Learning to Transform Natural to Formal Languages

Rohit J. Kate Yuk Wah Wong Raymond J. Mooney
Department of Computer Sciences
The University of Texas at Austin
1 University Station C0500
Austin, TX 78712-0233, USA
{rjkate,ywwong,mooney}@cs.utexas.edu

Abstract

This paper presents a method for inducing transformation rules that map natural-language sentences into a formal query or command language. The approach assumes a formal grammar for the target representation language and learns transformation rules that exploit the non-terminal symbols in this grammar. The learned transformation rules incrementally map a natural-language sentence or its syntactic parse tree into a parse-tree for the target formal language. Experimental results are presented for two corpora, one which maps English instructions into an existing formal coaching language for simulated RoboCup soccer agents, and another which maps English U.S.-geography questions into a database query language. We show that our method performs overall better and faster than previous approaches in both domains.

Introduction

The ability to map natural language to a formal query or command language is critical to developing more user-friendly interfaces to many computing systems (e.g. databases (Woods 1977)). However, relatively little research in empirical natural-language processing (NLP) has addressed the problem of learning such semantic parsers from corpora of sentences paired with their formal-language equivalents. Most recent work in corpus-based semantic parsing has focused on shallow thematic (case-role) analysis (Gildea & Jurafsky 2002). By learning to transform natural language (NL) to a complete formal language, NL interfaces to complex computing and AI systems can be more easily developed.

In this paper, we consider two formal languages for performing useful, complex tasks. The first one is a database query language as used in a previously-developed corpus of U.S. geography questions (Zelle & Mooney 1996). The second formal language is a coaching language for robotic soccer developed for the RoboCup Coach Competition, in which AI researchers compete to provide effective instructions to a coachable team of agents in a simulated soccer domain (Chen *et al.* 2003).

The few previous systems for directly learning such NL interfaces have employed complex and slow inductive-logic-programming (ILP) methods to acquire parsers for mapping sentences to a logical query language (Zelle & Mooney 1996; Tang & Mooney 2001). In this paper, we introduce a simpler, more-efficient approach based on learning string-to-tree or tree-to-tree transformation rules. The approach is shown to be more effective in the RoboCup domain. It assumes that a deterministically-parsable grammar for the target formal language is available. Transformation rules are learned that map substrings in NL sentences or subtrees in their corresponding syntactic parse trees to subtrees of the formal-language parse tree. The non-terminal symbols in the formal-language grammar provide convenient intermediate representations that enable the construction of general, effective transformation rules.

Our approach has been implemented in a system called SILT (Semantic Interpretation by Learning Transformations). One version of the system assumes that a syntactic parse-tree for the NL sentence is provided by an existing parser, the other version maps directly from the original NL string. Using an assembled corpus of 300 RoboCup coaching instructions and an existing corpus of 250 geography database queries, we present experimental results evaluating SILT's ability to efficiently learn accurate NL interfaces and compare it to previously developed ILP approaches.

Target Formal Languages

We restrict our formal languages to those with deterministic context-free grammars (a.k.a. LR grammars), so that a given formal representation always has a unique parse tree that is used when learning to transform from natural language. Almost all computer languages fall into this category, including the following two that motivate our work.

CLANG: the RoboCup Coach Language

RoboCup (www.robocup.org) is an international AI research initiative using robotic soccer as its primary domain. In the Coach Competition, teams of agents compete on a simulated soccer field and receive advice from a team coach in a formal language called CLANG. In CLANG, tactics and behaviors are expressed in terms of if-then rules. As described in (Chen *et al.* 2003), its grammar consists of 37 non-terminal symbols and 133 productions.

We augmented CLANG with additional expression types for several concepts that are easily expressible in natural language but not in CLANG. An example is *all players*

except player 4, which in CLANG has to be written as `{1 2 3 5 6 7 8 9 10 11}`. These new expression types are automatically converted into the original language. Below is a sample rule with its English translation:

```
((bpos (penalty-area our))
 (do (player-except our {4})
     (pos (half our))))
```

"If the ball is in our penalty area, all our players except player 4 should stay in our half."

The semantic parsers we have developed for CLANG are part of a larger research project on advice-taking reinforcement-learning agents that accept advice stated in natural language (Kuhlmann *et al.* 2004).

GEOQUERY: a Database Query Application

GEOQUERY is a logical query language for a small database of U.S. geography containing about 800 facts. This domain was originally chosen to test corpus-based semantic parsing due to the availability of a hand-built natural-language interface, GEOBASE, supplied with Turbo Prolog 2.0 (Borland International 1988). The GEOQUERY language consists of Prolog queries augmented with several meta-predicates (Zelle & Mooney 1996). Below is a sample query with its English translation:

```
answer(A,count(B,(city(B),loc(B,C),
      const(C,countryid(usa))),A))
```

"How many cities are there in the US?"

Semantic Parsing using Transformations

SILT is a new approach to mapping NL sentences into their formal representations using transformation rules. Although transformation rules have been used in other NLP tasks such as part-of-speech tagging (Brill 1995), our approach differs substantially from Brill's method. In SILT, transformation rules associate patterns found in natural language with templates that are based on productions of the formal-language grammar. The pattern of a rule is matched against phrases in the sentence, and when successfully matched, the corresponding template is instantiated to create part of the formal representation.

Throughout the paper, we use the phrase *formal grammar* to refer to the grammar of the target formal language. *Productions* and *non-terminals* always refer to those in the formal grammar. A *rule* is always a transformation rule that maps NL phrases to formal expressions.

Rule Representation

Every transformation rule in SILT associates a pattern with a template. We have developed two versions of SILT based on the representation of their patterns. The first version (*string-based*) uses strings of words as patterns and matches them directly against natural language sentences. The second version (*tree-based*) uses trees (words and syntactic markers) as patterns and matches them against syntactic parse-trees of the NL sentences. For both versions, the rule templates are based on the productions of the formal grammar.

A sample rule with a string-based pattern and a template based on a CLANG production is:

*"*TEAM UNUM *has the ball"*
CONDITION → `(bowner` TEAM `{`UNUM`})`

Here TEAM, UNUM (uniform number) and CONDITION are non-terminals of the CLANG grammar. When the pattern is matched, an instance of the associated template is introduced as part of the formal representation. The next subsection illustrates this process with a running example.

Below is a sample tree-based pattern in prefix notation, associated with the same template:

(S (NP TEAM UNUM) (VP (VBZ *has*) (NP (DT *the*) (NN *ball*))))

Here S, NP, NN *etc.* in the pattern are syntactic markers. Notice that in a tree-based pattern, CLANG non-terminals are always at the leaves of the tree.

Example of Semantic Parsing

SILT builds the formal representation by repeatedly applying transformation rules to the given NL sentence. We illustrate the bottom-up version of this process with an example. This example uses string-based transformation rules, but it can be easily extended to tree-based rules. Consider the sentence:

"If our player 4 has the ball, our player 4 should shoot."

First, the following rules whose patterns match the sentence are applied. Notice that none of these patterns contain CLANG non-terminals.

"our"	TEAM → `our`
"player 4"	UNUM → `4`
"shoot"	ACTION → `(shoot)`

The matched portions of the sentence are replaced by the left-hand-side (LHS) non-terminals of the associated rule templates, namely TEAM, UNUM and ACTION. These non-terminals are treated the same as words in the sentence, and each is associated with a formal representation instantiated from the right hand side (RHS) of the rule template, as shown below by the downward arrows. These representations are later used to assemble the complete formal representation of the sentence.

"If TEAM UNUM *has the ball,* TEAM UNUM *should*
`our` `4` `our` `4`
ACTION *."*
`(shoot)`

Next, the rule shown in the previous subsection is applied since its pattern *"*TEAM UNUM *has the ball"* now matches the sentence. The matched portion of the sentence is replaced with the LHS non-terminal of the template, CONDITION. In turn, this non-terminal is associated with the representation `(bowner our {4})`, obtained by instantiating the RHS of the template, `(bowner` TEAM `{`UNUM`})`, substituting the representations associated with the matched TEAM and UNUM non-terminals. The result is:

"*If* CONDITION , TEAM UNUM *should*
(bowner our {4}) our 4
ACTION ."
(shoot)

This rule illustrates the constraint that a pattern must contain the same set of non-terminal symbols as the RHS of the associated template. The matched non-terminals in the sentence supply the arguments needed to instantiate the template. Next, the rule:

"TEAM UNUM *should* ACTION"
DIRECTIVE → (do TEAM{UNUM} ACTION)

applies and the sentence becomes:

"*If* CONDITION , DIRECTIVE ."
(bowner our{4}) (do our{4}(shoot))

Finally, the sentence matches the rule:

"*if* CONDITION , DIRECTIVE ."
RULE → (CONDITION DIRECTIVE)

and a complete formal representation is obtained:

" RULE "
((bowner our{4})(do our{4}(shoot)))

The parsing process can also proceed in a top-down fashion, by reversing the order in which rules are applied.

Variations of Rule Representation

To increase the coverage of a transformation rule, SILT allows patterns that skip some number of words (or tree nodes). The skipped words (or nodes) are called a *gap*, and the maximum gap size is specified in the pattern. For example, the pattern "*if* CONDITION , ⟨1⟩ DIRECTIVE ." allows at most one word to be skipped between "," and "DIRECTIVE", and allows matching the sentence "*If* CONDITION, *then* DIRECTIVE", skipping the word "*then*".

SILT's rule representation also allows for constraints on the context surrounding the pattern to be transformed. For example, a rule that transforms "*in* REGION" to CONDITION → (bpos REGION)[1] is more accurate when the preceding context is required to be "*the ball* ⟨1⟩". Specifying additional context instead of including the constraint in the pattern reduces the number of words consumed when applying the rule. This is useful in cases like "*player 4 has the ball in* REGION", where "*the ball*" contributes to more than one CLANG predicate (namely bowner and bpos). This is an example of non-compositionality, in which the phrases in a sentence do not have a one-to-one correspondence with subexpressions of the formal representation.

Another way of handling non-compositionality is to use templates with multiple productions. For example, a transformation rule that maps "TEAM *player* UNUM *has the ball in* REGION" to CONDITION → (and (bowner TEAM UNUM) (bpos REGION)), introduces a combination of three CLANG productions, eliminating the need for constraints on surrounding context.

[1] In CLANG, (bpos r) means the ball is in the region r.

Input: A training set T of NL sentences paired with formal representations; a set of productions Π in the formal grammar
Output: A learned rule base, L
Algorithm:
Parse all formal representations in T using Π.
Collect positive $\mathcal{P}_\pi$ and negative examples $\mathcal{N}_\pi$ for all $\pi \in \Pi$.
$L = \emptyset$
Until all positive examples are covered, or no more good rules can be found for any $\pi \in \Pi$, **do**:
  $R^* = \text{FINDBESTRULES}(\Pi, \mathcal{P}_\Pi, \mathcal{N}_\Pi)$ // see text
  $L = L \cup R^*$
  Apply rules in L to sentences in T.

Figure 1: LEARNRULES(T, Π): SILT's learning algorithm

Currently, the string-based version of SILT uses constraints on surrounding context and the tree-based version uses rule templates for multiple productions. However, we expect both approaches could be made to work with either string-based or tree-based patterns.

Learning Transformation Rules

SILT induces transformation rules from a set of NL sentences paired with their formal representations. Figure 1 shows an overview of the algorithm (LEARNRULES). First, unique parse trees are computed for each formal representation, which are well defined because the formal grammar is deterministic. Next, positive and negative examples are collected for each production π in the formal grammar as follows. Given an NL sentence s, if π is used in the parse tree of the formal representation of s, then s is a positive example of π, otherwise s is a negative example of π. SILT starts with an empty rule base and iteratively adds the best rules to it. The procedure for finding the best rules to add (FINDBESTRULES) is described in the following two sections. After the rule base is updated, all rules in it are applied to all training sentences. When a rule is applied, the parts of the sentences that are matched by it are replaced by the LHS non-terminal of its template. This allows subsequently-learned rules to use these non-terminals in their patterns. Rule learning is complete when all positive examples of all productions are covered by the rule base, or when no more good rules can be found for any production.

For the rule learning to be successful, two important issues need to be addressed. The first one is non-compositionality, which was discussed in the previous section. The second one is finding a set of rules that cooperate with each other to produce complete, correct formal-language translations. This is not trivial because rules are learned in a particular order. When a learned rule r is overly-general, it is possible that all subsequently-found candidate rules are overly-general, and no rules can be found to cooperate with r at a global level. We propose two approaches to this problem in SILT. The first one is to avoid learning overly-general rules by finding the single best rule for all competing productions in each iteration. The second one is to over-generate rules, and then find a subset of the learned rules that cooperate with each other. Currently, the string-

Input: A set of productions Π in the formal grammar; sets of positive $\mathcal{P}_\pi$ and negative examples $\mathcal{N}_\pi$ for each π in Π
Output: The best rule r^*
Algorithm:
$R = \emptyset$
For each production $\pi \in \Pi$:
  Let R_π be the maximally-specific rules derived from $\mathcal{P}_\pi$.
  Repeat for $k = 1000$ times:
    Choose $r_1, r_2 \in R_\pi$ at random.
    $g = \text{GENERALIZE}(r_1, r_2, \pi)$ // see text
    Add g to R_π.
  $R = R \cup R_\pi$
$r^* = \arg\max_{r \in R} goodness(r)$
Remove positive examples covered by r^* from $\mathcal{P}_{\pi^*}$.

Figure 2: FINDBESTRULES$(\Pi, \mathcal{P}_\Pi, \mathcal{N}_\Pi)$: Finding the best string-based rule to be included in the rule base

based version of SILT uses the former approach, and the tree-based version uses the latter. However both approaches are expected to work with either version.

Learning String-Based Transformation Rules

In order to find a set of inter-dependent cooperating rules, the procedure FINDBESTRULES of the string-based version of SILT simultaneously generates candidate rules for all productions and returns the overall best rule to be included in the rule base. Figure 2 shows a summary of this procedure. For every production π in the formal grammar, bottom-up rule induction is performed using a set of positive examples $\mathcal{P}_\pi$. Rule induction starts with maximally-specific rules, one rule for each positive example whose pattern is the complete, original sentence. These rules form the initial set of candidate rules, R_π, which are repeatedly generalized to form more general rules. In each of the $k = 1000$ iterations, two rules are randomly chosen from R_π and their generalization computed. The generalized rule is then added back to R_π for further generalization. This process is similar to the learning algorithm of GOLEM (Muggleton & Feng 1990). After candidate rules are found for all productions in the formal grammar, all rules are ranked using a goodness measure based on the number of positive $pos(r)$ and negative examples $neg(r)$ that a rule covers:

$$goodness(r) = \frac{(pos(r))^2}{pos(r) + neg(r)}$$

The goodness of a rule is the product of its accuracy and coverage. The overall best rule r^* is then chosen, and the positive examples that it covers are removed from $\mathcal{P}_{\pi^*}$. This approach allows rules for different productions to effectively compete with each other during the learning process in order to eventually produce an accurate, complete set of interacting transformations. The approach is similar to the one used for Multiple Predicate Learning (De Raedt, Lavrac, & Dzeroski 1993), in which Prolog clauses for several different inter-dependent concepts are learned simultaneously.

The key operation in the above rule induction procedure is the generalization of two candidate rules (GENERALIZE). Given two rules for a production π, GENERALIZE first computes all common subsequences of the rules' patterns that contain all non-terminals present in the RHS of π. Parameterized gaps are inserted between every two adjacent words in a common subsequence. GENERALIZE then chooses a subsequence which is long and has small gaps. Longer subsequences are preferred since they are less likely to match negative examples. It is also desirable that the words are nearby, so large gaps are discouraged. Each subsequence $c = c_1 c_2 \ldots c_n$ is scored using $\sigma(c) = n - \eta \sum_{i=0}^{n-1} g(c_i, c_{i+1})$, where $\eta = 0.4$ is the penalizing parameter and $g(c_i, c_{i+1})$ is the gap size between c_i and c_{i+1}. GENERALIZE returns the rule whose pattern is the subsequence with the highest $\sigma(c)$. Figure 3 shows the output of GENERALIZE given two sample string-based patterns.

Pattern 1: TEAM *'s penalty box*
Pattern 2: TEAM *penalty area*
Generalization: TEAM ⟨1⟩ *penalty*

Figure 3: Output of GENERALIZE with two string-based patterns, with π being REGION → `(penalty-area` TEAM`)`

The string-based version of SILT addresses the problem of non-compositionality by making part of the pattern a constraint on the surrounding context. Given a rule r returned by FINDBESTRULES, SILT heuristically finds the portion of its pattern that determines the subsequence of the sentence to be replaced by the transformation (called the *replacement subpattern*). The rest of the pattern only constrains the surrounding context. For every contiguous subpattern $r'$of r, the number of positive and negative examples matched by r' is computed and the subpattern that maximizes the heuristic $v(r') = (neg(r') - neg(r))/(pos(r') - pos(r) + \epsilon)$ is made the replacement subpattern. The small value $\epsilon = 0.01$ prevents the denominator from becoming zero. This heuristic captures the idea that the replacement pattern should cover basically the same positive examples covered by the complete pattern since it represents the NL phrase whose "meaning" is the formal subexpression introduced by the transformation. The goal of the remainder of the pattern is to eliminate negative examples, thereby disambiguating the given NL phrase (which may also have other "meanings") based on surrounding context.

Learning Tree-Based Transformation Rules

Similar to the string-based version, the tree-based version of SILT generates candidate rules using bottom-up rule induction. The rule generalization procedure (GENERALIZE) is based on computing common subgraphs of patterns. Given two rules, r_1 and r_2, GENERALIZE returns rules whose patterns are the largest common subgraphs of the subtrees of r_1's and r_2's patterns. For simplicity, gaps are not allowed in a pattern except above a non-terminal or below a syntactic marker at the leaf position. Figure 4 shows the output of GENERALIZE given a pair of sample tree-based patterns.

To handle non-compositionality, rules are learned with templates that simultaneously introduce multiple productions. Before LEARNRULES is invoked, the set of produc-

Pattern 1: (NP (NP TEAM (POS 's)) (NN *penalty*) (NN *box*))
Pattern 2: (NP (PRP$ TEAM) (NN *penalty*) (NN *area*))
Generalizations: (NP TEAM (NN *penalty*) (NN)), TEAM

Figure 4: Output of GENERALIZE with two tree-based patterns, with π being REGION → (penalty-area TEAM)

tions Π is augmented with *macro-productions*, chains of two or more productions. Macro-productions are created for all combinations of up to m productions observed in the training data. The optimal values of m for CLANG and GEOQUERY are 3 and 5, respectively. Macro-productions are then treated the same as original productions in the learning algorithm.

To find a cooperative set of rules, rules are first overgenerated. This is done by performing a form of beam search, where at most $\beta = 5$ different rules are allowed to redundantly cover the same positive example in the training data. The FINDBESTRULES procedure for the tree-based version of SILT is similar to the string-based version (Figure 2). A major difference is that instead of returning a single best rule, up to β best rules are now returned. The positive examples covered by these rules are not removed from the training set until they are redundantly covered by β different rules. Also in each invocation of FINDBESTRULES, only the π that has the most positive examples in $\mathcal{P}_\pi$ is examined, which is less time-consuming than examining all $\pi \in \Pi$ at once.

To find a subset of the learned rules that cooperate with each other, rules that seldom lead to complete, correct formal-language translations for the training data are removed from the rule base, as a post-processing step after LEARNRULES. The pruned rule base is further refined by learning additional constraints for each rule. These constraints specify the maximum number of words that can be skipped when a rule is applied, and the words that can or cannot be skipped. These constraints are learned by simple counting. For example, if for all sentences in the training data, at most n words are skipped when a rule r is applied and the application of r leads to a complete, correct parse, then the gap size of r is set to n.

Since the size of the resulting rule base is potentially large, a sentence usually gives rise to many formal-language translations. These translations are ranked using a heuristic function $\rho(f) = \sum_{r \in R_f} a(r) \times (w(r)+1)$, where R_f is the set of rules used in deriving the formal representation f, $a(r)$ the accuracy of rule r, and $w(r)$ the number of words that the pattern of r contains. The formal representation f with the highest $\rho(f)$ is then chosen as the output. This heuristic captures the idea that as many words should be matched as possible using rules that are reasonably accurate.

Experiments

Methodology

Two corpora of NL sentences paired with formal representations were constructed as follows. For CLANG, 300 pieces of coaching advice were randomly selected from the log files of the 2003 RoboCup Coach Competition. Each formal instruction was translated into English by one of four annotators. The average length of an NL sentence in this corpus is 22.52 words. For GEOQUERY, 250 questions were collected by asking undergraduate students to generate English queries for the given database. Queries were then manually translated into logical form (Zelle & Mooney 1996). The average length of an NL sentence in this corpus is 6.87 words. The queries in this corpus are more complex than those in the ATIS database-query corpus used in the speech recognition community (Zue & Glass 2000) which makes the GEOQUERY problem harder, as also shown by the results in (Popescu *et al.* 2004).

SILT was evaluated using standard 10-fold cross validation. Syntactic parses required by SILT's tree-based version were generated by Collins' parser (Bikel 2004) trained with the WSJ treebank and gold-standard parse trees of the training sentences. The test sentences were then transformed using the learned semantic parser. We computed the number of test sentences that produced complete translations, and the number of these translations that were correct. For CLANG, a translation is correct if it exactly matched the correct representation, up to reordering of the arguments of commutative operators like and. For GEOQUERY, a translation is correct if the resulting query retrieved the same answer as the correct representation when submitted to the database. Then the performance of the parser was measured in terms of precision (the percentage of completed translations that were correct) and recall (the percentage of all sentences that were correctly translated).

We compared SILT's performance with that of CHILL, an ILP framework for learning semantic parsers. We ran two versions of CHILL, one based on the CHILLIN induction algorithm (Zelle & Mooney 1996), the other based on COCKTAIL (Tang & Mooney 2001), which uses multiple clause constructors based on the CHILLIN and mFOIL algorithms (Lavrac & Dzeroski 1994). We also compared SILT with GEOBASE (Borland International 1988), a hand-built NL interface for the GEOQUERY domain.

The original formal queries in GEOQUERY were in Prolog; however, SILT's transformation mechanism is best suited for functional or imperative command languages like CLANG. Hence, when applying SILT to GEOQUERY, the logical queries were automatically translated into an equivalent variable-free functional form using a simple conversion program. In an exact opposite manner, CHILL's parsing mechanism is best suited for logical languages. Hence in order to apply it to the CLANG corpus, the advice expressions were automatically translated into an equivalent Prolog form using another simple conversion program.

Results

Figures 5 and 6 show the precision and recall learning curves for GEOQUERY, and Figures 7 and 8 for CLANG. The labels "SILT-string" and "SILT-tree" denote the string-based and tree-based versions of SILT, respectively. Table 1 gives the training time in minutes for all the systems for the last points on the learning curves. Since COCKTAIL is memory intensive, it could not be run with larger training sets of the

	GEOQUERY	CLANG
SILT-string	0.35	3.2
CHILLIN	6.3	10.4
SILT-tree	21.5	81.4
COCKTAIL	39.6	-

Table 1: Average training time in minutes for the last points on the learning curves

CLANG corpus.

On the GEOQUERY corpus, COCKTAIL has the best recall but the worst precision. On the CLANG corpus, both versions of SILT do a lot better than either COCKTAIL or CHILLIN. There are two main reasons for this. First, COCKTAIL and CHILLIN do not exploit the formal grammar of the target language. The intermediate non-terminals of CLANG are very suitable for expressing good generalizations, and SILT uses them to its advantage. Second, COCKTAIL and CHILLIN use a shift-reduce parsing framework to parse a sentence from left to right. This type of parsing is restrictive because if the system fails to parse the left side of the sentence then it will fail to parse the entire sentence. In contrast, SILT can parse a sentence starting from anywhere; for example, it may first parse the right side of the sentence which may help it in parsing the left side leading to a complete parse. It may also be noted that the CLANG corpus is the harder corpus since it has longer sentences with larger formal representations.

The two versions of SILT give comparable precisions but the tree-based version gives much higher recalls. The string-based version, however, runs much faster. The tree-based version uses the additional information of English syntax and does a much more thorough search for finding inter-dependent co-operating rules which contributes to its better performance.

Results on a larger GEOQUERY corpus with 880 queries have been reported for PRECISE (Popescu, Etzioni, & Kautz 2003): 100% precision and 77.5% recall. On the same corpus, the tree-based version of SILT obtains 87.85% precision and 53.41% recall. However, the figures are not comparable. PRECISE can return multiple distinct SQL queries when it judges a question to be ambiguous and it is considered correct when *any* of these SQL queries is correct. Our measure only considers the top result. Moreover, PRECISE is designed to work only for the specific task of NL database interfaces. By comparison, SILT is more general and can work with other target formal languages as well (e.g. CLANG).

Future Work

Compared to previous ILP-based methods, SILT allows for a more global approach to semantic parsing that is not constrained to making correct local decisions after incrementally processing each word. However, it still lacks some of the robustness of statistical parsing. A more recent approach by Ge & Mooney (2005) adds detailed semantics to a state-of-the-art statistical parser. Their system learns a statistical

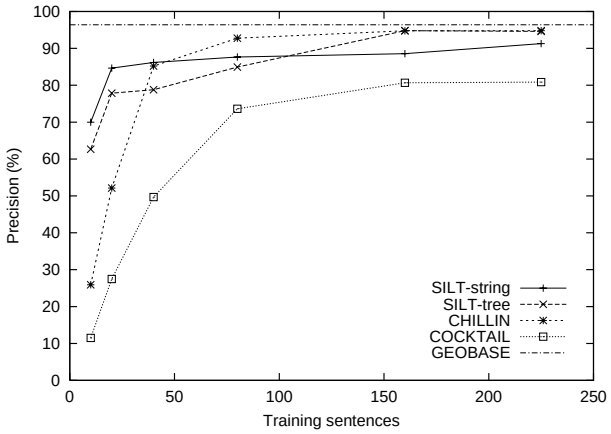

Figure 5: Precision learning curves for GEOQUERY

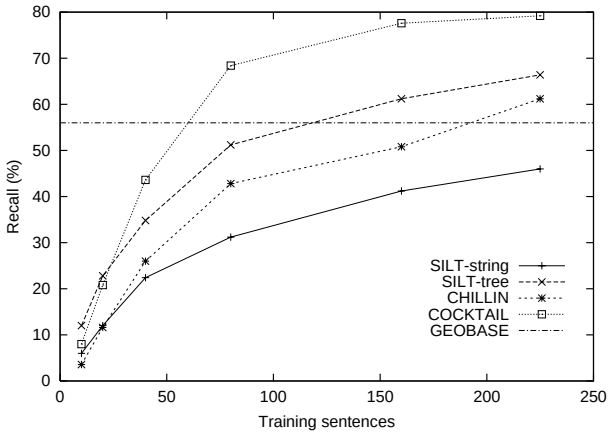

Figure 6: Recall learning curves for GEOQUERY

parser that generates a *semantically augmented parse tree*, in which each internal node is given both a syntactic and a semantic label. By integrating syntactic and semantic interpretation into a single statistical model and finding the globally most probable parse, an accurate combined syntactic/semantic analysis can be obtained. However, this approach requires semantically augmented parse trees as additional training input.

Currently, the hard-matching symbolic rules of SILT are sometimes too brittle to appropriately capture the range of contexts in which a concept in the formal language should be introduced. We are currently developing an improved version of SILT by exploiting the use of string and tree *kernels* (Lodhi *et al.* 2002). This will allow a more robust and flexible mechanism for triggering transformations compared to hard-matching regular expressions.

Finally, we are also developing a more unified implementation of the string-based and tree-based versions of SILT that will allow a direct comparison that evaluates the benefit of using initial syntactic parses. The current two versions differ along several dimensions that are independent of the

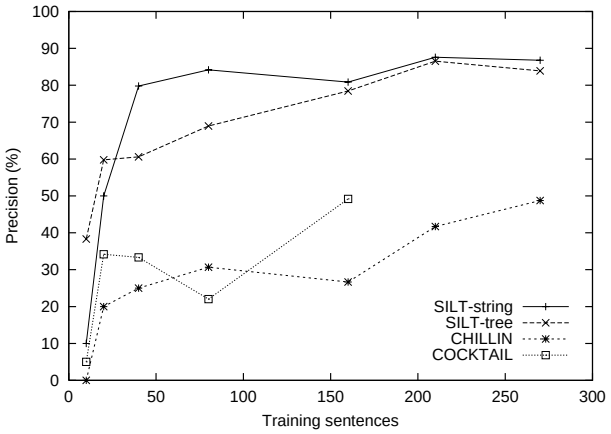

Figure 7: Precision learning curves for CLANG

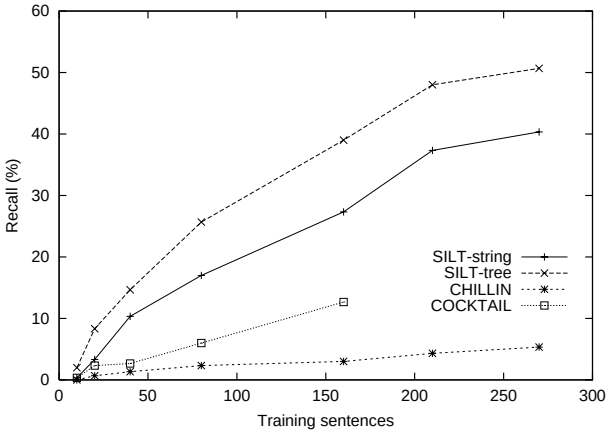

Figure 8: Recall learning curves for CLANG

use of a syntactic parser, which prevents attributing the performance advantage of SILT-tree to this aspect alone.

Conclusions

We have presented a novel approach, SILT, for learning transformation rules that map NL sentences into a formal representation language. The system learns these transformation rules from the training data by doing bottom-up rule induction using the target language grammar. It was applied to two very different domains and was shown to perform overall better and faster than previous ILP-based approaches.

Acknowledgments

We would like to thank Ruifang Ge and Gregory Kuhlmann for their help in annotating the CLANG corpus. This research was supported by Defense Advanced Research Projects Agency under grant HR0011-04-1-0007.

References

Bikel, D. M. 2004. Intricacies of Collins' parsing model. *Computational Linguistics* 30(4):479–511.

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

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.

Chen, M.; Foroughi, E.; Heintz, F.; Kapetanakis, S.; Kostiadis, K.; Kummeneje, J.; Noda, I.; Obst, O.; Riley, P.; Steffens, T.; Wang, Y.; and Yin, X. 2003. Users manual: RoboCup soccer server manual for soccer server version 7.07 and later. Available at http://sourceforge.net/projects/sserver/.

De Raedt, L.; Lavrac, N.; and Dzeroski, S. 1993. Multiple predicate learning. In *Proc. of IJCAI-93*, 1037–1042.

Ge, R., and Mooney, R. J. 2005. A statistical semantic parser that integrates syntax and semantics. To appear in *Proc. of CoNLL-05*.

Gildea, D., and Jurafsky, D. 2002. Automated labeling of semantic roles. *Computational Linguistics* 28(3):245–288.

Kuhlmann, G.; Stone, P.; Mooney, R.; and Shavlik, J. 2004. Guiding a reinforcement learner with natural language advice: Initial results in RoboCup soccer. In *Proc. of the AAAI-04 Workshop on Supervisory Control of Learning and Adaptive Systems*.

Lavrac, N., and Dzeroski, S. 1994. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood.

Lodhi, H.; Saunders, C.; Shawe-Taylor, J.; Cristianini, N.; and Watkins, C. 2002. Text classification using string kernels. *JMLR* 2:419–444.

Muggleton, S., and Feng, C. 1990. Efficient induction of logic programs. In *Proc. of 1st Conf. on Algorithmic Learning Theory*. Tokyo, Japan: Ohmsha.

Popescu, A.-M.; Armanasu, A.; Etzioni, O.; Ko, D.; and Yates, A. 2004. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *Proc. of COLING-04*.

Popescu, A.-M.; Etzioni, O.; and Kautz, H. 2003. Towards a theory of natural language interfaces to databases. In *Proc. of IUI-03*, 149–157. Miami, FL: ACM.

Tang, L. R., and Mooney, R. J. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proc. of ECML-01*, 466–477.

Woods, W. A. 1977. Lunar rocks in natural English: Explorations in natural language question answering. In Zampoli, A., ed., *Linguistic Structures Processing*. New York: Elsevier North-Holland.

Zelle, J. M., and Mooney, R. J. 1996. Learning to parse database queries using inductive logic programming. In *Proc. of AAAI-96*, 1050–1055.

Zue, V. W., and Glass, J. R. 2000. Conversational interfaces: Advances and challenges. In *Proc. of the IEEE*, volume 88(8), 1166–1180.