# Transforming Meaning Representation Grammars to Improve Semantic Parsing

**Rohit J. Kate**
Department of Computer Sciences*
The University of Texas at Austin
1 University Station C0500
Austin, TX 78712-0233, USA
rjkate@cs.utexas.edu

## Abstract

A semantic parser learning system learns to map natural language sentences into their domain-specific formal meaning representations, but if the constructs of the meaning representation language do not correspond well with the natural language then the system may not learn a good semantic parser. This paper presents approaches for automatically transforming a meaning representation grammar (MRG) to conform it better with the natural language semantics. It introduces grammar transformation operators and meaning representation macros which are applied in an error-driven manner to transform an MRG while training a semantic parser learning system. Experimental results show that the automatically transformed MRGs lead to better learned semantic parsers which perform comparable to the semantic parsers learned using manually engineered MRGs.

## 1 Introduction

Semantic parsing is the task of converting *natural language* (NL) sentences into their *meaning representations* (MRs) which a computer program can execute to perform some domain-specific task, like controlling a robot, answering database queries etc. These MRs are expressed in a formal *meaning representation language* (MRL) unique to the domain to suit the application, like some specific command language to control a robot or some query language to execute database queries. A machine learning system for semantic parsing takes NL sentences paired with their respective MRs as training data and induces a semantic parser which can then map novel NL sentences into their MRs.

The grammar of an MRL, which we will call *meaning representation grammar* (MRG), is assumed to be deterministic and context-free which is true for grammars of almost all the computer executable languages. A semantic parsing learning system typically exploits the given MRG of the MRL to learn a semantic parser (Kate and Mooney, 2006; Wong and Mooney, 2006). Although in different ways, but the systems presented in these papers learn how the NL phrases relate to the *productions* of the MRG, and using this information they parse a test sentence to compositionally generate its best MR. In order to learn a good semantic parser, it is necessary that the productions of the MRG accurately represent the semantics being expressed by the natural language. However, an MRL and its MRG are typically designed to best suit the application with little consideration for how well they correspond to the semantics of a natural language.

Some other semantic parser learning systems which need MRL in the form of Prolog (Tang and Mooney, 2001) or $\lambda$-calculus (Zettlemoyer and Collins, 2007; Wong and Mooney, 2007) do not use productions of the MRG but instead use predicates of the MRL. However, in order to learn a good semantic parser, they still require that these predicates correspond well with the semantics of the natural language. There are also systems which learn semantic parsers from more detailed training data in the form of semantically augmented parse trees of NL sentences in which each internal node has a syntactic and a semantic label (Ge

*Alumnus at the time of submission.

(a) NL: *If the ball is in our midfield then player 5 should go to (-5,0).*
```
  MR: (bpos (rec (pt -32 -35)(pt 0 35))
        (do (player our {5})(pos (pt -5 0))))
```

(b) NL: *Which is the longest river in Texas?*
```
  MR: answer(longest(river(loc_2(stateid('Texas')))))
```

(c) NL: *Which is the longest river in Texas?*
```
  MR: select river.name from river where
       river.traverse='Texas' and river.length=
         (select max(river.length) from river
          where river.traverse='Texas');
```

Figure 1: Examples of NL sentences and their MRs from (a) the CLANG domain (b) GEOQUERY domain with functional MRL (c) GEOQUERY domain with SQL.

and Mooney, 2005; Nguyen et al., 2006). For these systems to work well, it is also necessary that the semantic labels of the MRL correspond well with natural language semantics.

If the MRG of a domain-specific MRL does not correspond well with natural language semantics then manually re-engineering the MRG to work well for semantic parsing is a tedious task and requires considerable domain expertise. In this paper, we present methods to automatically transform a given MRG to make it more suitable for learning semantic parsers. No previous work addresses this issue to our best knowledge. We introduce grammar transformation operators and meaning representation macros to transform an MRG. We describe how these are applied in an error-driven manner using the base semantic parsing learning algorithm presented in (Kate and Mooney, 2006) resulting in a better learned semantic parser. Our approach, however, is general enough to improve any semantic parser learning system which uses productions of the MRG. We present experimental results with three very different MRLs to show how these grammar transformations improve the semantic parsing performance.

## 2 Background

The following subsection gives some examples of semantic parsing domains and their corresponding MRLs and illustrates why incompatibility between MRGs and natural language could hurt semantic parsing. The next subsection then briefly describes a base semantic parser learning system which we use in our experiments.

### 2.1 MRLs and MRGs for Semantic Parsing

Figure 1 (a) gives an example of a natural language sentence and its corresponding MR in an MRL called CLANG which is a formal declar-

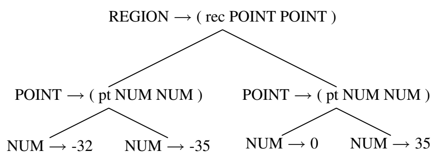

Figure 2: The parse for the CLANG expression “(rec (pt -32 -35) (pt 0 35))” corresponding to the natural language utterance “our midfield” using its original MRG.

ative language with LISP-like prefix notation designed to instruct simulated soccer players in the RoboCup[1] Coach Competition. The MRL and its MRG was designed by the Coach Competition community (Chen et al., 2003) to suit the requirements of their application independent of how well the MRG conforms with the natural language semantics. They were, in fact, not aware that later (Kate et al., 2005) this will be introduced as a test domain for learning semantic parsers. In this original MRG for CLANG, there are several constructs which do not correspond well with their meanings in the natural language. For example, the MR expression of the rectangle `(rec (pt -32 -35) (pt 0 35))` from the example MR in Figure 1 (a), whose parse according to the original MRG is shown in Figure 2, corresponds to the NL utterance “our midfield”. In the parse tree, the nodes are the MRG productions and the tokens in upper-case are non-terminals of the MRG while the tokens in lower-case are terminals of the MRG, this convention will be used throughout the paper. As can be seen, the numbers as well as the productions in the parse of the MR expression do not correspond to anything in its natural language utterance. It is also impossible to derive a semantic parse tree of this MR expression over its natural language utterance because there are not enough words in it to cover all the productions present in the MR parse at the lowest level. To alleviate this problem, the provided MRG was manually modified (Kate et al., 2005) to make it correspond better with the natural language by replacing such long MR expressions for soccer regions by shorter expressions like `(midfield our)`[2]. This new MRG was used in all the previous work which uses the CLANG corpus. In the next sections of the paper, we will present methods to automatically obtain a

[1]http://www.robocup.org

[2]The names for the new tokens introduced were chosen for readability and their similarity to the natural language words is inconsequential for learning semantic parsers.

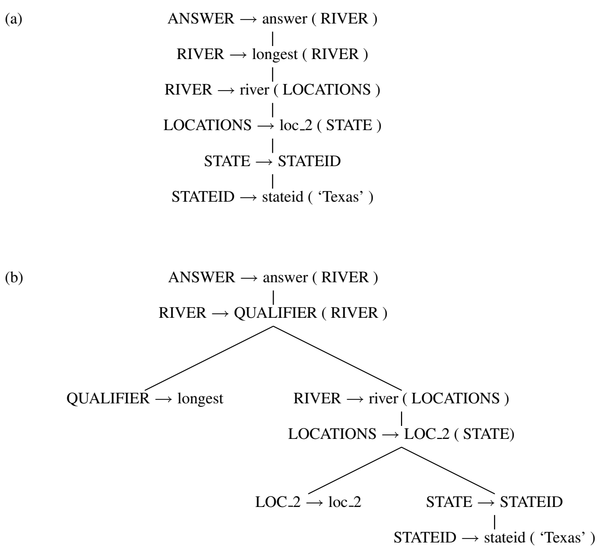
(a)
ANSWER → answer ( RIVER )
RIVER → longest ( RIVER )
RIVER → river ( LOCATIONS )
LOCATIONS → loc_2 ( STATE )
STATE → STATEID
STATEID → stateid ( 'Texas' )
(b)
ANSWER → answer ( RIVER )
RIVER → QUALIFIER ( RIVER )
QUALIFIER → longest
RIVER → river ( LOCATIONS )
LOCATIONS → LOC_2 ( STATE)
LOC_2 → loc_2
STATE → STATEID
STATEID → stateid ( 'Texas' )

Figure 3: Different parse trees obtained for the MR "answer(longest(river(loc_2(stateid('Texas')))))" corresponding to the NL sentence "Which is the longest river in Texas?" using (a) a simple MRG (b) a manually designed MRG.

better MRG which corresponds well with the NL semantics.

Figure 1 (b) shows an NL sentence and its MR from the GEOQUERY domain (Zelle and Mooney, 1996) which consists of a database of U.S. geographical facts about which a user can query. The MRL used for GEOQUERY in some of the previous work is a variable-free functional query language, that was constructed from the original MRs in Prolog (Kate et al., 2005). From this MRL, the MRG was then manually written so that its productions were compatible with the semantics expressible in natural language. This MRG was different from some simple MRG one would otherwise design for the MRL. Figure 3 (a) shows the parse tree obtained using a simple MRG for the MR shown in Figure 1 (b). The MR parse obtained using the simple MRG is more like a linear chain which means that in a semantic parse of the NL sentence each production will have to correspond to the entire sentence. But ideally, different productions should correspond to the meanings of different substrings of the sentence. Figure 3 (b) shows a parse tree obtained using the manually designed MRG in which the productions QUALIFIER → longest and LOC_2 → loc_2 would correspond to the semantic concepts of "longest" and "located in" that are expressible in natural language.

Finally, Figure 1 (c) shows the same NL sentence from the GEOQUERY domain but the MR in SQL which is the standard database query language. The inner expression finds the length of the longest river in Texas and then the outer expression finds the river in Texas which has that length. Due to space restriction, we are not showing the parse tree for this SQL MR, but its incompatibility with the NL sentence can be seen from the MR itself because part of the query repeats itself with `'Texas'` appearing twice while in the NL sentence everything is said only once.

### 2.2 KRISP: A Semantic Parser Learning System

We very briefly describe the semantic parser learning system, KRISP (Kate and Mooney, 2006), which we will use as a base system for transforming MRGs, we however note that the MRG transformation methods presented in this paper are general enough to work with any system which learns semantic parser using MRGs. KRISP (Kernel-based Robust Interpretation for Semantic Parsing) is a supervised learning system for semantic parsing which takes NL sentences paired with their MRs as training data. The productions of the MRG are treated like semantic concepts. For each of these productions, a Support-Vector Machine classifier is trained using string similarity as the kernel (Lodhi et al., 2002). Each classifier can then estimate the probability of any NL substring representing the semantic concept for its production. During semantic parsing, the classifiers are called to estimate probabilities on different substrings of the sentence to compositionally build the most probable MR parse over the entire sentence with its productions covering different substrings of the sentence. KRISP was shown to perform competitively with other existing semantic parser learning systems and was shown to be particularly robust to noisy NL input.

## 3 Transforming MRGs Using Operators

This section describes an approach to transform an MRG using grammar transformation operators to conform it better with the NL semantics. The following section will present another approach for transforming an MRG using macros which is sometimes more directly applicable.

The MRLs used for semantic parsing are always assumed to be context-free which is true for almost all executable computer languages. There has been some work in learning *context-free grammars* (CFGs) for a language given several examples of its expressions (Lee, 1996). Most of the approaches directly learn a grammar from the expressions but there also have been approaches that first start with a simple grammar and then transform it using suitable operators to a better grammar (Langley and Stromsten, 2000). The goodness for a grammar is typically measured in terms of its simplicity and coverage. Langley and Stromsten (2000) transform syntactic grammars for NL sentences. To our best knowledge, there is no previous work on transforming MRGs for semantic parsing. For this task, since an initial MRG is always given with the MRL, there is no need to first learn it from its MRs. The next subsection describes the operators our method uses to transform an initial MRG. The subsection following that then describes how and when the operators are applied to transform the MRG during training. Our criteria for goodness of an MRG is the performance of the semantic parser learned using that MRG.

## 3.1 Transformation Operators

We describe five transformation operators which are used to transform an MRG. Each of these operators preserves the coverage of the grammar, i.e. after application of the operator, the transformed grammar generates the same language that the previous grammar generated[3]. The MRs do not change but only the way they are parsed may change because of grammar transformations. This is important because the MRs are to be used in an application and hence should not be changed.

**1. Create Non-terminal from a Terminal (CreateNT):** Given a terminal symbol $t$ in the grammar, this operator adds a new production $T \rightarrow t$ to it and replaces all the occurrences of the terminal $t$ in all the other productions by the new non-terminal $T$. In the context of semantic parsing learning algorithm, this operator introduces a new semantic concept the previous grammar was not explicit about. For example, this operator may introduce a production (a semantic concept) LONGEST → longest to the simple grammar whose parse was shown in Figure 3 (a). This is close to the production QUALIFIER → longest of the manual grammar used in the parse shown in Figure 3 (b).

**2. Merge Non-terminals (MergeNT):** This operator merges $n$ non-terminals $T_1$, $T_2$, ..., $T_n$, by introducing $n$ productions $T \rightarrow T_1$, $T \rightarrow T_2$, ..., $T \rightarrow T_n$ where $T$ is a new non-terminal. All the occurrences of the merged non-terminals on the right-hand-side (RHS) of all the remaining productions are then replaced by the non-terminal $T$. In order to ensure that this operator preserves the coverage of the grammar, before applying it, it is made sure that if one of these non-terminals, say $T_1$, occurs on the RHS of a production $\pi_1$ then there also exist productions $\pi_2$, ..., $\pi_n$ which are same as $\pi_1$ except that $T_2$, ..., $T_n$ respectively occur in them in place of $T_1$. If this condition is violated for any production of any of the $n$ non-terminals then this operator is not applicable. This operator enables generalization of some non-terminals which occur in similar contexts leading to generalization of productions in which they occur on the RHS. For example, this operator may generalize non-terminals LONGEST and SHORTEST in GEOQUERY MRG to form QUALIFIER[4] → LONGEST and QUALIFIER → SHORTEST productions.

**3. Combine Two Non-terminals (CombineNT):** This operator combines two non-terminals $T_1$ and $T_2$ into one new non-terminal $T$ by introducing a new production $T \rightarrow T_1\ T_2$. All the instances of $T_1$ and $T_2$ occurring adjacent in this order on the RHS (with at least one more non-terminal[5]) of all the other productions are replaced by the new non-terminal $T$. For example, the production $A \rightarrow a\ B\ T_1\ T_2$ will be changed to $A \rightarrow a\ B\ T$. This operator will not eliminate other occurrences of $T_1$ and $T_2$ on the RHS of other productions in which they do not occur adjacent to each other. In the context of semantic parsing, this operator adds an extra level in the MR parses which does not seem to be useful in itself, but later if the non-terminals $T_1$ and $T_2$ get eliminated (by the application of the DeleteProd operator described shortly), this operator will be combining the concepts represented by the two non-terminals.

**4. Remove Duplicate Non-terminals (RemoveDuplNT):** If a production has the same non-terminal appearing twice on its RHS then this operator adds an additional production which differs from the first production in that it has only one occurrence of that non-terminal. For example, if a production is $A \rightarrow b\ C\ D\ C$, then this operator will introduce a new production $A \rightarrow b\ C\ D$ re-

[3]This is also known as weak equivalence of grammars.

[4]A system generated name will be given to the new non-terminal.

[5]Without the presence of an extra non-terminal on the RHS, this change will merely add redundancy to the parse trees using this production.

moving the second occurrence of the non-terminal $C$. This operator is applied only when the subtrees under the duplicate non-terminals of the production are often found to be the same in the parse trees of the MRs in the training data. As such this operator will change the MRL the new MRG will generate, but this can be easily reverted by appropriately duplicating the subtrees in its generated MR parses in accordance to the original production. This operator is useful during learning a semantic parser because it eliminates the type of incompatibility between MRs and NL sentences illustrated with Figure 1 (c) in Subsection 2.1.

**5. Delete Production (DeleteProd):** This last operator deletes a production and replaces the occurrences of its left-hand-side (LHS) non-terminal with its RHS in the RHS of all the other productions. In terms of semantic parsing, this operator eliminates the need to learn a semantic concept. It can undo the transformations obtained by the other operators by deleting the new productions they introduce.

We note that the CombineNT and MergeNT operators are same as the two operators used by Langley and Stromsten (2000) to search a good syntactic grammar for natural language sentences from the space of its possible grammars. We also note that the applications of CreateNT and CombineNT operators can reduce a CFG to its Chomsky normal form[6], and conversely, because of the reverse transformations achieved by the DeleteProd operator, a Chomsky normal form of a CFG can be converted into any other CFG which accepts the same language.

## 3.2 Applying Transformation Operators

In order to transform an MRG to improve semantic parsing, since a simple hill-climbing type approach to search the space of all possible MRGs will be computationally very intensive, we use the following error-driven heuristic search which is faster although less thorough.

First, using the provided MRG and the training data, a semantic parser is trained using KRISP. The trained semantic parser is applied to each of the training NL sentences. Next, for each production $\pi$ in the MRG, two values $total_\pi$ and $incorrect_\pi$ are computed. The value $total_\pi$ counts how many MR parses from the training examples use the production $\pi$. The value $incorrect_\pi$ counts the number of training examples for which the semantic parser incorrectly uses the production $\pi$, i.e. it either did not include the production $\pi$ in the parse of the MR it produces when the correct MR's parse included it, or it included the production $\pi$ when it was not present in the correct MR's parse. These two statistics for a production indicate how well the semantic parser was able to use the production in semantic parsing. If it was not able to use a production $\pi$ well, then the ratio $incorrect_\pi/total_\pi$, which we call $mistakeRatio_\pi$, will be high indicating that some change needs to be made to that production. After computing these values for all the productions, the procedure described below for applying the first type of operator is followed. After this, the MRs in the training data are re-parsed using the new MRG, the semantic parser is re-trained and the $total_\pi$ and $incorrect_\pi$ values are re-computed. Next, the procedure for applying the next operator is followed and so on. The whole process is repeated for a specified number of iterations. In the experiments, we found that the performance does not improve much after two iterations.

**1. Apply CreateNT:** For each terminal $t$ in the grammar, $total_t$ and $incorrect_t$ values are computed by summing up the corresponding values for all the productions in which $t$ occurs on the RHS with at least one non-terminal[7]. If $total_t$ is greater than $\beta$ (a parameter) and $mistakeRatio_t = incorrect_t/total_t$ is greater than $\alpha$ (another parameter), then the CreateNT operator is applied, provided the production $T \rightarrow t$ is not already present.

**2. Apply MergeNT:** All the non-terminals occurring on the RHS of all those productions $\pi$ are collected whose $mistakeRatio_\pi$ value is greater than $\alpha$ and whose $total_\pi$ value is greater than $\beta$. The set of these non-terminals is then partitioned such that the criteria for applying the MergeNT is satisfied by the non-terminals in each partition with size at least two. The MergeNT operator is then applied to the non-terminals in each partition with size at least two.

**3. Apply CombineNT:** For every non-terminal pair $T_1$ and $T_2$, $total_{T_1T_2}$ and $incorrect_{T_1T_2}$ values are computed by summing their corresponding values for the productions in which the two non-terminals are adjacent in the RHS in the

[6]In which all the productions are of the form $A \rightarrow a$ or $A \rightarrow B\,C$.

[7]Without a non-terminal on the RHS, the operator will only add a redundant level to the parses which use this production.

presence of at least one more non-terminal. If $mistakeRatio_{T_1T_2} = incorect_{T_1T_2}/total_{T_1T_2}$ is greater than $\alpha$ and $total_{T_1T_2}$ is greater than $\beta$, then the CombineNT operator is applied to these two non-terminals.

**4. Apply RemoveDuplNT:** If a production $\pi$ has duplicate non-terminals on the RHS under which the same subtrees are found in the MR parse trees of the training data more than once then this operator is applied provided its $mistakeRatio_\pi$ is greater than $\alpha$ and $total_\pi$ is greater than $\beta$.

**5. Apply DeleteProd:** The DeleteProd operator is applied to all the productions $\pi$ and whose $mistakeRatio_\pi$ is greater than $\alpha$ and $total_\pi$ is greater than $\beta$. This step simply deletes the productions which are mostly incorrectly used.

For the experiments, we set the $\alpha$ parameter to $0.75$ and $\beta$ parameter to 5, these values were determined through pilot experiments.

## 4 Transforming MRGs Using Macros

As was illustrated with Figure 2 in Subsection 2.1, sometimes there can be large parses for MR expressions which do not correspond well with their semantics in the natural language. While it is possible to transform the MRG using the operators described in the previous section to reduce a subtree of the parse to just one production which will then correspond directly to its meaning in the natural language, it will require a particular sequence of transformation operators to achieve this which may rarely happen during the heuristic search used in the MRG transformation algorithm. In this section, we describe a more direct way of obtaining such transformations using macros.

### 4.1 Meaning Representation Macros

A meaning representation macro for an MRG is a production formed by combining two or more existing productions of the MRG. For example, for the CLANG example shown in Figure 2, the production REGION → (rec(pt -32 -35)(pt 0 35)) is a meaning representation macro. There could also be non-terminals on its RHS. From an MR parse drawn with non-terminals at the internal nodes (instead of productions), a macro can be derived from a subtree[8] rooted at any of the internal nodes by making its root as the LHS non-terminal and the left-to-right sequence formed by its leaves (which could be non-terminals) as the RHS. We use the following error-driven procedure to introduce macros in the MRG in order to improve the performance of semantic parsing.

[8]Each node of a subtree must either include all the children nodes of the corresponding node from the original tree or none of them.

### 4.2 Learning Meaning Representation Macros

A semantic parser is first learned from the training data using KRISP and the given MRG. The learned semantic parser is then applied to the training sentences and if the system can not produce any parse for a sentence then the parse tree of its corresponding MR is included in a set called *failed parse trees*. Common subtrees in these failed parse trees are likely to be good candidates for introducing macros. Then a set of *candidate trees* is created as follows. This set is first initialized to the set of failed parse trees. The largest common subtree of every pair of trees in the candidate trees is then also included in this set if it is not an empty tree. The process continues with the newly added trees until no new tree can be included. This process is similar to the repeated bottom-up generalization of clauses used in the inductive logic programming system GOLEM (Muggleton and Feng, 1992). Next, the trees in this set are sorted based on the number of failed parse trees of which they are a subtree. The trees which are part of fewer than $\beta$ subtrees are removed. Then in highest to lowest order, the trees are selected one-by-one to form macros, provided their height is greater than two (otherwise it will be an already existing MRG production) and an already selected tree is not its subtree. A macro is formed from a tree by making the non-terminal root of the tree as its LHS non-terminal and the left-to-right sequence of the leaves as its RHS.

These newly formed macros (productions) are then included in the MRG. The MRs in the training data are re-parsed and the semantic parser is re-trained using the new MRG. In order to delete the macros which were not found useful, a procedure similar to the application of DeleteProd is used. The $total_\pi$ and $incorrect_\pi$ values for all the macros are computed in a manner similar to described in the previous section. The macros for which $mistakeRatio_\pi = total_\pi/incorrect_\pi$ is greater than $\alpha$ and $total_\pi$ is greater than $\beta$ are removed. This whole procedure of adding and deleting macros is repeated a specified number of iterations. In the experiments, we found that two

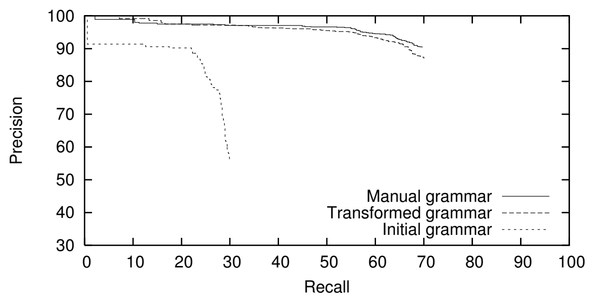

Figure 4: The results comparing the performances of the learned semantic parsers on the GEOUQERY domain with the functional query language using different MRGs.

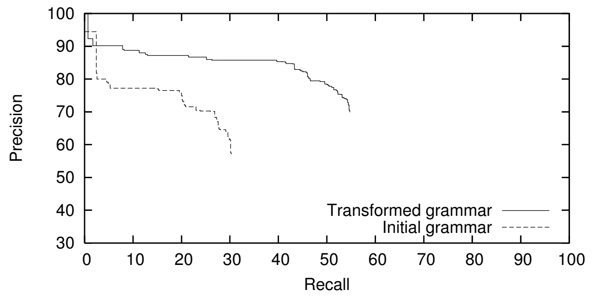

Figure 5: The results comparing the performances of the learned semantic parsers on the GEOUQERY domain with SQL as the MRL using different MRGs.

iterations are usually sufficient.

## 5 Experiments

We tested our MRG transformation methods with MRGs of three different MRLs which were described in the Background section. In each case, we first transformed the given MRG using macros and then using grammar transformation operators. The training and testing was done using standard 10-fold cross-validation and the performance was measured in terms of precision (the percentage of generated MRs that were correct) and recall (the percentage of all sentences for which correct MRs were obtained). Since we wanted to evaluate how the grammar transformation changes the performance on the semantic parsing task, in each of the experiments, we used the same system, KRISP, and compared how it performs when trained using different MRGs for the same MRL. Since KRISP assigns confidences to the MRs it generates, an entire range of precision-recall trade-off was plotted by measuring precision and recall values at various confidence levels.

Figure 4 shows the results on the GEOQUERY domain using the functional query language whose

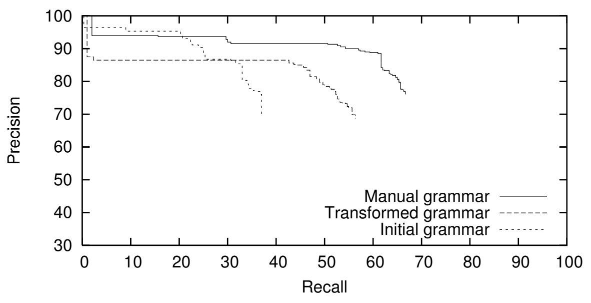

Figure 6: The results comparing the performances of the learned semantic parsers on the CLANG corpus using different MRGs.

corpus contained total 880 NL-MR pairs. As can be seen, the performance of the semantic parser that KRISP learns when trained using the initial simple MRG for the MRL is not good. But when that MRG is transformed, the performance of the semantic parser dramatically improves and is very close to the performance obtained with the manually-engineered grammar. The macro transformations did not help improve the performance with this MRG, and most of the the performance gain was obtained because of the CreateNT and DeleteProd operators.

We next tested our MRG transformation algorithm on SQL as the MRL for the GEOQUERY domain. This corpus contains 700 NL-MR pairs in which the NL sentences were taken from the original 880 examples. This corpus was previously used to evaluate the PRECISION system (Popescu et al., 2003), but since that system is not a machine learning system, its results cannot be directly compared with ours. The initial MRG we used contained the basic SQL productions. Figure 5 shows that results improve by a large amount after MRG transformations. We did not have any manually-engineered MRG for SQL for this domain available to us. With this MRG, most of the improvement was obtained using the macros and the RemoveDuplNT transformation operator.

Finally, we tested our MRG transformation algorithm on the CLANG domain using its original MRG in which all the chief regions of the soccer field were in the form of numeric MR expressions which do not correspond to their meanings in the natural language. Its corpus contains 300 examples of NL-MR pairs. Figure 6 shows the results. After applying the MRG transformations the performance improved by a large margin. The gain was due to transformations obtained using macros while the grammar transformation operators did not help with this MRG. Although the precision was lower for low recall values, the recall increased by a large quantity and the best F-measure improved from 50% to 63%. But the performance still lagged behind that obtained using the manually-engineered MRG. The main reason for this is that the manual MRG introduced some domain specific expressions, like `left`, `right`, `left-quarter` etc., which correspond directly to their meanings in the natural language. On the other hand, the only way to specify "left" of a region using the original CLANG MRG is by specifying the coordinates of the left region, like `(rec(pt -32 -35)(pt 0 0))` is the left of `(rec (pt -32 -35) (pt 0 35))` etc. It is not possible to learn the concept of "left" from such expressions even with MRG transformations.

## 6 Conclusions

A meaning representation grammar which does not correspond well with the natural language semantics can lead to a poor performance by a learned semantic parser. This paper presented grammar transformation operators and meaning representation macros using which the meaning representation grammar can be transformed to make it better conform with the semantics of natural language. Experimental results on three different grammars demonstrated that the performance on semantic parsing task can be improved by large amounts by transforming the grammars.

## Acknowledgments

I would like to thank Raymond Mooney for many useful discussions regarding the work described in this paper.

## References

Chen et al. 2003. Users manual: RoboCup soccer server manual for soccer server version 7.07 and later. Available at `http://sourceforge.net/projects/sserver/`.

Ge, R. and R. J. Mooney. 2005. A statistical semantic parser that integrates syntax and semantics. In *Proc. of CoNLL-2005*, pages 9–16, Ann Arbor, MI.

Kate, R. J. and R. J. Mooney. 2006. Using string-kernels for learning semantic parsers. In *Proc. of COLING/ACL-2006*, pages 913–920, Sydney, Australia.

Kate, R. J., Y. W. Wong, and R. J. Mooney. 2005. Learning to transform natural to formal languages. In *Proc. of AAAI-2005*, pages 1062–1068, Pittsburgh, PA.

Langley, Pat and Sean Stromsten. 2000. Learning context-free gramamr with a simplicity bias. In *Proc. of ECML-2000*, pages 220–228, Barcelona, Spain.

Lee, Lillian. 1996. Learning of context-free languages: A survey of the literature. Technical Report TR-12-96, Center for Research in Computing Technology, Harvard University.

Lodhi, Huma, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. 2002. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444.

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

Nguyen, Le-Minh, Akira Shimazu, and Xuan-Hieu Phan. 2006. Semantic parsing with structured SVM ensemble classification models. In *Proc. of COLING/ACL 2006 Main Conf. Poster Sessions*, pages 619–626, Sydney, Australia.

Popescu, Ana-Maria, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *Proc. of IUI-2003*, pages 149–157, Miami, FL.

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

Wong, Y. W. and R. Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proc. of HLT/NAACL-2006*, pages 439–446, New York City, NY.

Wong, Y. W. and R. J. Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proc. of ACL-2007*, pages 960–967, Prague, Czech Republic.

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

Zettlemoyer, Luke S. and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proc. of EMNLP-CoNLL-2007*, pages 678–687, Prague, Czech Republic.