Appears in *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*
pp. 1050-1055, Portland, OR, August 1996

Learning to Parse Database Queries Using Inductive Logic Programming

John M. Zelle
Department of Mathematics and Computer Science
Drake University
Des Moines, IA 50311
jz6011r@acad.drake.edu

Raymond J. Mooney
Department of Computer Sciences
University of Texas
Austin, TX 78712
mooney@cs.utexas.edu

Abstract

This paper presents recent work using the Chill parser acquisition system to automate the construction of a natural-language interface for database queries. Chill treats parser acquisition as the learning of search-control rules within a logic program representing a shift-reduce parser and uses techniques from Inductive Logic Programming to learn relational control knowledge. Starting with a general framework for constructing a suitable logical form, Chill is able to train on a corpus comprising sentences paired with database queries and induce parsers that map subsequent sentences directly into executable queries. Experimental results with a complete database-query application for U.S. geography show that Chill is able to learn parsers that outperform a pre-existing, hand-crafted counterpart. These results demonstrate the ability of a corpus-based system to produce more than purely syntactic representations. They also provide direct evidence of the utility of an empirical approach at the level of a complete natural language application.

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. Recent approaches to constructing robust parsers from corpora primarily use statistical and probabilistic methods such as stochastic grammars (Black, Lafferty, & Roukaos 1992; Periera & Shabes 1992; Charniak & Carroll 1994) or transition networks (Miller *et al.* 1994). Several current methods learn some symbolic structures such as decision trees (Black *et al.* 1993; Magerman 1994; Kuhn & De Mori 1995) and transformations (Brill 1993). Zelle and Mooney (1993, 1994) have proposed a method called Chill based on the relational learning techniques of Inductive Logic Programming.

To date, these systems have been demonstrated primarily on the problem of syntactic parsing, grouping the words of a sentence into hierarchical constituent structure. Since syntactic analysis is only a small part of the overall problem of understanding, these approaches have been trained on corpora that are "artificially" annotated with syntactic information. Similarly, they are typically evaluated with artificial metrics of parsing accuracy. While such metrics can provide rough comparisons of relative capabilities, it is not clear to what extent these measures reflect differences in performance on real language-processing tasks. The acid test for empirical approaches is whether they allow the construction of better natural language systems, or perhaps allow for the construction of comparable systems with less overall effort. This paper reports on the experience of using Chill to engineer a natural language front–end for a database–query task.

A database-query task was a natural choice as it represents a significant real-world language-processing problem that has long been a touch-stone in NLP research. It is also a nontrivial problem of tractable size and scope for actually carrying out evaluations of empirical approaches. Finally, and perhaps most importantly, a parser for database queries is easily evaluable. The bottom line is whether the system produces a correct answer for a given question, a determination which is straight-forward for many database domains.

Learning to Parse DB queries

Overview of CHILL

Space does not permit a complete description of the Chill system here. The relevant details may be found in (Zelle & Mooney 1993; 1994; Zelle 1995). What follows is a brief overview.

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 that maps sentences into parses. Chill treats parser induction as a problem of learning rules to control the actions of a shift-reduce parser expressed as a Prolog program. Control-rules are expressed as definite-clause concept definitions. These rules are induced using a general concept learning system employing techniques from Inductive Logic Programming (ILP) a subfield of machine learning that addresses the problem of learning definite-clause logic descriptions from examples (Lavrač & Džeroski 1994; Muggleton 1992).

The central insight in CHILL is that the general operators required for a shift-reduce parser to produce a given set of sentence analyses are directly inferable from the representations themselves. For example, if the target is syntactic analysis, the fact the the parser requires a reduction to combine a determiner and a noun to form an NP follows directly from the existence of such an NP in the training examples. However, just inferring an appropriate set of operators does not produce a correct parser, because more knowledge is required to apply operators accurately during the course of parsing an example.

The current context of a parse is contained in the contents of the stack and the remaining input buffer. CHILL uses parses of the training examples to figure out the contexts in which each of the inferred operators is and is not applicable. These contexts are then given to a general induction algorithm that learns rules to classify the contexts in which each operator should be used. Since the contexts are arbitrarily-complex parser-states involving nested (partial) constituents, CHILL employs an ILP learning algorithm which can deal with structured inputs and produce relational concept descriptions.

Figure 1 shows the basic components of CHILL. During Parser Operator Generation, the training examples are analyzed to formulate an overly-general shift-reduce parser that is capable of producing parses from sentences. The initial parser is overly-general in that it produces a great many spurious analyses for any given input sentence. In Example Analysis, the training examples are parsed using the overly-general parser to extract contexts in which the various parsing operators should and should not be employed. Control-Rule Induction then employs a general ILP algorithm to learn rules that characterize these contexts. Finally, Program Specialization "folds" the learned control-rules back into the overly-general parser to produce the final parser.

Previous experiments have evaluated CHILL's performance in learning parsers to perform case-role parsing (Zelle & Mooney 1993) and syntactic parsers for portions of the ATIS corpus (Zelle & Mooney 1994; 1996). These experiments have demonstrated that CHILL works as well or better than neural-network or statistical approaches on comparable corpora.

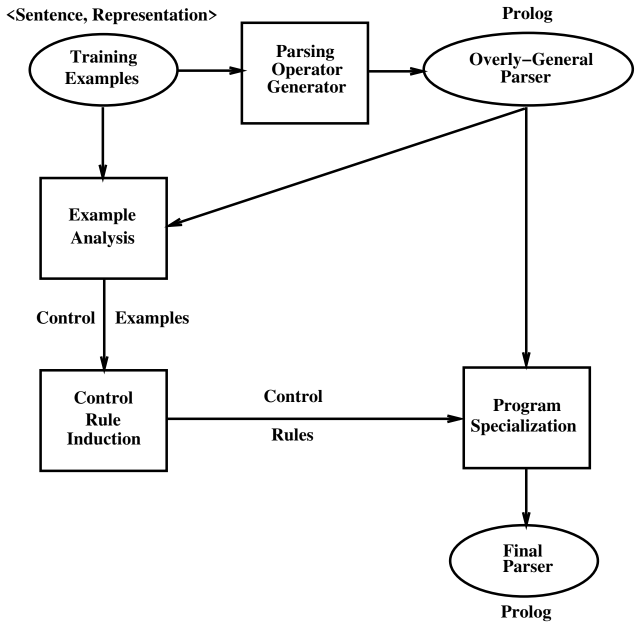

Figure 1: The CHILL Architecture

Parsing DB Queries

Overview of the Problem For the database-query task, the input to CHILL consists of sentences paired with executable database queries. The query language considered here is a logical form similar to the types of 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 choice of a logical query language rather than the more ubiquitous SQL was made because the former provides a more straight-forward, compositional mapping from natural language utterances—a property that is necessary for the CHILL approach. The process of translating from an unambiguous logical form into other query formats is easily automated.

The domain of the chosen database is United States geography. The choice was motivated by the availability of an existing natural language interface for a simple geography database. This system, called *Geobase* was supplied as an example application with a commercial Prolog available for PCs, specifically Turbo Prolog 2.0 (Borland International 1988). Having such an example provides a database already coded in Prolog for which a front-end can be built; it also serves as a convenient benchmark against which CHILL's performance can be compared.

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)))).`

Figure 2: Sample Database Queries

Type	Form	Example
country	`countryid(Name)`	`countryid(usa)`
city	`cityid(Name, State)`	`cityid(austin,tx)`
state	`stateid(Name)`	`stateid(texas)`
river	`riverid(Name)`	`riverid(colorado)`
place	`placeid(Name)`	`placeid(pacific)`

Figure 3: Basic Objects in Geoquery

The Geobase data 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 2 shows some sample questions and associated query representations.

Development of the database application required work on two components: a framework for parsing into the logical query representations, and a specific query language for the geography database. The first component is domain-independent and consists of algorithms for parsing operator generation and example analysis to infer the required operators and parse the training examples. The resulting parsing framework is quite general and could be used to generate parsers for a wide range of logic-based representations.

The second component, which is domain specific, is a query language having a vocabulary sufficient for expressing interesting questions about geography. The database application itself comprises a parser produced by CHILL coupled with an interpreter for the query language. The specific query language for these experiments (hereafter referred to as *Geoquery*) was initially developed by considering a sample of 50 sentences. A simple query interpreter was developed concurrently with the query language, thus insuring that the representations were grounded in the database-query task.

The Query Language, Geoquery The query language considered here is basically a first-order logical form augmented with some higher-order predicates or *meta-predicates*, for handling issues such as quantification over implicit sets. This general form of representation is useful for many language processing tasks. The particular constructs of Geoquery, however, were not designed around any notion of appropriateness for representation of natural language in general, but rather as a direct method of compositionally translating English sentences into unambiguous, logic-oriented database queries.

The most basic constructs of the query representation are the terms used to represent the objects referenced in the database and the basic relations between them. The basic forms are listed in Figure 3. The objects of interest are states, cities, rivers and places (either a high-point of low-point of a state). Cities are represented using a two argument term with the second argument containing the abbreviation of the state. This is done to insure uniqueness, since different states may have cities of the same name (e.g. `cityid(columbus,oh)` vs. `cityid(columbus,ga)`). This convention also allows a natural form for expressing partial information; a city known only by name is given an uninstantiated variable for its second term.

Form	Predicate
`capital(C)`	`C` is a capital (city).
`city(C)`	`C` is a city.
`major(X)`	`X` is major.
`place(P)`	`P` is a place.
`river(R)`	`R` is a river.
`state(S)`	`S` is a state.
`capital(C)`	`C` is a capital (city).
`area(S,A)`	The area of `S` is `A`.
`capital(S,C)`	The capital of `S` is `C`.
`equal(V,C)`	variable `V` is ground term `C`.
`density(S,D)`	The (population) density of `S` is `P`
`elevation(P,E)`	The elevation of `P` is `E`.
`high_point(S,P)`	The highest point of `S` is `P`.
`higher(P1,P2)`	`P1`'s elevation is greater than `P2`'s.
`loc(X,Y)`	`X` is located in `Y`.
`low_point(S,P)`	The lowest point of `S` is `P`.
`len(R,L)`	The length of `R` is `L`.
`next_to(S1,S2)`	`S1` is next to `S2`.
`size(X,Y)`	The size of `X` is `Y`.
`traverse(R,S)`	`R` traverses `S`.

Figure 4: Basic Predicates in Geoquery

The basic relations are shown in Figure 4. The *equal/2* predicate is used to indicate that a certain variable is bound to a ground term representing an object in the database. For example, a phrase like "the capital of Texas" translates to `(capital(S,C), equal(S, stateid(texas)))` rather than the more traditional `capital(stateid(texas),C)`. The use of `equal` allows objects to be introduced at the point where they are actually named in the sentence.

Although the basic predicates provide most of the expressiveness of Geoquery, meta-predicates are required to form complete queries. A list of the implemented meta-predicates is shown in Figure 5. These predicates

Form	Explanation
`answer(V,Goal)`	`V` is the variable of interest in `Goal`.
`largest(V, Goal)`	`Goal` produces only the solution that maximizes the size of `V`
`smallest(V,Goal)`	Analogous to `largest`.
`highest(V,Goal)`	Like `largest` (with elevation).
`lowest(V,Goal)`	Analogous to `highest`.
`longest(V,Goal)`	Like `largest` (with length).
`shortest(V,Goal)`	Analogous to `longest`.
`count(D,Goal,C)`	`C` is count of unique bindings for `D` that satisfy `Goal`.
`most(X,D,Goal)`	`Goal` produces only the `X` that maximizes the count of `D`
`fewest(X,D,Goal)`	Analogous to `most`.

Figure 5: Meta-Predicates in Geoquery

are distinguished in that they take completely-formed conjunctive goals as one of their arguments. The most important of the meta-predicates is `answer/2`. This predicate serves as a "wrapper" for query goals indicating the variable whose binding is of interest (i.e. answers the question posed). The other meta-predicates provide for the quantification over and selection of extremal elements from implicit sets.

A Parsing Framework for Queries Although the logical representations of Geoquery look very different from parse-trees or case-structures on which CHILL has been previously demonstrated, they are amenable to the same general parsing scheme as that used for the shallower representations. Adapting CHILL to work with this representation requires only the identification and implementation of suitable operators for the construction of Geoquery-style analyses.

The parser is implemented by translating parsing actions into operator clauses for a shift-reduce parser. The construction of logical queries involves three different types of operators. Initially, a word or phrase at the front of the input buffer suggests that a certain structure should be part of the result. The appropriate structure is pushed onto the stack. For example, the word "capital" might cause the `capital/2` predicate to be pushed on the stack. This type of operation is performed by an `introduce` operator. Initially, such structures are introduced with new (not co-referenced) variables. These variables may be unified with variables appearing in other stack items through a `co-reference` operator. For example, the first argument of the `capital/2` structure may be unified with the argument of a previously introduced `state/1` predicate. Finally, a stack item may be embedded into the argument of another stack item to form conjunctive goals inside of meta-predicates; this is performed by a `conjoin` operation.

For each class of operator, the overly-general operators required to parse any given example may be easily inferred. The necessary `introduce` operators are determined by examining what structures occur in the given query and which words that can introduce those structures appear in the training sentence. `Co-reference` operators are constructed by finding the shared variables in the training queries; each sharing requires an appropriate operator instance. Finally, `conjoin` operations are indicated by the term-embedding exhibited in the training examples. It is important to note that only the operator generation phase of CHILL is modified to work with this representation; the control-rule learning component remains unchanged.

As an example of operator generation, the first query in Figure 2 gives rise to four `introduce` operators: "capital" introduces `capital/2`, "state" introduces `state/1`, "largest" introduces `largest/2` and "population" introduces `population/2`. The initial parser-state has `answer/2` on the stack, so its introduction is not required. The example generates four `co-reference` operators for the variables (e.g., when `capital/2` is on the top of the stack, its second argument may be unified with the first argument of `answer/2`, which is below it). Finally, the example produces four `conjoin` operators. When `largest/2` is on the top of the stack, `state/1` is "lifted" into the second argument position from its position below in the stack. Conversely, when `population/2` is on the top of the stack, it is "dropped" into the second argument of `largest/2` to form the conjunction. Similar operators embed `capital/2` and `largest/2` into the conjunction that is the second argument of `answer/2`.

Experimental Results

Experiments

A corpus of 250 sentences was gathered by submitting a questionnaire to 50 uninformed subjects. For evaluation purposes, the corpus was split into training sets of 225 examples with the remaining 25 held-out for testing. CHILL was run using default values for various parameters.

Testing employed the most stringent standard for accuracy, namely whether the application produced the correct answer to a question. Each test sentence was parsed to produce a query. This query was then executed to extract an answer from the database. The extracted answer was then compared to the answer produced by the correct query associated with the test sentence. Identical answers were scored as a correct parsing, any discrepancy resulted in a failure. Figure 6 shows the average accuracy of CHILL's parsers over 10

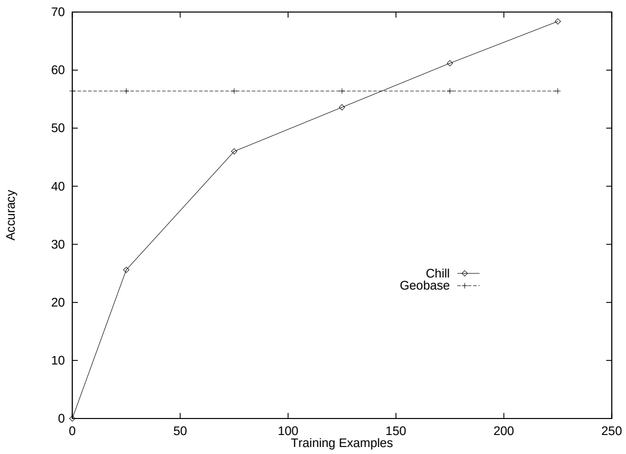

Figure 6: Geoquery: Accuracy

trials using different random splits of training and testing data. The line labeled "Geobase" shows the average accuracy of the Geobase system on these 10 testing sets of 25 sentences. 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.

In this application, it is important to distinguish between two modes of failure. The system could either fail to parse a sentence entirely, or it could produce a query which retrieves an incorrect answer. The parsers learned by CHILL for Geoquery produced few spurious parses. At 175 training examples, CHILL produced 3.2% spurious parses, dropping to 2.3% at 200 examples. This compares favorably with the 3.8% rate for Geobase.

Discussion

These results are interesting in two ways. First, they show the ability of CHILL to learn parsers that map sentences into queries without intermediate syntactic parsing or annotation. This is an important consideration for empirical systems that seek to reduce the linguistic expertise needed to construct NLP applications. Annotating corpora with useful *final* representations is a much easier task than providing detailed linguistic annotations. One can even imagine the construction of suitable corpora occurring as a natural side-effect of attempting to automate processes that are currently done manually (e.g. collecting examples of the queries produced by database users in the normal course of their work).

Second, the results demonstrate the utility of an empirical approach at the level of a complete natural-language application. While the Geobase system probably does not represent a state-of-the-art standard for natural language database query systems, neither is it a "straw man." Geobase uses a semantics-based parser which scans for words corresponding to the entities and relationships encoded in the database. Rather than relying on extensive syntactic analysis, the system attempts to match sequences of entities and associations in sentences with an entity-association network describing the schemas present in the database. The result is a relatively robust parser, since many words can simply be ignored. That CHILL performs better after training on a relatively small corpus is an encouraging result.

Related Work

As noted in the introduction, most work on corpus-based parsing has focused on the problem of syntactic analysis rather than semantic interpretation. However, a number of groups participating in the ARPA-sponsored ATIS benchmark for speech understanding have used learned rules to perform some semantic interpretation. The Chronus system from AT&T (Pieraccini *et al.* 1992) used an approach based on stochastic grammars. Another approach employing statistical techniques is the Hidden Understanding Models of Miller, et. al. (1994). Kuhn and De Mori (1995) have investigated an approach utilizing semantic classification trees, a variation on decision trees familiar in machine learning.

These approaches differ from work reported here in that learning was used in only a one component of a larger hand-crafted grammar. The ATIS benchmark is not an ideal setting for the evaluation of empirical components *per se*, as overall performance may be significantly affected by the performance of other components in the system. Additionally, the hand-crafted portions of these systems encompassed elements that were part of the learning task for CHILL. CHILL learns to map from strings of words directly into query representations without any intermediate analysis; thus, it essentially automates construction of virtually the entire linguistic component. We also believe that CHILL's relational learning algorithms make the approach more flexible, as evidenced by the range of representations for which CHILL has successfully learned parsers. Objective comparison of various approaches to empirical NLP is an important area for future research.

Future Work and Conclusions

Clearly, there are many open questions regarding the practicality of using CHILL for the development of NLP systems. Experiments with larger corpora and other domains are indicated. Another interesting avenue of investigation is the extent to which performance can be improved by corpus "manufacturing." Since an initial corpus must be annotated by hand, one method of increasing the regularity in the training corpus (and hence the generality of the resulting parser) would be to allow the annotator to introduce related sentences. Although this approach would require extra effort from the annotator, it would be far easier than annotating an equal number of random sentences and might produce better results.

The development of automated techniques for lexicon construction could also broaden the applicability of CHILL. Currently, the generation of `introduce` operators relies on a hand-built lexicon indicating which words can introduce various predicates. Thompson (1995) has demonstrated an initial approach to corpus-based acquisition of lexical mapping rules suitable for use with CHILL-style parser acquisition systems.

We have described a framework using ILP to learn parsers that map sentences into database queries using a training corpus of sentences paired with queries. This method has been implemented in the CHILL system, which treats parser acquisition as the learning of search-control rules within a logic program representing a shift-reduce parser. Experimental results with a complete application for answering questions about U.S. geography show that CHILL's parsers outperform a pre-existing hand-crafted counterpart. These results demonstrate CHILL's ability to learn semantic mappings and the utility of an empirical approach at the level of a complete natural-language application. We hope these experiments will stimulate further research in corpus-based techniques that employ ILP.

Acknowledgments

Portions of this research were supported by the National Science Foundation under grant IRI-9310819.

References

Abramson, H., and Dahl, V. 1989. *Logic Grammars.* New York: Springer-Verlag.

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

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

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

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

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

Kuhn, R., and De Mori, R. 1995. The application of semantic classification trees to natural language understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17(5):449–460.

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

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

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

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

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

Pieraccini, R.; Tzoukermann, E.; Z. Gorelov, J. L. G.; Levin, E.; Lee, C. H.; and Wilpon, J. 1992. A speech understanding system based on statistical representation of semantics. In *Proceedings ICASSP 92.* I–193–I–196.

Thompson, C. A. 1995. Acquisition of a lexicon from semantic representations of sentences. In *Proceeding of the 33rd Annual Meeting of the Association for Computational Linguistics*, 335–337.

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

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

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

Zelle, J., and Mooney, R. 1996. Comparative results on using inductive logic programming for corpus-based parser construction. In Wermter, S.; Riloff, E.; and Scheler, G., eds., *Symbolic, Connectionist, and Statistical Approaches to Learning for Natural Language Processing.* Springer Verlag.

Zelle, J. M. 1995. *Using Inductive Logic Programming to Automate the Construction of Natural Language Parsers.* Ph.D. Dissertation, University of Texas, Austin, TX. available via http://cs.utexas.edu/users/ml.