Appears inInductive Logic Programming: Selected Papers from the 6th International Workshop
S. Muggleton (Ed.), pp.3-22, Springer Verlag, Berlin, 1997.

Inductive Logic Programming for Natural Language Processing

Raymond J. Mooney

Department of Computer Sciences
University of Texas, Austin TX 78712-1188, USA

Abstract. This paper reviews our recent work on applying inductive logic programming to the construction of natural language processing systems. We have developed a system, Chill, that learns a parser from a training corpus of parsed sentences by inducing heuristics that control an initial overly-general shift-reduce parser. Chill learns syntactic parsers as well as ones that translate English database queries directly into executable logical form. The ATIS corpus of airline information queries was used to test the acquisition of syntactic parsers, and Chill performed competitively with recent statistical methods. English queries to a small database on U.S. geography were used to test the acquisition of a complete natural language interface, and the parser that Chill acquired was more accurate than an existing hand-coded system. The paper also includes a discussion of several issues this work has raised regarding the capabilities and testing of ILP systems as well as a summary of our current research directions.

1 Introduction

Developing a system capable of communicating in natural language is one of the long-standing goals of computing research. Although significant progress has been made in the last forty years (Allen, 1995), developing a natural language processing (NLP) system for a particular application is still an extremely difficult and laborious task. However, a promising approach is to use machine learning techniques to help automate the development of NLP systems.

In recent years, there has been an increasing focus in computational linguistics on *empirical* or *corpus-based* methods that obtain much of their knowledge by training on large corpora of speech or text (Church & Mercer, 1993; Charniak, 1993; Brill & Church, 1996). Almost all of this work has employed statistical techniques such as *n-gram models*, *hidden Markov models* (HMMs), and *probabilistic context free grammars* (PCFGs). The computational linguistics community has focused on these techniques largely due to their successful application in prior work on speech recognition (Waibel & Lee, 1990). There has also been a fair amount of recent research on applying neural-network techniques, such as *simple recurrent networks*, to natural language processing (Reilly & Sharkey, 1992; Miikkulainen, 1993). However, there has been relatively little recent work using symbolic machine learning techniques for language applications, although some recent systems have employed decision trees (Magerman, 1995; Anoe & Bennett, 1995), transformation rules (Brill, 1993, 1995), and other symbolic methods (Wermter, Riloff, & Scheler, 1996).

However, all of these approaches are limited to examples represented as fixed-length feature vectors and are therefore subject to the standard limitations of propositional representations. Language processing, on the other hand, seems to require a very rich knowledge representation language that includes relations, recursion, and unbounded structural representations. Current empirical NLP systems employ carefully-engineered processing architectures and sets of features laboriously constructed by the system developer in order to circumvent these issues. The richness of first-order logic employed in inductive logic programming (ILP) can hopefully provide advantages for NLP applications by increasing flexibility and limiting the amount of feature-engineering required. Despite this fact, other than our own work, there has apparently been no application of ILP methods to language processing, with one early exception (Wirth, 1988, 1989).

Over the last four years, we have explored the application of ILP to NLP. In particular, we have developed and extended the CHILL system for acquiring natural language parsers (Zelle & Mooney, 1993b, 1994b, 1996a, 1996b; Zelle, 1995). [1] This system has learned both syntactic and "semantic" parsers that map a natural language database query directly into an executable Prolog query that will answer the question. Specifically, CHILL uses a training corpus of parsed sentences to induce heuristics that control and specialize an initial overly-general shift-reduce parser. CHILL has learned syntactic parsers for the ATIS corpus of airline information queries, and the results were comparable to current statistical methods. It has also acquired semantic parsers that process and answer English queries about a simple database on U.S. geography. The learned system was more accurate than a hand-built program for this application. The current paper reviews this previous research, attempts to draw some broader implications for ILP, and discusses our directions for future research.

2 Using ILP for Parser Acquisition

The primary task of most natural language systems is parsing. 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 corpus, a case-role (agent, patient, instrument) analysis of a simple sentence, and an executable logical form for a database query about U.S. geography. CHILL is able to learn parsers that produce each of these types of analyses.

[1] These and additional papers, software, and data are available through our web site at `http://www.cs.utexas.edu/users/ml`.

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))))).
```

Fig. 1. Examples of Several Types of Parses

Frequently, language learning has been interpreted as simply acquiring a syntactic recognizer, a unary predicate that simply returns "yes" or "no" to the question: "Is this string a syntactically well-formed sentence in the language." However, such a syntactic recognizer is of limited use to an NLP system, except perhaps a limited grammar checker or a speech recognizer entertaining several word sequences as possible interpretations of an utterance. 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 are useful for further processing. However, most natural language grammars assign multiple parses to sentences, most of which do not correspond to useful, meaningful interpretations. For example, any 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 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 unique 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. If ILP is to be taken as a serious approach to constructing NLP systems, it must be tested on such problems and compared to the existing statistical methods.

2.1 Parser Acquisition by Generic ILP

A straight-forward application of ILP to parser acquisition would be to give a generic ILP system a corpus of sentences paired with representations as a set of positive examples of the predicate `parse(Sentence, Representation)` that takes a sentence as input and produces a syntactic or semantic analysis as output. However, it should be noticed that negative examples of sentence/representation pairs will generally not be available and that using a closed-world assumption to explicitly generate negative examples is intractable given the large space of possible sentences and representations. [2] In addition, it is generally agreed that when children acquire language they are exposed to little if any negative feedback (Bloom, 1994). Consequently, a method is needed for learning without explicit negative tuples. Fortunately, several ILP methods have been proposed for learning from only positive tuples when the target predicate represents a function (Bergadano & Gunetti, 1993; Quinlan, 1996) or when the training data is in some sense complete (De Raedt & Bruynooghe, 1993; Zelle, Thompson, Califf, & Mooney, 1995). If the goal is to construct a parser that produces a unique analysis for each sentence, then the `parse/2` predicate can be treated as a function and any outputs other than the preferred analysis of a training sentence can be treated implicitly as negative examples. If it is desired that the parser produce several preferred outputs for truly ambiguous sentences, a more general assumption of *output completeness* can be used to specify that the analyses provided for each training sentence are the only correct ones and that all other potential outputs are implicitly negative (Zelle et al., 1995; Mooney & Califf, 1995). Using these techniques, a generic ILP system can be used to construct parsers from only positive sentence/representation pairs.

However, it seems unlikely that an uninformed ILP system could produce a program that generalizes well to novel sentences. Parsers are complex programs, the space of possible logic programs is very large, and providing the appropriate set of background predicates is difficult. It is generally agreed that human

[2] Explicit generation of negative examples using a closed-world assumption is performed automatically in many systems such as FOIL (Quinlan, 1990).

language acquisition exploits fairly restrictive constraints or biases in order to learn complex natural languages from limited data (Pinker, 1994). Of course, evaluating the success of this approach is, ultimately, an empirical question. In this paper, we compare the generic approach with a specific alternative, namely CHILL, which acquires parsers by specializing a general parsing architecture by learning control rules.

2.2 Parser Acquisition as Control-Rule Learning

Rather than using ILP techniques to directly learn a complete parser, CHILL begins with a well-defined parsing framework and uses ILP to learn control strategies within this framework. Treating language acquisition as a control learning problem is not in itself a new idea. Berwick (1985) used this approach to learn 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 and 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 above, these control acquisition systems used feature-vector representations. CHILL is the first system to use ILP techniques 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. Figure 2 shows the basic components of the system. CHILL employs a simple deterministic, shift-reduce parser with the current parse state represented by the content of the 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 3. 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.

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 the analyses. For example, an operator to reduce the

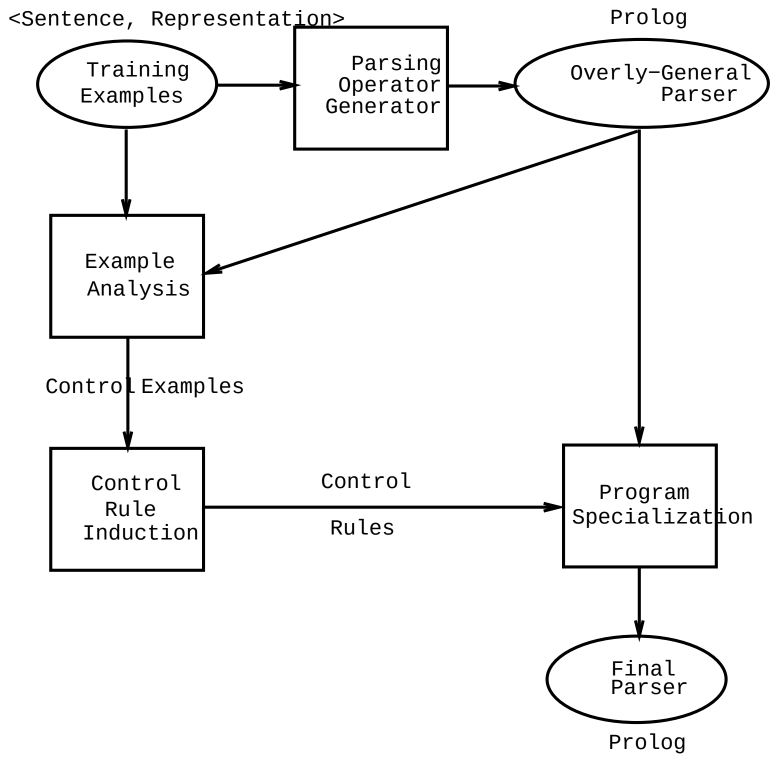

Fig. 2. The CHILL Architecture

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]]]

Fig. 3. Shift-Reduce Case-Role Parsing of "The man ate the pasta."

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 `reduce/4` 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 over-general, 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.

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. These contexts form sets of positive and negative *control examples* from which the appropriate control rules can be subsequently induced. A control example is a "snapshot" of the subgoal to which a particular operator clause may be 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 subgoal to which an operator is applied in this successful parse becomes a positive control example for that operator.

For the `agent` 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)`. This is the only subgoal to which this operator is applied in the correct parsing of the sentence. `A` and `B` are uninstantiated variables since they are outputs from the `op/4` 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 there are additional parse states such as `op([], [the,man,ate,the, pasta], A, B)` that do not appear in this list. This is because the agent clause of `op/4` requires that its first argument be a list containing at least two items. Since the clause cannot match these other subgoals, they will not be included as negative examples.

The Control-Rule Induction phase uses a general ILP system to learn a *control rule* for each operator. This control rule comprises a definite-clause definition that covers the positive control examples for the operator but not the negative. 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 and Mooney (1994a) and Zelle (1995). Given our simple example, a control rule that can be learned for the agent 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. Of course, the new predicate would have a system generated name. It 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 clause of `op/4`. These control rules are then passed on to the Program Specialization phase.

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:

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

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 and only the preferred parses for each of the training examples.

3 Learning Syntactic Parsers for the ATIS Corpus

In section 2.1, we noted that the generic application of ILP to parser acquisition would induce a program directly from the examples of the `parse/2` relation. The advantage gained by the control-rule framework can be assessed by comparing Chill to the performance achieved by Chill's ILP component trying to learn the `parse/2` relation directly. These two approaches were compared by choosing a random set of test examples and learning and evaluating parsers trained on increasingly larger subsets of the remaining examples. Since only positive tuples of `parse/2` are available, the generic ILP approach employed a version of the induction algorithm that exploits the output-completeness assumption to learn in the context of *implicit* negative examples (Zelle et al., 1995) as outlined in section 2.1.

The experiment was carried out using a portion of the ATIS corpus from a preliminary version of the Penn Treebank. The first example in Figure 1 is taken from this corpus. We chose this particular data because it represents realistic input from human-computer interaction, and because it has been used in a number of other studies on automated parser acquisition (Brill, 1993; Periera & Shabes, 1992) that can serve as a basis for comparison to Chill. The corpus contains 729 sentences with an average length of 10.3 words. The experiments reported here were performed using strings of lexical categories rather than words as input. Tagging words with their appropriate part of speech can be performed with high accuracy using various techniques (Church, 1988; Brill, 1995). Zelle and Mooney (1994a) and Zelle (1995) present results both with and without part-of-speech information.

Our initial experiments used a straightforward set of syntactic shift-reduce parsing operators (Zelle & Mooney, 1994b). However, better results were obtained by making the operators more specific, effectively increasing the number of operators, but reducing the complexity of the control-rule induction task for each operator. The basic idea was to index the operators based on some relevant portion of the parsing context. In these experiments, the operators were indexed according to the syntactic category at the front of the input buffer. For example, the general "shift" operator `op(Stack, [Word|Words], [Word|Stack], Words)` becomes multiple operators in slightly differing contexts such as:

```
op(Stack, [det|Ws], [det|Stack], Ws)
op(Stack, [nn|Ws], [nn|Stack], Ws)
op(Stack, [np|Ws], [np|Stack], Ws)
```

The operators in the initial parser were placed in order of increasing frequency of use as indicated by the training set. This allows the learning of control rules to take advantage of "default" effects where specific exceptions are learned first before control falls through to the more generally applicable operators.

Obviously, the most stringent measure of accuracy is the proportion of test sentences for which the produced parse tree exactly matches the human parse for the sentence. Sometimes, however, a parse can be useful even if it is not perfectly accurate; the treebank itself is not completely consistent in the handling of various constructs.

To better gauge the partial accuracy of the parser, we adopted a procedure for returning and scoring partial parses. If the parser runs into a "dead-end" while parsing a novel test sentence, the contents of the stack at the time of impasse is returned as a single, flat constituent labeled `S`. Since the parsing operators are ordered and the shift operator is invariably the most frequently used operator in the training set, shift serves as a sort of default when no reduction action applies. Therefore, at the time of impasse, all of the words of the sentence will be on the stack, and partial constituents will have been built. The contents of the stack therefore reflect the partial progress of the parser in finding constituents.

Partial scoring of trees is based on the overlap between the computed parse and the correct parse as recorded in the treebank. Two constituents are said to match if they span exactly the same words in the sentence. If constituents match and have the same label, then they are identical. The overlap between the computed parse and the correct parse is computed by trying to match each constituent of the computed parse with some constituent in the correct parse. If an identical constituent is found, the score is 1.0, a matching constituent with an incorrect label scores 0.5. The sum of the scores for all constituents is the overlap score for the parse. The accuracy of the parse is then computed as $Accuracy = (\frac{O}{Found} + \frac{O}{Correct})/2$ where O is the overlap score, $Found$ is the number of constituents in the computed parse, and $Correct$ is the number of constituents in the correct tree. The result is an average of the proportion of the computed parse that is correct and the proportion of the correct parse that was actually found.

Another accuracy measure that has been used in evaluating systems that bracket the input sentence into unlabeled constituents, is the proportion of constituents in the parse that do not cross any constituent boundaries in the correct tree (Black & et. al., 1991; Goodman, 1996). We have computed the number of sentences with parses containing no crossing constituents, as well as the proportion of constituents that are non-crossing over all test sentences. This gives a basis of comparison with previous bracketing results, although it should be emphasized that CHILL is designed for the harder task of actually producing labeled parses, and is not optimized for bracketing.

Learning curves averaged over 5 random trials using independent testing sets of 204 sentences are shown in Figure 4. *Correct* is the percentage of test sentences with parses that matched the treebanked parse exactly. *Partial* is partial correct-

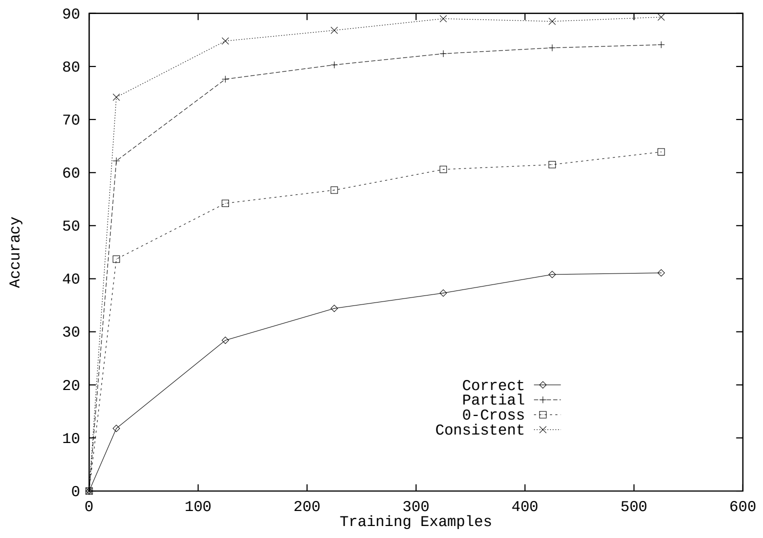

Fig. 4. CHILL ATIS Results

ness using the overlap metric. *0-Cross* is the proportion of test sentences having no constituents that cross constituent boundaries in the correct parsing. Finally, *Consistent* shows the overall percentage of constituents that are consistent with the treebank (i.e. cross no constituents in the correct parse).

The results are quite encouraging. After training on 525 sentences, CHILL constructed completely correct parses for 41% of the novel testing sentences. Using the partial scoring metric, CHILL's parses garnered an average accuracy of over 84%. The figures for 0-cross and consistent compare favorably with those reported in previous studies of automated bracketing for the ATIS corpus. Brill (1993) reports 60% and 91.12%, respectively. CHILL scores higher on the percentage of sentences with no crossing violations (64%) and slightly lower (90%) on the total percentage of non-crossing constituents. This is understandable as Brill's transformation learner tries to optimize the latter value, while CHILL's preference for complete sentence accuracy tends to improve the former.

Figure 5 shows the results for the partial accuracy metric for both CHILL and generic ILP. CHILL has an overwhelming advantage, achieving 84% accuracy compared to the 20% accuracy of generic ILP. Clearly, providing the shift-reduce parsing framework significantly eases the task of the inductive component. Trying to learn a complete parser from scratch is obviously much more difficult.

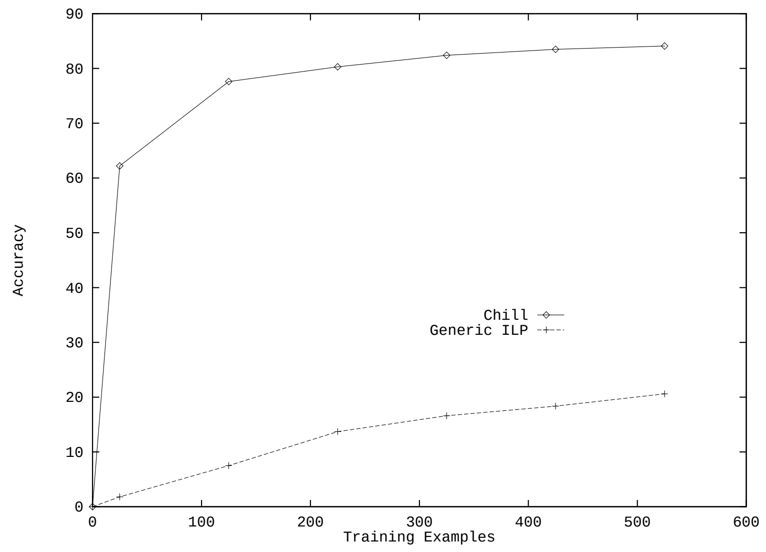

Fig. 5. CHILL vs. Generic ILP on the ATIS Corpus

4 Learning a Complete Natural-Language Interface

The previous experiment demonstrates that ILP techniques as implemented in CHILL can produce results comparable to other empirical approaches for constructing syntactic parsers using a standard treebank. However, syntactic parsing is only a small part of the larger problem of natural language understanding. Consequently, parsers are usually compared on the "artificial" metrics presented above. Unfortunately, it is unclear how well these metrics translate to performance on actual language processing tasks.

As argued in the introduction, one of the major attractions of the ILP approach is its flexibility. The type of representation produced by CHILL's parsers is controlled only by the parsing operators employed. In an effort to assess the utility of CHILL in constructing a complete natural language application, an operator framework was devised that allows the parsing of natural language queries directly into executable Prolog queries. The input to CHILL in this case task consists of sentences paired with executable database queries, where the query language used is a logical form similar to the meaning representation typically produced by logic grammars (Warren & Pereira, 1982; Abramson & Dahl, 1989). The semantics of the representation is grounded in a query interpreter that executes queries and retrieves relevant information from the database.

The chosen database concerns United States geography for which a hand-coded natural-language interface already exists. The system, called *Geobase* was supplied as a sample application with a commercial Prolog, specifically 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. Figure 6 shows some sample questions and associated query representations in addition to the example already presented in Figure 1.

What is the highest point of the state with the largest area?
```
answer(P, (high-point(S,P), largest(A, (state(S), area(S,A))))).
```
What are the major cities in Kansas?
```
answer(C, (major(C), city(C), loc(C,S), equal(S,stateid(kansas)))).
```

Fig. 6. Sample Database Queries

The 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. Experiments were then performed by training on subsets of the corpus and evaluating the resulting parser on the unseen examples. The parser was judged to have parsed a new sentence correctly when the generated query produced exactly the same final answer from the database as 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.

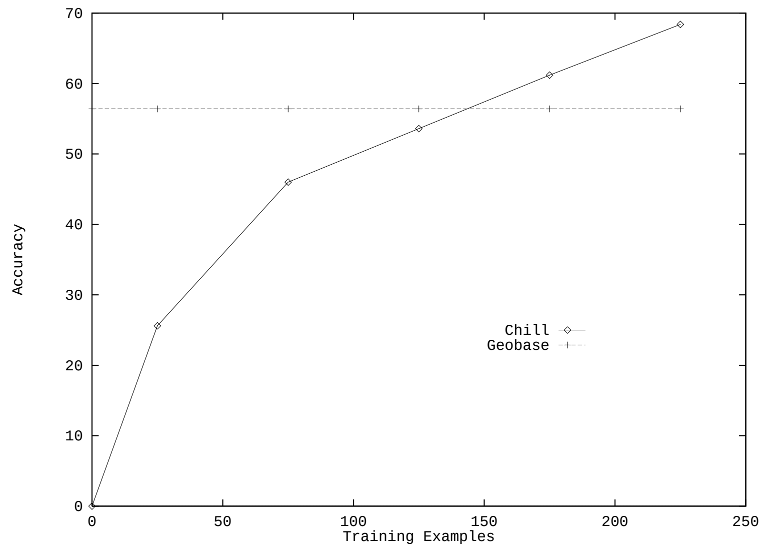

Fig. 7. Chill Accuracy on Database Query Domain

Figure 7 shows the accuracy of Chill's parsers over a 10 trial average. The line labeled "Geobase" shows the average accuracy of the hand-coded system. The curves show that Chill outperforms the existing system when trained on 175 or more examples. In the best trial, Chill induced a parser from 225 examples comprising 1100 lines of Prolog code in approximately 25 minutes on a SPARCstation 5 and achieved 84% accuracy in answering novel queries. 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 questions on average are actually answered incorrectly. Zelle and Mooney (1996b) and Zelle (1995) provide additional details on the Geobase application and results.

5 Lessons and Challenges for ILP

Applying ILP to natural language processing has highlighted several broader issues in developing and evaluating ILP systems. Natural language problems present a number of interesting challenges for ILP systems, many of which may have counterparts in other complex applications. In addition to parser acquisition, we have also applied ILP to natural language morphology, specifically to generating the past tense of English verbs. English past-tense generation has become a benchmark problem in the computational modeling of human language acquisition (Rumelhart & McClelland, 1986; Ling & Marinov, 1993). Mooney and Califf (1995) showed that a particular ILP system, FOIDL, could learn this transformation more effectively than previous neural-network and decision-tree methods. This section discusses several issues in ILP that our work on CHILL and FOIDL have uncovered.

5.1 Learning to Control Existing Programs

Inducing a complex logic program completely from examples is a difficult task. The standard way of easing the problem has been to supply an ILP system with relevant background knowledge (subroutines) and induce only the top-level clauses (Lavrač & Džeroski, 1994). Another approach has been to revise an existing program that is partially correct (De Raedt, 1992; Richards & Mooney, 1995). CHILL illustrates a third approach: specializing an existing program by learning control rules that restrict the application of specific clauses.

Induction of control rules has a fairly long history in learning and problem solving (Mitchell, 1983; Langley, 1985) and more recent work has applied ILP to this task (Cohen, 1990; Leckie & Zuckerman, 1993; Zelle & Mooney, 1993a; Estlin & Mooney, 1996). These systems focus on learning control rules that improve the *efficiency* of an existing program, such as transforming an $O(n!)$ naive sorting program into an $O(n^2)$ insertion sort (Zelle & Mooney, 1993a). CHILL illustrates that this approach can also be used to improve the *accuracy* of an initial (extremely) overly-general program. Other problems may also lend themselves to providing or constructing an initial overly-general program that can be appropriately specialized by inducing control rules.

5.2 Improving Generic ILP Systems

Initial attempts to apply existing ILP systems such as FOIL and GOLEM to parser construction and past-tense generation met with important difficulties. Limitations such as requiring extensional background and negative examples, lack of predicate invention, inability to handle functions or cuts (!), and search limitations (e.g. local minima, combinatorial explosions) prevented existing systems from performing well or even being applicable to these problems. Consequently, we had to develop new ILP systems such as CHILLIN (Zelle & Mooney, 1994a) and FOIDL (Mooney & Califf, 1995) to overcome these limitations by using techniques that integrate bottom-up and top-down search, incorporate predicate invention, eliminate the need for explicit negative examples, and allow restricted use of cuts. Existing techniques had to be improved and integrated in order to build ILP systems that could handle natural language problems.

Consequently, there is still an need for flexible, robust, efficient ILP systems that incorporate a range of abilities and features. Generic ILP systems are still unable to handle many large, complex problems such as those that arise in NLP. Statistical language learning systems have been trained on real corpora of up to

40,000 sentences (Magerman, 1995; Collins, 1996). Current ILP techniques are incapable of handling such large problems.

5.3 Training and Testing for Programs that Generate Output

Most ILP systems have been tested on their accuracy of classifying ground tuples as positive or negative examples of the target predicate. However, many applications such as parsing and morphological analysis require computing outputs rather than testing ground tuples. In these applications, ILP systems need to be tested on their ability to generate correct outputs from novel inputs.

With respect to training, an ILP system needs to guarantee that it will generate a program that will terminate and generate ground outputs when it is queried with the outputs uninstantiated. Most ILP systems cannot provide these guarantees; those that guarantee termination (Quinlan, 1990) do not guarantee ground outputs. The use of an output completeness assumption and implicit negatives is one way to guarantee ground outputs (Zelle et al., 1995).

With respect to testing, experiments need to specifically evaluate the ability of the learned program to generate correct outputs given only novel inputs. Unlike evaluations of other ILP systems, experiments with CHILL and FOIDL specifically tested this ability. Also, in many applications, exactly matching the output specified in the test data may not be the best measure of performance. Induced programs may generate complex outputs that are more or less similar to the "correct" output (as with parse trees) or there may be multiple correct outputs that are semantically equivalent (as with database queries). Therefore, one may want to measure various types of partial correctness of outputs, such as the number of bracketing errors for parse trees, or use some other procedure for judging the correctness of the output, such as whether it produces the same answer from a database as the "correct" output. In general, appropriate testing of logic programs generated for complex applications may require measuring something other than their accuracy at classifying ground tuples.

6 Ongoing Research

Our current research concerns using learning techniques such as ILP to develop a larger natural language application. We hope to field an application on the world-wide-web that will attract a significant number of users and therefore serve as an automatic source of larger amounts of language data. The specific application we are considering is a system that can process the computer job announcements posted to the USENET 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?"

This application will involve using learning techniques to build two major components. The first is an *information extraction* system that processes individual messages and extracts specific pieces of information for the database such as the type of job, the location, the salary, the starting date, etc.. Such natural-language information extraction systems have been hand-built as part of ARPA's MUC (Message Understanding Conference) program (Lehnert & Sundheim, 1991; ARPA, 1993) and several projects have used learning techniques to automatically acquire rules for this task (Riloff, 1993; Soderland & Lehnert, 1994; Huffman, 1996). We plan to develop a system that uses ideas from ILP to learn patterns for extracting information from newsgroup postings. Examples of messages paired with filled templates will be used to train the system, and the learned rules will then be used to extract a database of information from the newsgroup postings.

The second major component is a query system for answering natural-language questions about the database built by the information extraction module. Chill will be used to learn this component by training on sample pairs of English/Prolog job queries in the same manner used to construct the geography database interface discussed in section 4. After building a prototype system from an initial training set, we plan to put it on line and collect additional query examples. Questions that the system cannot parse will 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.

7 Conclusions

Constructing natural language systems is a complex task, and machine learning is becoming an increasingly important tool in aiding their development. This paper has summarized research on employing inductive logic programming to learn natural language parsers, and presented results illustrating that such methods can successfully learn syntactic parsers as well as complete natural language interfaces. In addition, the ILP-constructed systems were shown to perform as well as if not better than existing hand-built and statistically-trained systems.

Unfortunately, current learning research in computational linguistics is focused on alternative statistical methods. Convincing computational linguists of the utility of ILP for constructing NLP systems will not be an easy task. However, by clearly demonstrating the ability of ILP systems to easily and flexibly build real systems from large amounts of real language data without laborious feature engineering, a convincing case for ILP can be made. The research reviewed in this paper is a first step in this direction, and will hopefully encourage and assist additional research in the area.

Acknowledgments Much thanks to John Zelle, Chill's primary developer. This research was supported by the National Science Foundation under grant IRI-9310819.

References

Abramson, H., & Dahl, V. (1989). *Logic Grammars*. Springer-Verlag, New York.

Allen, J. F. (1995). *Natural Language Understanding (2nd Ed.).* Benjamin/Cummings, Menlo Park, CA.

Anoe, C., & Bennett, S. W. (1995). Evaluating automated and manual acquisition of anaphora resolution strategies. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pp. 122–129 Cambridge, MA.

ARPA (Ed.). (1993). *Proceedings of the Fifth DARPA Message Understanding Evaluation and Conference*, San Mateo, CA. Morgan Kaufman.

Bergadano, F., & Gunetti, D. (1993). An interactive system to learn functional logic programs. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp. 1044–1049 Chambery, France.

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

Black, E., & et. al. (1991). A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of the Fourth DARPA Speech and Natural Language Workshop*, pp. 306–311.

Bloom, P. (1994). Overview: Controversies in language acquisition. In Bloom, P. (Ed.), *Language Acquisition: Core Readings*, pp. 5–48. MIT Press, Cambridge, MA.

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

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*, pp. 259–265 Columbus, Ohio.

Brill, E. (1995). Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, *21*(4), 543–565.

Brill, E., & Church, K. (Eds.). (1996). *Proceedings of the Conference on Empirical Methods in Natural Language Processing.* University of Pennsylvania, Philadelphia, PA.

Charniak, E. (1993). *Statistical Language Learning.* MIT Press.

Church, K. (1988). A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the Second Conference on Applied Natural Language Processing.* Association for Computational Linguistics.

Church, K., & Mercer, R. L. (1993). Introduction to the special issue on computational linguistics using large corpora. *Computational Linguistics*, *19*(1), 1–24.

Church, K., & 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.

Cohen, W. W. (1990). Learning approximate control rules of high utility. In *Proceedings of the Seventh International Conference on Machine Learning*, pp. 268–276 Austin, TX.

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*, pp. 184–191 Santa Cruz, CA.

De Raedt, L. (1992). *Interactive Theory Revision: An Inductive Logic Programming Approach.* Academic Press, New York, NY.

De Raedt, L., & Bruynooghe, M. (1993). A theory of clausal discovery. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp. 1058–1063 Chambery, France.

Estlin, T. A., & Mooney, R. J. (1996). Multi-strategy learning of search control for partial-order planning. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence* Portland, OR.

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

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

Huffman, S. B. (1996). Learning information extraction patterns from examples. In Wermter, S., Riloff, E., & Scheler, G. (Eds.), *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, pp. 246–260. Springer, Berlin.

Kijsirikul, B., Numao, M., & Shimura, M. (1992). Discrimination-based constructive induction of logic programs. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 44–49 San Jose, CA.

Langley, P. (1985). Learning to search: From weak methods to domain specific heuristics. *Cognitive Science*, *9*(2), 217–260.

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

Leckie, C., & Zuckerman, I. (1993). An inductive approach to learning search control rules for planning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp. 1100–1105 Chamberry,France.

Lehnert, W., & Sundheim, B. (1991). A performance evaluation of text-analysis technologies. *AI Magazine*, *12*(3), 81–94.

Ling, C. X., & Marinov, M. (1993). Answering the connectionist challenge: A symbolic model of learning the past tense of English verbs. *Cognition*, *49*(3), 235–290.

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

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

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

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

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

Mitchell, T. (1983). Learning and problem solving. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pp. 1139–1151 Karlsruhe, West Germany.

Mooney, R. J., & Califf, M. E. (1995). Induction of first-order decision lists: Results on learning the past tense of English verbs. *Journal of Artificial Intelligence Research*, *3*, 1–24.

Muggleton, S., & Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pp. 339–352 Ann Arbor, MI.

Muggleton, S., & Feng, C. (1992). Efficient induction of logic programs. In Muggleton, S. (Ed.), *Inductive Logic Programming*, pp. 281–297. Academic Press, New York.

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

Pinker, S. (Ed.). (1994). *The Language Instinct: How the Mind Creates Language.* William Morrow, N.Y.

Quinlan, J. R. (1996). Learning first-order definitions of functions. *Journal of Artificial Intelligence Research*, *to appear*.

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

Reilly, R. G., & Sharkey, N. E. (Eds.). (1992). *Connectionist Approaches to Natural Language Processing.* Lawrence Erlbaum and Associates, Hilldale, NJ.

Richards, B. L., & Mooney, R. J. (1995). Automated refinement of first-order Horn-clause domain theories. *Machine Learning*, *19*(2), 95–131.

Riloff, E. (1993). Automatically constructing a dictionary for information extraction tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp. 811–816.

Rumelhart, D. E., & McClelland, J. (1986). On learning the past tense of English verbs. In Rumelhart, D. E., & McClelland, J. L. (Eds.), *Parallel Distributed Processing, Vol. II*, pp. 216–271. MIT Press, Cambridge, MA.

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

Soderland, S., & Lehnert, W. (1994). Wrap-Up: A trainable discourse module for information extraction. *Journal of Artificial Intelligence Research*, *2*, 131–158.

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

Waibel, A., & Lee, K. F. (Eds.). (1990). *Readings in Speech Recognition.* Morgan Kaufmann, San Mateo,CA.

Warren, D., & Pereira, F. (1982). An efficient easily adaptable system for interpreting natural language queries. *American Journal of Computational Linguistics*, *8*(3-4), 110–122.

Wermter, S., Riloff, E., & Scheler, G. (Eds.). (1996). *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing.* Springer Verlag, Berlin.

Wirth, R. (1988). Learning by failure to prove. In *Proceedings of the Third European Working Session on Learning*, pp. 237–251. Pitman.

Wirth, R. (1989). Completing logic programs by inverse resolution. In *Proceedings of the Fourth European Working Session on Learning*, pp. 239–250. Pitman.

Zelle, J. M. (1995). *Using Inductive Logic Programming to Automate the Construction of Natural Language Parsers.* Ph.D. thesis, University of Texas, Austin, TX.

Zelle, J. M., & Mooney, R. J. (1993a). Combining FOIL and EBG to speed-up logic programs. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp. 1106–1111 Chambery, France.

Zelle, J. M., & Mooney, R. J. (1993b). Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp. 817–822 Washington, D.C.

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

Zelle, J. M., & Mooney, R. J. (1994b). Inducing deterministic Prolog parsers from treebanks: A machine learning approach. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pp. 748–753 Seattle, WA.

Zelle, J. M., & Mooney, R. J. (1996a). Comparative results on using inductive logic programming for corpus-based parser construction. In Wermter, S., Riloff, E., & Scheler, G. (Eds.), *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, pp. 355–369. Springer, Berlin.

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

Zelle, J. M., Thompson, C., Califf, M. E., & Mooney, R. J. (1995). Inducing logic programs without explicit negative examples. In *Proceedings of the Fifth International Workshop on Inductive Logic Programming*, pp. 403–416 Leuven, Belgium.

This article was processed using the LaTeX macro package with LLNCS style