Appears in *Symbolic, Connectionist, and Statistical Approaches to Learning for Natural Language Processing,* Springer Verlag, 1996

Comparative Results on Using Inductive Logic Programming for Corpus-based Parser Construction

John M. Zelle[1], and Raymond J. Mooney[2]

[1] Department of Mathematics and Computer Science, Drake University, Des Moines IA 50311, USA (jz6011r@dunix.drake.edu)
[2] Department of Computer Sciences, University of Texas, Austin TX 78712, USA (mooney@cs.utexas.edu)

Abstract. This paper presents results from recent experiments with CHILL, a corpus-based parser acquisition system. CHILL treats language acquisition as the learning of search-control rules within a logic program. Unlike many current corpus-based approaches that use statistical learning algorithms, CHILL uses techniques from inductive logic programming (ILP) to learn relational representations. CHILL is a very flexible system and has been used to learn parsers that produce syntactic parse trees, case-role analyses, and executable database queries. The reported experiments compare CHILL's performance to that of a more naive application of ILP to parser acquisition. The results show that ILP techniques, as employed in CHILL, are a viable alternative to statistical methods and that the control-rule framework is fundamental to CHILL's success.

1 Introduction

Empirical or *corpus-based* methods for constructing natural language systems has been an area of growing research interest in the last several years. The empirical approach replaces hand-generated rules with models obtained automatically by training over language corpora. Corpus-based methods may be used to augment the knowledge of a traditional parser, for example by acquiring new case-frames for verbs [13] or acquiring models to resolve lexical or attachment ambiguities [11, 8]. More radical approaches attempt to replace the hand-crafted components altogether, constructing complete parsers directly from suitable corpora. Recent approaches to constructing robust parsers from corpora primarily use statistical and probabilistic methods such as stochastic grammars [4, 23, 7] or transition networks [17]. These methods eschew traditional, symbolic parsing in favor of statistical and probabilistic methods. Although several current methods learn some symbolic structures such as decision trees [3, 12] and transformations [6], statistical methods dominate.

A common thread in all of these approaches is that the acquired knowledge is represented in a *propositional* form (perhaps with associated probabilities). This means for example, a decision about how to label a node in a parse tree is made by considering a fixed set of properties (e.g., syntactic category) about a fixed context of surrounding nodes (e.g., parent and immediate left sibling). The exact conditions of the rule(s) are determined by the acquisition algorithm but the context over which the rules are formed, and the exact properties which may be tested are determined *a priori* by the designer of the acquisition system. In machine learning, such approaches are often called feature-vector representations, as each decision context can be specified by a finite vector of atomic values associated with the various features of interest.

In contrast, the CHILL system [28, 30, 27] uses a framework for learning with structured representations. Relational representations have long been a tool of traditional NLP. Virtually all of this work has utilized hand-crafted grammars as suitable methods for automating the construction of relational knowledge bases had not yet been developed. Now, however, a growing subfield of machine learning research called Inductive Logic Programming (ILP) addresses the problem of learning first-order logic descriptions (Prolog programs). Due to the expressiveness of first-order logic, ILP methods can learn relational and recursive concepts that cannot be represented in the featural languages assumed by most machine-learning algorithms. ILP methods have successfully induced small programs for sorting and list manipulation [24] as well as produced encouraging results on important applications such as predicting protein secondary structure [21]. Our research seeks to apply ILP methods to NLP in a effort to bridge the gap between traditional NLP and empirical approaches.

A major advantage using ILP techniques is the resulting flexibility. Given the power of first-order rules, there is less need to hand-engineer appropriate features and contexts over which the system learns. The induction algorithm can automatically extract the relevant portions of structured contexts and construct new predicates to represent novel syntactic and/or semantic word and phrase categories that are necessary to perform accurate parsing. Furthermore, the combination of traditional and empirical approaches allows the insights of traditional parsing to provide structure or *bias* to the learning component. Stronger biases potentially allow better results from smaller training corpora. Finally, the approach is easily adapted to learn parsers for various types of analyses. CHILL has been used to learn parsers that produce syntactic parse trees, case-role analyses, and actual executable database queries.

This paper considers two methods of applying ILP to the problem of corpus-based parser construction. We present a naive approach based on the direct induction of logic programs to perform parsing, and a more sophisticated system, CHILL, which applies ILP within the context of a fixed parsing framework. Experimental results demonstrate that the bias provided by the parsing framework significantly improves the accuracy of the resulting parsers.

2 Using ILP for Parser Construction

2.1 Introduction to ILP

ILP research considers the problem of inducing a first-order, definite-clause logic program from a set of examples and given background knowledge[10, 22]. As such, it stands at the intersection of the traditional fields of machine learning and logic programming.

As an example ILP task, consider learning the concept of list membership. The input to the learning system consists of a number of positive and negative instances of the predicate, `member/2`.[3] Some positive instances might be: `member(1, [1,2])`, `member(2, [1,2])`, `member(1,[3,1])`. Instances such as `member(1,[])`, `member(2,[1,3])`, would serve as negative examples. Additional information is provided in the form background relations in terms of which the desired concept is to be learned. In the case of list membership, this information might include a definition of the concept, `components/3` which decomposes a list into its component head and tail. This type of "constructor" predicate is typically used as many ILP systems learn function-free clauses; using `components/3` eliminates the need for list constructions (e.g. `[X|Y]`) within learned clauses. Given this input, an ILP system attempts to construct a concept definition which entails the positive training examples, but not the negatives. In this case, we hope to learn the correct definition of `member`, namely:

member(X, List) :- components(List, X, Tail).
member(X, List) :- components(List, Head, Tail), member(X, Tail).

2.2 "Naive" Parser Construction

A straight-forward application of ILP to parser construction would be to simply present a corpus of sentences paired with representations as a set of positive examples to an ILP system. For example, we might try to learn a definition of the concept `parse(Sentence,Rep)`. The induced logic program, might then be used to prove goals having the second argument uninstantiated, effectively producing parses of sentences provided as input. There are a number of difficulties with this approach.

First, it is not clear what the other inputs to the ILP system should be. We do not have a convenient set of negative examples or a good theory of what the background relations should be. Both of these are critical to the success of current ILP algorithms. Clearly, it is intractable to generate all possible sentences paired with incorrect analyses as a set of negative examples. Even selecting a manageable–sized random subset of these negative examples is unlikely to be sufficient, as such a sample is unlikely to include the many "near-miss" examples which are crucial to learning good generalizations. Rather than attempting to generate a set of explicit negative examples, we have developed a technique of quantifying *implicit* negative examples that effectively overcomes this difficulty [31]. The problem of providing appropriate background knowledge has been largely unexplored; in our experiments, we relied on the ability of the ILP algorithm to invent suitable background relations.

[3] We use the standard notation ⟨**name**⟩/⟨**number**⟩ to indicate the name and arity (number of arguments) for a predicate.

Even given these other inputs, it seems unlikely that an uninformed ILP system could produce a program which generalizes well to new inputs. The space of logic programs is too large, and the learning problem too unconstrained to hope that the results would be useful. In short, the "inductive leap" is simply too great. Evaluating the success of the naive approach is, ultimately, an empirical question. In this paper we compare the naive technique with a more sophisticated alternative, namely CHILL, which addresses these issues by considering language acquisition as a control-rule learning problem.

2.3 Acquisition as Control Rule Learning

Rather than using ILP techniques to directly learn a logic grammar, CHILL begins with a well-defined parsing framework and uses ILP to learn control strategies within this framework. Treating language acquisition as a control-rule learning problem is not in itself a new idea. Berwick [1] used this approach to learn rules for a Marcus-style deterministic parser. 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 [26] 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 corresponding to the best matching previous context. Like the statistical approaches mentioned above, these early control-rule acquisition systems used pre-defined 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 which maps sentences into parses. Figure 1 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. Parsing operators are program clauses which 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 which are required to produce the analyses. For example, when producing case-role analyses, an operator to reduce the top two items on the stack by attaching the second item as an agent of the first is represented by the clause `op([Top,Second|Rest],In,[NewTop|Rest],In) :- reduce(Top,agt,Second,NewTop)` where the arguments to `op/4` are, in order, the current stack, current input buffer, new stack, and new input buffer. 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 attachment-role 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.

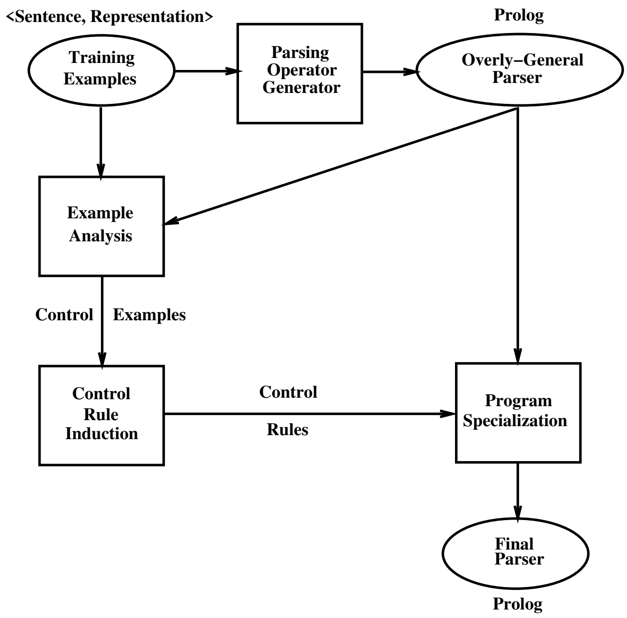

Fig. 1. The CHILL Architecture

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) which 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 first-order induction algorithm 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. There is a growing body of research in inductive logic programming which addresses this problem. CHILL combines elements from bottom-up techniques found in systems such as CIGOL [19] and GOLEM [20] and top-down methods from systems like FOIL [25], and is able to invent new predicates in a manner analogous to CHAMP [9]. Details of the CHILL induction algorithm can be found in [28, 29, 27]. Given our simple example, a control rule that might 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.

3 CHILL vs. Naive ILP

3.1 Experimental Method

In Section 2.2, we noted that the naive application of ILP to parser acquisition would be to 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 using the standard machine learning paradigm of first choosing a random set of test examples and then learning and evaluating parsers using increasingly larger subsets of the remaining examples.

Since the naive approach requires positive and negative examples of the `parse/2` concept, we employed a version of the induction algorithm which exploits the output-completeness assumption to learn in the context of *implicit* negative examples [31, 18]. A complete discussion of this technique is beyond the scope of this paper, but the intuition is straightforward. A developing program is evaluated by using it to construct all possible parses that it can generate from a given training sentence. Since all of the correct parses are supplied for each sentence, any generated outputs which are not present in the training set are considered to be negative examples. Likewise, outputs which are not complete (i.e. contain uninstantiated variables) are considered to cover many negative examples, since there could be a large number of incorrect representations that they could match.

3.2 Results for Case-Role Mapping

In one experiment, CHILL and naive ILP were compared on an artificial data set for case-role mapping that has been used to demonstrate certain language processing abilities of artificial neural networks [16]. This task involves a corpus of 1475 sentence/case-structure pairs originally presented in [15]. The corpus was produced from a set of 19 sentence templates, generating sentences such as "The HUMAN ate the FOOD with the UTENSIL", where the capitalized items are replaced with words of the given category. The sample actually comprises 1390 unique sentences, some of which allow multiple analyses. Since our parser is capable (through backtracking) of generating all legal parses for an input, training was done considering each unique sentence as a single example, and insuring that the training corpus contained all correct parses.

Our results reflect averages over five trials using different testing sets of 740 sentences each. During testing, the parser was used to enumerate all analyses for a given test sentence. Parsing of a sentence can fail in two ways: an incorrect analysis may be generated, or a correct analysis may not be generated. In order to account for both types of inaccuracy, a metric was introduced to calculate the "average correctness" for a given test sentence as follows: $Accuracy = (\frac{C}{P} + \frac{C}{A})/2$ where P is the number of distinct analyses produced, C is the number of the produced analyses which were correct, and A is the number of correct analyses possible for the sentence. This measure can be viewed as an average of the parser's precision and recall for a given sentence.

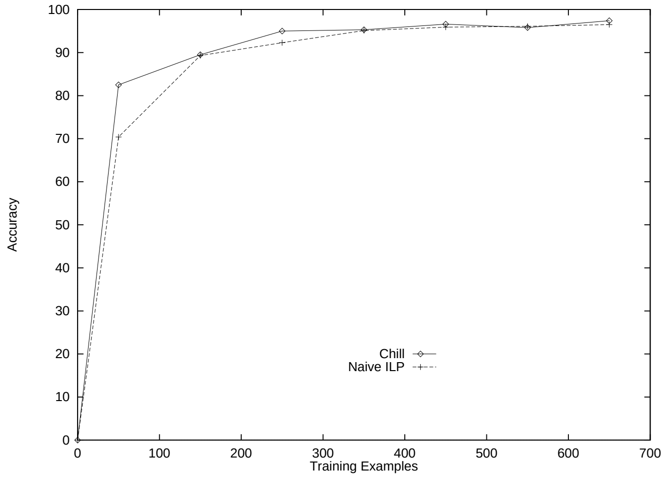

Fig. 2. CHILL vs. Naive ILP on Artificial Case-role Task

Figure 2 shows the average learning curves of these two systems. Both are able to learn this simple task very accurately, significantly outperforming the previous neural network approaches [16]. The results show that the naive approach performs only slightly worse than CHILL for small examples sets, and becomes indistinguishable as the sample grows. Inspection of the resulting programs showed that the induction algorithm was fairly accurately recreating the template-and-filler style of program which was used to generate this corpus. This provides strong evidence of the power of the ILP induction algorithm for inducing programs from examples exhibiting regular structure. The next subsection considers a more realistic task.

3.3 Parsing the Penn Tree-bank

Applying CHILL: A second set of experiments was carried out using a portion of the ATIS dataset from a preliminary version of the Penn Tree-bank (specifically, the sentences in the file `ti_tb`)[14]. 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 grammar acquisition [6, 23] 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 "tagged" strings of lexical categories as input rather than words.

The learning component of CHILL remained exactly the same as in the case-role experiments except that the initial Parsing Operator Generator was modified to produce operators appropriate for the syntactic analyses of the Penn Tree-bank. As in case-role parsing, building an overly-general parser from a set of training examples is accomplished by constructing clauses for the `op` predicate. For example, an operator to reduce a prepositional phrase might look like: `op([S1,S2|Ss], Words, [pp:[S2,S1]|Ss], Words)`.

Our initial experiments used this simple representation of parsing actions [30]. 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. As an 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, [np|Ws], [np|Stack], Ws)
```

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

CHILL **Results:** Obviously, the most stringent measure of accuracy is the proportion of test sentences for which the produced parse tree exactly matches the tree-banked parse for the sentence. Sometimes, however, a parse can be useful even if it is not perfectly accurate; the tree-bank itself is not completely consistent in the handling of various structures.

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 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 stack reflect the partial progress of the parser in finding constituents.

Partial scoring of trees is computed by determining the extent of overlap between the computed parse and the correct parse as recorded in the tree-bank. 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, which 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 [2]. We have computed the number of sentences with parses containing no crossing constituents, as well as the proportion of constituents which 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.

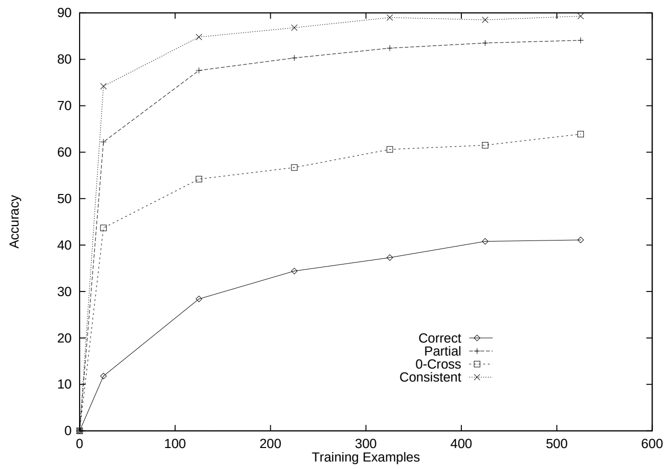

Fig. 3. CHILL ATIS Results

The average learning curves over 5 trials using independent testing sets of 204 sentences are shown in Figure 3. *Correct* is the percentage of test sentences with parses that matched the tree-banked parse exactly. *Partial* is partial correctness 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 tree-bank (i.e. cross no constituents in the correct parse).

The results are very 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 very favorably with those reported in 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 sentence accuracy might tend to improve the former (since correctly parsed sentences are consistent).

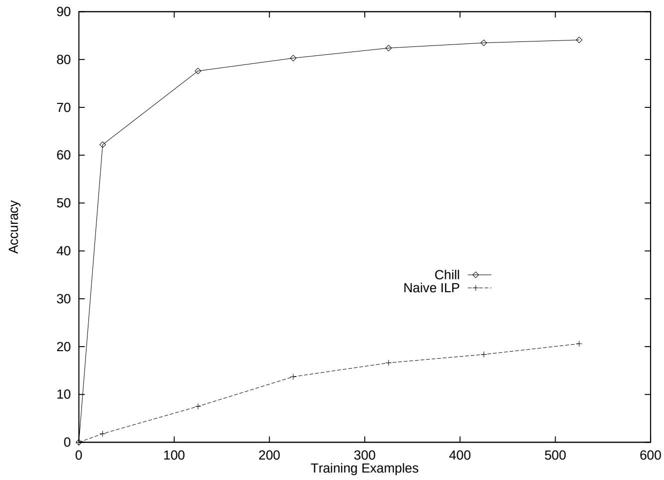

Fig. 4. CHILL vs. Naive ILP on ATIS Corpus

Comparison: On this less-structured, real-world corpora, the advantage of the control-rule framework over the naive approach becomes readily apparent. Figure 4 shows the results for the partial accuracy metric of the two systems in the ATIS experiment. Here CHILL has an overwhelming advantage, achieving 84% accuracy compared to the 20% accuracy of induction with implicit negatives. Clearly, providing the shift-reduce parsing framework significantly eases the task of the inductive component.

4 An Application: Parsing Database Queries

The foregoing experiments demonstrate that ILP techniques as implemented in CHILL can produce results that are comparable to, or better than, previous empirical approaches for constructing syntactic parsers on certain corpora. One of the shortcomings of these experiments is that the target representations were either syntactic or relatively shallow semantic structures. Parsing to this level of representation is only a small part of the larger problem of natural language understanding. Because of this, parsing systems are usually compared on the types of artificial metrics presented here. Unfortunately, it is not clear how well these metrics translate to performance on actual language processing tasks.

As we argued in the introduction, one of the major attractions of an approach based on first-order learning is its flexibility. The type of representation learned by CHILL is controlled only by the type of parsing operators employed. We have already shown that changing the parsing operators allows CHILL to learn parsers for either case-role assignments or syntactic parse-trees. However, CHILL is not restricted to learning syntactic representations. In an effort to assess the utility of CHILL in constructing a complete natural language application, a third operator framework was devised that allows the parsing of logic-based database queries.

What is the capital of the state with the largest population?
```
answer(C, (capital(S,C), largest(P, (state(S), population(S,P))))).
```

What are the major cities in Kansas?
```
answer(C, (major(C), city(C), loc(C,S), equal(S,stateid(kansas)))).
```

Fig. 5. Sample Database Queries

In the db-query task, CHILL is presented with a training set consisting of sentences paired with executable database queries. CHILL then learns a parser which maps subsequent sentences directly into queries with no other intermediate representation. Experiments have been performed using a database on U.S. geography. Figure 5 shows a sample of the type of queries employed. The data for these experiments was gathered by asking uninformed subjects to generate sample questions for the system. An analyst then paired the questions with approriate queries to generate an experimental corpus. 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 result as the query provided by the analyst. Hence, the metric is a true measure of performance in a complete database-query application.

Figure 6 shows the accuracy of CHILL's parsers over a 10 trial average. The line labeled "Geobase" shows the average accuracy of the Geobase system, a natural-language front-end supplied as an example application with a commercial Prolog system (Turbo Prolog 2.0 [5]). The curves show that CHILL outperforms the existing system when trained on 175 or more examples. In the best trial, CHILL's induced parser comprising 1100 lines of Prolog code achieved 84% accuracy in answering novel queries.

5 Future Work

Obviously, there is room for improvement in the results reported here. Improving the accuracy of the resulting parsers requires progress on two fronts. One way

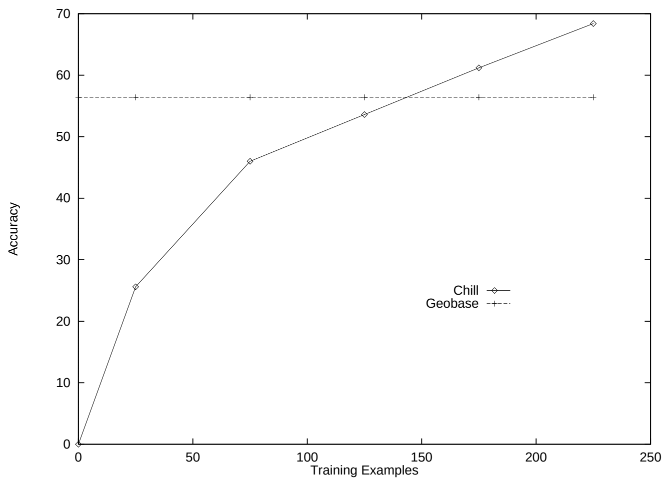

Fig. 6. CHILL Accuracy on Database Query Domain

to improve accuracy is to use larger training sets. Making this practical requires further improvements in the efficiency of the induction algorithm. Another avenue to improvement is the incorporation of more language-specific biases into the induction process. The shift-reduce framework of the current system is a rather weak bias compared to the types of restrictions which might be found in a "principles and parameters" based approach. A tighter integration of linguistic insights with ILP methods could probably create more efficient learning systems for language tasks.

We are also investigating the extension of these methods to deal with a broader range of NLP issues. On the lexical side, we are investigating new methods of parsing operator generation. These operators will allow the creation of deeper semantic representations that replace words with semantic tokens and infer information not explicitly given in the sentence. We also believe that ILP techniques might be usefully applied in learning larger discourse structures and information extraction tasks. We view CHILL as a mere starting point in the investigation of the usefulness of relational learning techniques for NLP in general.

6 Conclusions

In this paper we have argued that ILP techniques offer a more flexible approach to learning in natural language systems than do feature-vector representations. Experimental results with CHILL show that ILP techniques perform as well or better than propositional approaches on some comparable tasks. One of the major attractions of ILP is the ease with which it may be integrated with traditional, symbolic parsing methods. Indeed, the experiments demonstrate that the traditional shift-reduce framework used by CHILL is fundamental to CHILL's success in learning realistic language processing tasks.

Acknowledgments This research was supported by the National Science Foundation under grant IRI-9310819 and the Texas Advanced Research Program under grant ARP-003658-114.

References

1. B. Berwick. *The Acquisition of Syntactic Knowledge*. MIT Press, Cambridge, MA, 1985.
2. E. Black and et. al. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of the Fourth DARPA Speech and Natural Language Workshop*, pages 306–311, 1991.
3. E. Black, F. Jelineck, J. Lafferty, D. Magerman, R. Mercer, and S. Roukos. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 31–37, Columbus, Ohio, 1993.
4. E. Black, J. Lafferty, and S. Roukaos. 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*, pages 185–192, Newark, Delaware, 1992.
5. Borland International. *Turbo Prolog 2.0 Reference Guide*. Borland International, Scotts Valley, CA, 1988.
6. E. Brill. Automatic grammar induction and parsing free text: A transformation-based approach. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 259–265, Columbus, Ohio, 1993.
7. Eugene Charniak and Glenn Carroll. Context-sensitive statistics for improved grammatical language models. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA, August 1994.
8. D. Hindle and M. Rooth. Structural ambiguity and lexical relations. *Computational Linguistics*, 19(1):103–120, 1993.
9. B. Kijsirikul, M. Numao, and M. Shimura. Discrimination-based constructive induction of logic programs. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 44–49, San Jose, CA, July 1992.
10. N. Lavrač and S. Džeroski, editors. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.
11. Jill Fain Lehman. Toward the essential nature of satistical knowledge in sense resolution. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA, August 1994.
12. David M. Magerman. *Natrual Lagnuage Parsing as Statistical Pattern Recognition*. PhD thesis, Stanford University, 1994.
13. Christopher D. Manning. Automatic acquisition of a large subcategorization dictionary from corpora. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 235–242, Columbus, Ohio, 1993.
14. M. Marcus, B. Santorini, and M.A. Marcinkiewicz. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2):313–330, 1993.
15. J. L. McClelland and A. H. Kawamoto. Mechanisms of sentence processing: Assigning roles to constituents of sentences. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing, Vol. II*, pages 318–362. MIT Press, Cambridge, MA, 1986.

16. R. Miikkulainen and M. G. Dyer. Natural language processing with modular PDP networks and distributed lexicon. *Cognitive Science*, 15:343–399, 1991.
17. Scott Miller, Robert Bobrow, Robert Ingria, and Richard Schwartz. Hidden understanding models of natural language. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, pages 25–32, 1994.
18. R. J. Mooney and M. E. Califf. Induction of first-order decision lists: Results on learning the past tense of English verbs. *Journal of Artificial Intelligence Research*, 3:1–24, 1995.
19. S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 339–352, Ann Arbor, MI, June 1988.
20. S. Muggleton and C. Feng. Efficient induction of logic programs. In S. Muggleton, editor, *Inductive Logic Programming*, pages 281–297. Academic Press, New York, 1992.
21. S. Muggleton, R. King, and M. Sternberg. Protein secondary structure prediction using logic-based machine learning. *Protein Engineering*, 5(7):647–657, 1992.
22. S. H. Muggleton, editor. *Inductive Logic Programming*. Academic Press, New York, NY, 1992.
23. F. Periera and Y. Shabes. Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, pages 128–135, Newark, Delaware, 1992.
24. J. R. Quinlan and R. M. Cameron-Jones. FOIL: A midterm report. In *Proceedings of the European Conference on Machine Learning*, pages 3–20, Vienna, 1993.
25. J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.
26. R. F. Simmons and Y. Yu. The acquisition and use of context dependent grammars for English. *Computational Linguistics*, 18(4):391–418, 1992.
27. J. M. Zelle. *Using Inductive Logic Programming to Automate the Construction of Natural Language Parsers*. PhD thesis, University of Texas, Austin, TX, August 1995.
28. J. M. Zelle and R. J. Mooney. Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 817–822, Washington, D.C., July 1993.
29. J. M. Zelle and R. J. Mooney. Combining top-down and bottom-up methods in inductive logic programming. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 343–351, New Brunswick, NJ, July 1994.
30. J. M. Zelle and R. J. Mooney. Inducing deterministic Prolog parsers from treebanks: A machine learning approach. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 748–753, Seattle, WA, August 1994.
31. John M. Zelle, Cynthia A. Thompson, Mary Elaine Califf, and Raymond J. Mooney. Inducing logic programs without explicit negative examples. In *Proceedings of the Fifth International Workshop on Inductive Logic Programming*, 1995.