shift-Reduce Case-Role Parsing of "The man ate the pasta."

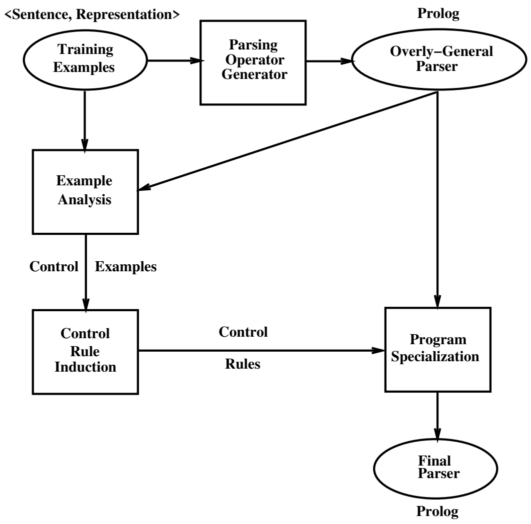

Figure 3: The CHILL Architecture

deal of knowledge. This knowledge is encoded in the rules learned by CHILL.

Figure 3 shows the basic components of the CHILL system. First, during Parsing Operator Generation, the training examples are analyzed to formulate an overly-general shift-reduce parser that is capable of producing parses from sentences. Next, in Example Analysis, the overly-general parser is used to parse the training examples to extract contexts in which the various parsing operators are actually useful. The third step is Control-Rule Induction which employs a general ILP algorithm to learn rules that characterize these contexts. Finally, Program Specialization "folds" the learned control-rules back into the overly-general parser to produce the final parser. The following subsections give details on each of these steps.

Parsing Operator Generation

In the Prolog parsing shell, parsing *operators* are program clauses that take the current stack and input buffer as input arguments and return a modified stack and buffer as outputs. During Parser Operator Generation, the training examples are analyzed to extract all of the general operators that are required to produce the analyses. For example, an operator to reduce the top two items on the stack by attaching the second item as an agent of the top is represented by the clause

```
op([Top,Second|Rest],In,[NewTop|Rest],In) :-
   reduce(Top,agt,Second,NewTop).
```

The arguments of the `op` predicate are the current stack, current input buffer, new stack and new input buffer. The `reduce` predicate simply combines `Top` and `Second` using the role `agt` to produce the new structure for the top of the stack. In general, one such operator clause is constructed for each case-role slot in the training examples. The resulting parser is severely overgeneral, as the operators contain no conditions specifying when they should be used; any operator may be applied to virtually any parse state resulting in many spurious parses.

Example Analysis

In Example Analysis, the overly-general parser is used to parse the training examples to extract contexts in which the various parsing operators should and should not be employed. The context consists of the parse stack and the remaining (natural language) input. These contexts form sets of positive and negative *control examples* from which the appropriate control rules can be induced in the third stage. A control example gives the context that held when a particular operator was applied in the course of parsing an example. Examples of correct operator applications are generated by finding the first correct parsing of each training pair with the overly-general parser; any context to which an operator is applied in this successful parse becomes a positive control example for that operator. Negative examples are generated by making a closed-world assumption. Any context to which the operator could have applied during parsing, but was not, is used as a negative example. Analyzing the examples is eased by the deterministic nature of the parser. If an operator is used in the derivation of the correct parse, it is a correct example of that operator's application. Additional analyses may be found by backtracking.

For the `reduce agt` operator shown above, the sentence "The man ate the pasta." would produce a single positive control example: `op([ate,[man, det:the]], [the,pasta], A, B)`. Note that the first two arguments correspond to the stack contents and input after applying the third shift action in Figure 2. This is the only subgoal to which this operator is applied in the correct parsing of the sentence. The

Appears in *Proceedings of the ICML-97 Workshop on Automata Induction, Grammatical Inference, and Language Aquicision,* Nashville, TN, July 1997

variables A and B are uninstantiated since they are outputs from the op clause and are not yet bound at the time the clause is being applied. The sentence generates the following negative control examples for this operator:

```
op([man,the],[ate,the,pasta],A,B)
op([the,[ate,agt:[man,det:the]]],[pasta],A,B)
op([pasta,the,[ate,agt:[man,det:the]]],[],A,B)
op([[pasta,det:the],[ate,agt:[man,det:the]]],
   [],A,B)
```

Note that we have transformed the training examples, which were all positive examples of correct parses, into control examples that contain both positive and negative examples.

Control-Rule Induction

At this stage, we are ready to apply ILP techniques to induce *control rules* for each operator. This is the Control-Rule Induction phase, which uses the control examples generated during the previous stage as input to the induction algorithm. The algorithm attempts to learn a concept definition for each operator that will classify the context in which it is actually useful. The control rules are comprised of a definite-clause definition that covers the positive control examples for the operator but not the negative.

ILP methods allow the system to induce over the unbounded context of the complete stack and remaining input string. There is no need to reduce this context to a predetermined, fixed set of features as required by propositional approaches such as neural-networks or decision trees. CHILL's ILP algorithm combines elements from bottom-up techniques found in systems such as CIGOL (Muggleton & Buntine 1988) and GOLEM (Muggleton & Feng 1992), and top-down methods from systems like FOIL (Quinlan 1990), and is able to invent new predicates in a manner analogous to CHAMP (Kijsirikul, Numao, & Shimura 1992). Details of the CHILL induction algorithm together with experimental comparisons to GOLEM and FOIL are presented by (Zelle & Mooney 1994) and (Zelle 1995).

Given our running example, a control rule that can be learned for the reduce agt operator is

```
op([X,[Y,det:the]], [the|Z], A, B) :-
      animate(Y).
animate(man). animate(boy). animate(girl) ....
```

Here the system has invented a new predicate to help explain the parsing decisions. The new predicate would have a system generated name, but is called "animate" here for clarity. This rule may be roughly interpreted as stating: "the agent reduction applies when the stack contains two items, the second of which is a completed noun phrase whose head is animate." The output of the Control-Rule Induction phase is a suitable control-rule for each general operator generated during Parsing Operator Generation. These control rules are then passed on to the Program Specialization phase.

Program Specialization

The final step, Program Specialization, "folds" the control information back into the overly-general parser. A control rule is easily incorporated into the overly-general program by unifying the head of an operator clause with the head of the control rule for the clause and adding the induced conditions to the clause body. The definitions of any invented predicates are simply appended to the program. Given the program clause:

```
op([Top,Second|Rest],In,[NewTop|Rest],In) :-
     reduce(Top,agt,Second,NewTop).
```

and the control rule above, the resulting clause is

```
op([A,[B,det:the]],[the|C],[D],[the|C]) :-
     animate(B), reduce(A,agt,[B,det:the],D).
animate(boy). animate(girl). animate(man)...
```

The final parser is just the overly-general parser with each operator clause suitably constrained. This specialized parser is guaranteed to produce all of and only the preferred parse(s) for each of the training examples that could be parsed by the overly-general parser.

Parsing Framework

Some might consider the shift-reduce framework too limiting for NLP, citing well-known results about the power of LR(k) grammars. It is important to note, however, that the potential lookahead in our parser is unlimited since the entire state of the parser (current stack contents and remaining input) may be examined in determining which action to perform. Furthermore, the control-rules that are learned are essentially arbitrary logic programs, therefore the class of languages recognized is, in principle, Turing complete.

Parsing into Logical Form

For simplicity, the above discussion focused on parsing into case-role representations; however, CHILL is fairly easily adapted to produce other types of output such as executable logical forms. Adapting the system requires identifying operators that allow the shift-reduce parser to construct outputs in the desired format.

Logical queries are built using three simple operator types. First, a word or phrase at the front of the input buffer suggests that a certain structure should be part of the result. The appropriate structure is pushed onto the stack. For example, the word "capital" might cause the capital/2 predicate to be pushed

Appears in *Proceedings of the ICML-97 Workshop on Automata Induction, Grammatical Inference, and Language Aquicision,* Nashville, TN, July 1997

on the stack. This type of operation is performed by an `introduce` operator and is very similar to a shift operation. Initially, the logical structures are introduced with new variables as arguments. These variables may be unified with variables appearing in other stack items using a `co-reference` operator. For example, the second argument of the `capital/2` structure may be unified with the first argument of an `answer/2` predicate. Finally, a stack item may be embedded into the argument of another stack item to form conjunctive goals; this is performed by a `conjoin` operation.

Figure 4 shows the sequence of states the parser goes through in parsing the sentence, "What is the capital of Texas?" The state of the parser is shown as a term of the form: `ps(Stack,Input)` where `Stack` is a list of constituents comprising the current stack and `Input` are the remaining words of the input buffer. The `answer` predicate is automatically placed on the parse stack of each sentence parsed, as it is a required component of the final logical query.

For each operator class, the individual operators required to parse the training examples are easily inferred during Parser Operator Generation. The necessary `introduce` operators are determined by examining what structures occur in each query and which words in the corresponding sentence can introduce those structures. This requires a semantic lexicon that gives for each word the logical structures it can introduce. Ambiguous words produce multiple operators, one for introducing each possible meaning. `Co-reference` operators are constructed by finding the shared variables in the training queries; each sharing requires an appropriate operator instance. Finally, `conjoin` operations are indicated by the term-embedding exhibited in the training examples.

Once the appropriate set of unconstrained initial operators are constructed, CHILL proceeds to learn control rules that constrain the operators to produce only the desired output for each of the training examples. For ambiguous words, for example, control rules are learned that select the appropriate meaning based on context. The resulting parser can then be tested on its ability to produce accurate logical forms for novel sentences.

```
        Parse State                                        Operation Type

ps([answer(_,_):[]],
   [what,is,the,capital,of,texas,?])                        shift
ps([answer(_,_):[what]],
   [is,the,capital,of,texas,?])                             shift
ps([answer(_,_):[is,what]],
   [the,capital,of,texas,?])                                shift
ps([answer(_,_):[the,is,what]],
   [capital,of,texas,?])                                    introduce
ps([capital(_,_):[],
    answer(_,_):[the,is,what]],
   [capital,of,texas,?])                                    co-reference
ps([capital(_,A):[],
    answer(A,_):[the,is,what]],
   [capital,of,texas,?])                                    shift
ps([capital(_,A):[capital],
    answer(A,_):[the,is,what]],
   [of,texas,?])                                            shift
ps([capital(_,A):[of,capital],
    answer(A,_):[the,is,what]],
   [texas,?])                                               shift/introduce
ps([equal(_,stateid(texas)):[texas],
    capital(_,A):[of,capital],
    answer(A,_):[the,is,what]],
   [?])                                                     co-reference
ps([equal(B,stateid(texas)):[texas],
    capital(B,A):[of,capital],
    answer(A,_):[the,is,what]],
   [?])                                                     conjoin
ps([equal(B,stateid(texas)):[texas],
    answer(A,capital(B,A)):[the,is,what]],
   [?])                                                     shift
ps([equal(B,stateid(texas)):[?,texas],
    answer(A,capital(B,A)):[the,is,what]],
   [])                                                      shift
ps(['EndOfInput',
    equal(B,stateid(texas)):[?,texas],
    answer(A,capital(B,A)):[the,is,what]],
   [])                                                      conjoin
ps(['EndOfInput',
    answer(A,(capital(B,A),
              equal(B,stateid(texas)))):[the,is,what]],
   [])
```

Figure 4: Sequence of Parse States for "What is the capital of Texas?"

Experimental Results

Geography Queries

Our initial results involve a database for United States geography for which a hand-coded natural-language interface already exists. The existing system, called *Geobase* was supplied as a sample application with Turbo Prolog 2.0 (Borland International 1988). This system provides a database already coded in Prolog and also serves as a convenient benchmark against which CHILL's performance can be compared. The database contains about 800 Prolog facts asserting relational tables for basic information about U.S. states, including: population, area, capital city, neighboring states, major rivers, major cities, and highest and lowest points along with their elevation.

The natural language data for the experiment was gathered by asking uninformed subjects to generate sample questions for the system. An analyst then paired the questions with appropriate logical queries to generate an experimental corpus of 250 examples. The original questions were collected in English, and the example in Figure 1 is taken from this corpus. The English queries were recently translated into Spanish to provide a new test corpus for the system. Additional examples from the corpus in both English and Spanish are given in Figure 5.

Experiments were then performed by training on

Appears in *Proceedings of the ICML-97 Workshop on Automata Induction, Grammatical Inference, and Language Aquicision,* Nashville, TN, July 1997

What state has the most rivers running through it?
¿Cual estado tiene mas rios corriendo por el?
```
answer(S, most(S, R, (state(S), river(R),
    traverse(R,S)))).
```

How many people live in Iowa?
¿Cuantas personas viven en Iowa?
```
answer(P, (population(S,P),
    equal(S, stateid(iowa)))).
```

What are the major cities in Kansas?
¿Que son las ciudades mayores en Kansas?
```
answer(C, (major(C), city(C), loc(C,S),
    equal(S,stateid(kansas)))).
```

Figure 5: Sample Geography Queries in English and Spanish

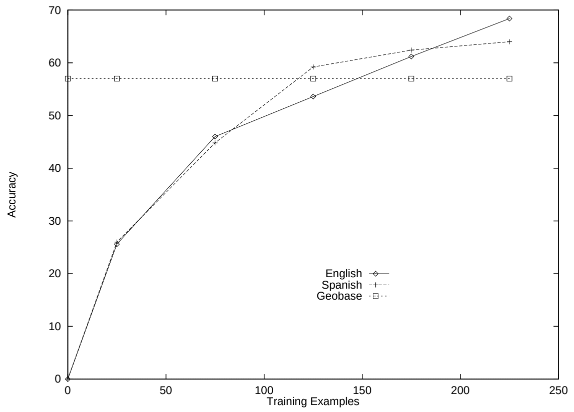

Figure 6: CHILL Accuracy on Geography Queries

random subsets of either the English or Spanish corpus and evaluating the resulting parser on 25 unseen examples (in the same language as the training sentences). The parser was judged to have processed a new sentence correctly when the generated query produced exactly the same final answer from the database as the answer from the query provided by the analyst. Hence, the metric is a true measure of the performance for a complete database-query application in this domain.

Figure 6 shows the accuracy of CHILL's parsers over a ten trial average. The line labeled "Geobase" shows the average accuracy of the Geobase system on the English test sets. The "English" curve shows that CHILL outperforms the hand-built English system when trained on 175 or more examples. Most of the "errors" CHILL makes on novel questions are due to an inability to parse the query rather than generation of an incorrect answer. After 225 training examples, only slightly over 2% of novel English questions on average are actually answered incorrectly. The remaining 30% of the questions that are not answered correctly are simply unable to be parsed. In these cases, the user might be asked to rephrase the question or just told the question was not understood.

The system learned for Spanish is only slightly less accurate after 225 training examples, as indicated by the "Spanish" curve in the figure. However, the Spanish corpus is learned more quickly. After 125 training examples, the Spanish parser outperforms the hand-built parser and the learned English parser, but the learned English version climbs ahead at 225 examples. For the Spanish version, only slightly over 3% of novel questions on average are actually answered incorrectly after 225 training examples.

Job Queries

As part of an on-going project, we are using learning methods to develop systems for extracting information from a USENET newsgroup and answering natural-language questions about the resulting database. We hope to field an application on the world-wide-web that will attract a significant number of users and therefore serve as a source of larger amounts of realistic language data for training and testing. The specific application we are currently pursuing is a system that can process computer job announcements posted to the newsgroup `misc.jobs.offered`, extract a database of available jobs, and then answer natural language queries such as "What jobs are available in California for C++ programmers paying over $100,0000 a year?"

In order to attract a sufficient number of initial users, we must first build a prototype that is reasonably accurate. Questions that this initial system is unable to parse will then be collected, annotated, and used to retrain the system to improve its coverage. In this way, learning techniques can be used to automatically improve and extend a system, based on data collected during actual use. In order to construct the initial system, we have developed a corpus of sample queries that were artificially generated from a hand-built grammar of question templates. Examples of templates from this grammar are shown in Figure 7, where square brackets indicate an optional word, curly braces indicate the choice of one word, and angle brackets indicate the occurrence of a word from that category. Only some of the many possibilities are given.

Random examples are then generated from this grammar to create an initial corpus of queries for testing the system. Examples of these artificially generated queries are shown in Figure 8 paired with the associated query language representation.

A corpus of 750 such queries was generated for initial

Appears in *Proceedings of the ICML-97 Workshop on Automata Induction, Grammatical Inference, and Language Aquicision,* Nashville, TN, July 1997

Show me [the] jobs in {<location> | <area> |
<language>} [on <platform>].
Show me [{<company> | <area> | <language> |
<title> | <location> | <platform>}]
jobs [for {<company> | <salary> | <degree>}].
What jobs are there {with <company> |
for an <area> specialist | using <language>}?

Figure 7: Sample Job-Query Templates

What jobs are there in Austin for Lamreen Inc.?

```
answer(J, (job(J), loc(J,C),
    equal(C, austin), company(J,I),
    equal(I,'Lamreen Inc.'))).
```

Are there any jobs at Lion's Time using C++ on Windows 95?

```
answer(A, (job(A), company(A,B),
    equal(B,'Lion''s Time'), language(A,C),
    equal(C,'c++'), platform(A,D),
    equal(D,'windows 95'))).
```

Show me California C jobs in 3D graphics.

```
answer(J, (loc(J,S), equal(S, california),
    language(J,L), equal(L,C), job(J),
    area(J, A), equal(A,'3d graphics'))).
```

Figure 8: Sample Job Queries

experimentation. Experiments were performed analogously to those for the Geoquery corpus, testing the ability of the learned parser to generate queries for 50 novel sentences. Figure 9 shows the accuracy of CHILL's parsers over a ten trial average. In this case we had no hand-built system to compare to.

After 700 training examples, 89.7% of the novel examples could be parsed into queries that returned the correct job(s) from the database, an encouraging result. The number of questions answered incorrectly is slightly higher in this domain than in the geography domain; 5% of the novel queries are actually answered incorrectly. This is in part due to the `job/1` predicate present in the query of every training example in this corpus. It is not too hard for CHILL to learn a parser that will introduce this predicate, and no others, into almost any query. This would lead to all jobs being retrieved from the database, when in fact only a few were requested. This predicate is needed to maintain the flexibility of the query language, and investigation is needed to determine a way to improve this result.

Related Work

(Berwick 1985) also used the approach of treating language acquisition as a control learning problem, by learning control rules for a Marcus-style deterministic parser (Marcus 1980). When the system came to a parsing impasse, a new rule was created by inferring the correct parsing action and creating a new rule using certain properties of the current parser state as trigger conditions for its application. In a similar vein, (Simmons & Yu 1992) controlled a simple shift-reduce parser by storing example contexts consisting of the syntactic categories of a fixed number of stack and input buffer locations. New sentences were parsed by matching the current parse state to the stored examples and performing the action performed in the best matching training context. Finally, (Miikkulainen 1996) presents a connectionist approach to language acquisition that learns to control a neural-network parsing architecture that employs a *continuous stack*. Like the statistical approaches mentioned previously, these control acquisition systems all used feature-vector representations.

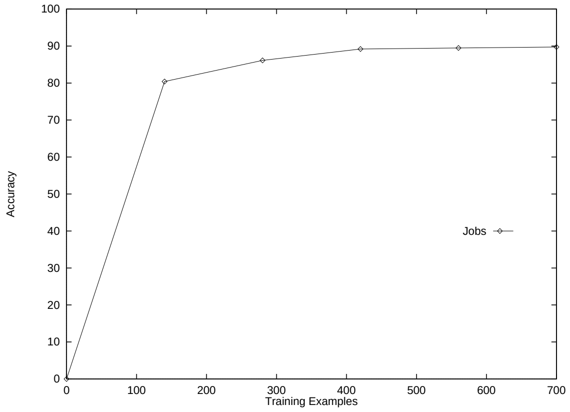

Figure 9: CHILL Accuracy on Job Query Domain

Most systems differ from CHILL along two fronts: type of analysis provided and type of training input required. CHILL learns parsers that produce complete, labeled parse trees; other systems have learned to produced simple bracketings of input sentences (Periera & Shabes 1992; Brill 1993), or probabilistic language models that assign sentences probabilities (Charniak & Carroll 1994). While CHILL requires only a suitably annotated corpus, other approaches have utilized an existing, complex, hand-crafted grammar that over-generates (Black *et al.* 1993; Black, Lafferty, & Roukos 1992). CHILL's ability to invent new categories also allows actual words to help make parsing decisions, whereas many systems are limited to representing sentences as strings of lexical categories (Brill 1993; Charniak & Carroll 1994).

The approach of (Magerman 1994; 1995) is more similar to CHILL. His system produces parsers from annotated corpora of sentences paired with syntactic representations. Parsing is treated as a problem of statistical pattern recognition. CHILL differs from this approach mainly in its flexibility. Magerman's system is hand-engineered for the particular representation being produced. Given this hand-crafting of features and rules, it is unclear how easily the approach could be adapted to differing representation schemes.

One approach that learns more semantically oriented representations is the hidden understanding models of (Miller *et al.* 1994). This system learns to parse into tree-structured meaning representations. These representations are similar to syntactic parse trees except that the nodes may be labeled by conceptual categories as in the analyses produced by semantic grammars. This approach was recently extended to construct a complete interface with separate statistically trained modules for syntactic, semantic and discourse analysis (Miller *et al.* 1996). However, the mapping to a final semantic representation employs two separate modules, requiring each training sentence to be labeled with both a parse tree *and* a semantic frame. CHILL maps directly into logical form and does not require annotating sentences with any additional intermediate representations. The hidden understanding model utilizes a propositional approach which renders it incapable of modeling phenomena requiring nonlocal references, a situation that does not hold for CHILL, which may examine any aspect of the parse context.

Future Work and Conclusions

There are still issues remaining to be resolved in using CHILL to develop NLP systems. Particularly in the job query domain, work remains to be done. Querying an expanded database, and "data mining" type queries will be investigated. For example, a user might want to know "How many jobs were posted in the last two months requiring a C++ programmer?". Both the training data and database language would need updating to handle such a query. Lowering the number of incorrect parses produced in the jobs domain is also a goal. Putting the system on the world-wide-web to monitor its performance on actual user's queries is the next logical step. Finally, we are working on methods to map from natural language sentences directly into SQL queries.

The ability to induce black-box recognizers or generic production-rule grammars is of limited utility to natural language processing. To significantly aid many natural language tasks, an efficient parser (transducer) that translates sentences into meaningful semantic representations is required. CHILL uses inductive logic programming to induce parsers that translate sentences into semantic representations given a training corpus of I/O pairs. In particular, it is capable of learning parsers for mapping natural language queries directly into executable logical form and can thereby construct complete natural-language database interfaces.

This paper has presented recent results on two new tasks, processing Spanish geography queries and English job queries, demonstrating the robustness of the system to construct a range of database interfaces. We hope that this work will challenge and encourage others interested in grammar learning to look more closely at the query mapping problem. Towards this end, our data is available on our WWW page at `http://www.cs.utexas.edu/users/ml/`.

Acknowledgements

Thanks to John Zelle as the primary developer and implementer of CHILL. Thanks to Agapito Sustaita for translating the geography queries into Spanish. This research was partially supported by the National Science Foundation through grant IRI-9310819 and through a grant from the Deimler-Benz Palo Alto Research Center.

References

Berwick, B. 1985. *The Acquisition of Syntactic Knowledge.* Cambridge, MA: MIT Press.

Black, E.; Jelineck, F.; Lafferty, J.; Magerman, D.; Mercer, R.; and Roukos, S. 1993. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, 31–37.

Black, E.; Lafferty, J.; and Roukos, S. 1992. Development and evaluation of a broad-coverage probabilistic grammar of English-language computer manuals. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, 185–192.

Borland International. 1988. *Turbo Prolog 2.0 Reference Guide.* Scotts Valley, CA: Borland International.

Brill, E. 1993. Automatic grammar induction and parsing free text: A transformation-based approach. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, 259–265.

Charniak, E., and Carroll, G. 1994. Context-sensitive statistics for improved grammatical language models. In *Proceedings of the Twelfth National Conference on Artificial Intelligence.*

Church, K., and Patil, R. 1982. Coping with syntactic ambiguity or how to put the block in the box on the table. *American Journal of Computational Linguistics* 8(3-4):139–149.

Collins, M. J. 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, 184–191.

Fillmore, C. J. 1968. The case for case. In Bach, E., and Harms, R. T., eds., *Universals in Linguistic Theory.* New York: Holt, Reinhart and Winston.

Goodman, J. 1996. Parsing algorithms and metrics. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, 177–183.

Kijsirikul, B.; Numao, M.; and Shimura, M. 1992. Discrimination-based constructive induction of logic programs. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 44–49.

Lavrač, N., and Džeroski, S. 1994. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood.

Magerman, D. M. 1994. *Natural Lagnuage Parsing as Statistical Pattern Recognition.* Ph.D. Dissertation, Stanford University.

Magerman, D. M. 1995. Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, 276–283.

Marcus, M.; Santorini, B.; and Marcinkiewicz, M. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics* 19(2):313–330.

Marcus, M. 1980. *A Theory of Syntactic Recognition for Natural Language.* Cambridge, MA: MIT Press.

McClelland, J. L., and Kawamoto, A. H. 1986. Mechanisms of sentence processing: Assigning roles to constituents of sentences. In Rumelhart, D. E., and McClelland, J. L., eds., *Parallel Distributed Processing, Vol. II.* Cambridge, MA: MIT Press. 318–362.

Miikkulainen, R. 1993. *Subsymbolic Natural Language Processing: An Integrated Model of Scripts, Lexicon, and Memory.* Cambridge, MA: MIT Press.

Miikkulainen, R. 1996. Subsymbolic case-role analysis of sentences with embedded clauses. *Cognitive Science* 20(1):47–73.

Miller, S.; Bobrow, R.; Ingria, R.; and Schwartz, R. 1994. Hidden understanding models of natural language. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, 25–32.

Miller, S.; Stallard, D.; Bobrow, R.; and Schwartz, R. 1996. A fully statistical approach to natural language interfaces. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, 55–61.

Muggleton, S., and Buntine, W. 1988. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, 339–352.

Muggleton, S., and Feng, C. 1992. Efficient induction of logic programs. In Muggleton, S., ed., *Inductive Logic Programming.* New York: Academic Press. 281–297.

Muggleton, S. H., ed. 1992. *Inductive Logic Programming.* New York, NY: Academic Press.

Periera, F., and Shabes, Y. 1992. Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, 128–135.

Quinlan, J. 1990. Learning logical definitions from relations. *Machine Learning* 5(3):239–266.

Simmons, R. F., and Yu, Y. 1992. The acquisition and use of context dependent grammars for English. *Computational Linguistics* 18(4):391–418.

St. John, M. F., and McClelland, J. L. 1990. Learning and applying contextual constraints in sentence comprehension. *Artificial Intelligence* 46:217–257.

Tomita, M. 1986. *Efficient Parsing for Natural Language.* Boston: Kluwer Academic Publishers.

Zelle, J. M., and Mooney, R. J. 1993. Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 817–822.

Zelle, J. M., and Mooney, R. J. 1994. Combining top-down and bottom-up methods in inductive logic programming. In *Proceedings of the Eleventh International Conference on Machine Learning*, 343–351.

Zelle, J. M., and Mooney, R. J. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence.*

Zelle, J. M. 1995. *Using Inductive Logic Programming to Automate the Construction of Natural Language Parsers.* Ph.D. Dissertation, University of Texas, Austin, TX. Also appears as Artificial Intelligence Laboratory Technical Report AI 96-249.