Benchmarking Meaning Representations in Neural Semantic Parsing

Jiaqi Guo1, Qian Liu2, Jian-Guang Lou3, Zhenwen Li4
Xueqing Liu5, Tao Xie4, Ting Liu1

1Xi’an Jiaotong University, Xi’an, China 2Beihang University, Beijing, China
3Microsoft Research, Beijing, China 4Peking University, Beijing, China
5Stevens Institute of Technology, Hoboken, New Jersey, USA

jasperguo2013@stu.xjtu.edu.cn, qian.liu@buaa.edu.cn
jlou@microsoft.com, {lizhenwen, taoxie}@pku.edu.cn
xueqing.liu@stevens.edu, tingliu@mail.xjtu.edu.cn

Abstract

Meaning representation is an important component of semantic parsing. Although researchers have designed a lot of meaning representations, recent work focuses on only a few of them. Thus, the impact of meaning representation on semantic parsing is less understood. Furthermore, existing work’s performance is often not comprehensively evaluated due to the lack of readily-available execution engines. Upon identifying these gaps, we propose UNIMER, a new unified benchmark on meaning representations, by integrating existing semantic parsing datasets, completing the missing logical forms, and implementing the missing execution engines. The resulting unified benchmark contains the complete enumeration of logical forms and execution engines over three datasets \( \times \) four meaning representations. A thorough experimental study on UNIMER reveals that neural semantic parsing approaches exhibit notably different performance when they are trained to generate different meaning representations. Also, program alias and grammar rules heavily impact the performance of different meaning representations. Our benchmark, execution engines and implementation can be found on: https://github.com/JasperGuo/Unimer.

1 Introduction

A remarkable vision of artificial intelligence is to enable human interactions with machines through natural language. Semantic parsing has emerged as a key technology for achieving this goal. In general, semantic parsing aims to transform a natural language utterance into a logic form, i.e., a formal, machine-interpretable meaning representation (MR) (Zelle and Mooney, 1996; Dahl et al., 1994).\(^1\) Thanks to the recent development of neural networks techniques, significant improvements have been made in semantic parsing performance (Jia and Liang, 2016; Yin and Neubig, 2017; Dong and Lapata, 2018; Shaw et al., 2019).

Despite the advancement in performance, we identify three important biases in existing work’s evaluation methodology. First, although multiple MRs are proposed, most existing work is evaluated on only one or two of them, leading to less comprehensive or even unfair comparisons. Table 1 shows the state-of-the-art performance of semantic parsing on different dataset \( \times \) MR combinations, where the rows are the MRs and the columns are the datasets. We can observe that while Lambda Calculus is intensively studied, the other MRs have not been sufficiently studied. This biased evaluation is partly caused by the absence of target logic forms in the missing cells. Second, existing work often compares the performance on different MRs directly (Sun et al., 2020; Shaw et al., 2019; Chen et al., 2020) without considering the confounding edge bases, instead of ungrounded semantic parsing.

<table>
<thead>
<tr>
<th>MR</th>
<th>Geo</th>
<th>ATIS</th>
<th>Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prolog</td>
<td>-</td>
<td>-</td>
<td>91.4</td>
</tr>
<tr>
<td>Lambda</td>
<td>90.4</td>
<td>91.3</td>
<td>85.0</td>
</tr>
<tr>
<td>FunQL</td>
<td>92.5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SQL</td>
<td>78.0</td>
<td>69.0</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1: State-of-the-art performance for MRs on Geo, ATIS, and Job. The top table shows exact-match accuracy whereas the bottom table shows execution-match accuracy. Most existing work focuses on evaluating only a small subset of dataset \( \times \) MR pairs, leaving most of the table unexplored. A finer-grained table is available in the supplementary material (Table 8).

\(^{1}\)Work done during an internship at Microsoft Research.

\(^{2}\)In this paper, we focus on grounded semantic parsing, where meaning representations are grounded to specific knowledge bases. This focus leads to a different evaluation methodology compared to ungrounded semantic parsing.
role that MR plays in the performance,\(^2\) causing unfair comparisons and misleading conclusions. Third, a more comprehensive evaluation methodology would consider both the exact-match accuracy and the execution-match accuracy, because two logic forms can be semantically equivalent yet do not match precisely in their surface forms. However, as shown in Table 1, most existing work is only evaluated with the exact-match accuracy. This bias is potentially due to the fact that execution engines are not available in six out of the twelve dataset \(\times\) MR combinations.

Upon identifying the three biases, in this paper, we propose UNIMER, a new unified benchmark, by unifying four publicly available MRs in three of the most popular semantic parsing datasets: Geo, ATIS and Jobs. First, for each natural language utterance in the three datasets, UNIMER provides annotated logical forms in four different MRs, including Prolog, Lambda Calculus, FunQL, and SQL. We identify that annotated logical forms in some MR \(\times\) dataset combinations are missing. As a result, we complete the benchmark by semi-automatically translating logical forms from one MR to another. Second, we implement six missing execution engines for MRs so that the execution-match accuracy can be readily computed for all the dataset \(\times\) MR combinations. Both the logical forms and their execution results are manually checked to ensure the correctness of annotations and execution engines.

After constructing UNIMER, to obtain a preliminary understanding on the impact of MRs on semantic parsing, we empirically study the performance of MRs on UNIMER by using two widely-used neural semantic parsing approaches (a seq2seq model (Dong and Lapata, 2016; Jia and Liang, 2016) and a grammar-based neural model (Yin and Neubig, 2017)), under the supervised learning setting.

In addition to the empirical study above, we further analyze the impact of two operations, i.e., *program alias* and *grammar rules*, to understand how they affect different MRs differently. First, *Program alias*. A semantically equivalent program may have many syntactically different forms. As a result, if the training and testing data have a difference in their syntactic distributions of logic forms, a naive maximum likelihood estimation can suffer from this difference because it fails to capture the semantic equivalence (Bunel et al., 2018). As different MRs have different degrees of syntactic difference, they suffer from this problem differently. Second, *Grammar rules*. Grammar-based neural models can guarantee that the generated program is syntactically correct (Yin and Neubig, 2017; Wang et al., 2020; Sun et al., 2020). For a given set of logical forms in an MR, there exist multiple sets of grammar rules to model them. We observe that when the grammar-based neural model is trained with different sets of grammar rules, it exhibits a notable performance discrepancy. This finding alias with the one made in traditional semantic parsers (Kate, 2008) that properly transforming grammar rules can lead to better performance of a traditional semantic parser.

In summary, this paper makes the following main contributions:

- We propose UNIMER, a new unified benchmark on meaning representations, by integrating and completing semantic parsing datasets in three datasets \(\times\) four MRs; we also implement six execution engines so that execution-match accuracy can be evaluated in all cases;
- We provide the baseline results for two widely used neural semantic parsing approaches on our benchmark, and we conduct an empirical study to understand the impact that program alias and grammar rule plays on the performance of neural semantic parsing;

2 Preliminaries

In this section, we provide a brief description of the MRs and neural semantic parsing approaches that we study in the paper.

2.1 Meaning Representations

We investigate four MRs in this paper, namely, Prolog, Lambda Calculus, FunQL, and SQL, because they are widely used in semantic parsing and we can obtain their corresponding labeled data in at least one semantic parsing domain. We regard Prolog, Lambda Calculus, and FunQL as domain-specific MRs, since the predicates defined in them are specific for a given domain. Consequently, the execution engines of domain-specific MRs need to be significantly customized for different domains, requiring plenty of manual efforts. In contrast, SQL is a domain-general MR for querying relational
<table>
<thead>
<tr>
<th>MR</th>
<th>Logical Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prolog</td>
<td><code>answer(A, (flight(A), tomorrow(A), during_day(A, B), const(B, period(morning)), from(A, C), const(C, city(Pittsburgh)), to(A, D), const(D, city(Atlanta))))</code></td>
</tr>
<tr>
<td>Lambda Calculus</td>
<td><code>(lambda A:e ( (flight A) ∧ (during_day A morning:pd) ∧ (from A Pittsburgh:ci) ∧ (to A Atlanta:ci) ∧ (tomorrow A) ) )</code></td>
</tr>
<tr>
<td>FunQL</td>
<td><code>answer (flight (tomorrow (intersect (during_day (period (morning))), from (city (Pittsburgh)), to (city (Atlanta)))) )</code></td>
</tr>
<tr>
<td>SQL</td>
<td><code>SELECT flight_id FROM ... WHERE city_1.city_name = 'pittsburgh' AND city_2.city_name = 'atlanta' AND date_day_1.year = 1991 AND date_day_1.month_number = 1 AND date_day_1.day_number = 20 AND departure_time BETWEEN 0 AND 1200</code></td>
</tr>
</tbody>
</table>

Table 2: Examples of meaning representations for utterance “what flights do you have in tomorrow morning from pittsburgh to atlanta?” in the ATIS domain.

Prolog has long been used to represent the meaning of natural language (Zelle and Mooney, 1996; Kate and Mooney, 2006). Prolog includes first-order logical forms, augmented with some higher-order predicates, e.g., most, to handle issues such as quantification and aggregation. Take the first logical form in Tables 2 as an example. The uppercase characters denote variables, and the predicates in the logical form specify the constraints between variables. In this case, character A denotes a variable, and it is required to be a flight, and the flight should depart tomorrow morning from Pittsburgh to Atlanta. The outer predicate answer indicates the variable whose binding is of interest. One major benefit of Prolog-style MRs is that they allow predicates to be introduced in the order where they are actually named in the utterance. For instance, the order of predicates in the logical form strictly follows their mentions in the natural language utterance.

Lambda Calculus is a formal system to express computation. It can represent all first-order logic and it naturally supports higher-order functions. It represents the meanings of natural language with logical expressions that contain constants, quantifiers, logical connectors, and lambda abstract. These properties make it prevalent in semantic parsing. Consider the second logical form in Table 2. It defines an expression that takes an entity A as input and returns true if the entity satisfies the constraints defined in the expressions. Lambda Calculus can be typed, allowing type checking during generation and execution.

FunQL, abbreviated for Functional Query Language, is a variable-free language (Kate et al., 2005). It abstracts away variables and encodes compositionality via its nested function-argument structure, making it easier to implement an efficient execution engine for FunQL. Concretely, unlike Prolog and Lambda Calculus, predicates in FunQL take a set of entities as input and return another set of entities that meet certain requirements. Considering the second logical form in Table 2, the predicate `during_day(period(morning))` returns a set of flights that depart in the morning. With this function-argument structure, FunQL can directly return the entities of interest.

SQL is a popular relational database query language. Since it is domain-agnostic and has well-established execution engines, the subtask of semantic parsing, Text-to-SQL, has received a lot of interests. Compared with domain-specific MRs, SQL cannot encapsulate too much domain prior knowledge in its expressions. As shown in Table 2, to query flights that depart tomorrow, one needs to specify the concrete values of year, month, and day in the SQL query. However, these values are not explicitly mentioned in the utterance and may even change over time.

It is important to note that although these MRs are all expressive enough to represent all meanings in some domains, they are not equivalent in terms of their general expressiveness. For example, FunQL is less expressive than Lambda Calculus in general, partially due to the elimination of variables and quantifiers.

2.2 Neural Semantic Parsing Approaches

During the last few decades, researchers have proposed different approaches for semantic parsing. Most state-of-the-art approaches are based on neural models and formulate the semantic parsing problem as a sequence transduction problem. Due to the generality of sequence transduction, these ap-
proaches can be trained to generate any MRs. In this work, without loss of generality, we benchmark MRs by evaluating the seq2seq model (Dong and Lapata, 2016; Jia and Liang, 2016) and the grammar-based model (Yin and Neubig, 2017) under the supervised learning setting. We select the two models because most neural approaches are designed based on them.

**Seq2Seq Model.** Dong and Lapata (2016) and Jia and Liang (2016) formulated the semantic parsing problem as a neural machine translation problem and employed the sequence-to-sequence model (Sutskever et al., 2014) to solve it. As illustrated in Figure 1a, the encoder takes an utterance as input and outputs a distributed representation for each word in the utterance. A decoder then sequentially predicts words in the logical form. When augmented with the attention mechanism (Bahdanau et al., 2014; Luong et al., 2015), the decoder can better utilize the encoder’s information to predict logical forms. Moreover, to address the problem caused by the long tail distribution of entities in logical forms, Jia and Liang (2016) proposed an attention-based copying mechanism. That is, at each time step, the decoder takes one of two types of actions, one to predict a word from the vocabulary of logical forms and the other to copy a word from the input utterance.

**Grammar-based Model.** By treating a logical form as a sequence of words, the seq2seq model cannot fully utilize the property that logical forms are well-formed and must conform to certain grammars of an MR. To bridge this gap, Yin and Neubig (2017) proposed a grammar-based decoder that outputs a sequence of grammar rules instead of words, as presented in Figure 1b. The decoded grammar rules can deterministically generate a valid abstract syntax tree (AST) of a logical form. In this way, the generated logical form is guaranteed to be syntactically correct. This property makes it widely used in a lot of code generation and semantic parsing tasks (Sun et al., 2020; Wang et al., 2020; Bogin et al., 2019). The grammar-based decoder can also be equipped with the attention-based copying mechanism to address the long-tail distribution problem.

### 3 Benchmark

To provide an infrastructure for exploring MRs, we construct UNIMER, a unified benchmark on MRs, based on existing semantic parsing datasets. Currently, UNIMER covers three domains, namely Geo, ATIS, and Job, each of which has been extensively studied in previous work and has annotated logical forms for at least two MRs. All natural language utterances in UNIMER are written in English.

**Geo** focuses on querying a database of U.S. geography with natural language. To solve the problem, Zelle and Mooney (1996) designed a Prolog-style MR and annotated 880 (utterance, logical form) pairs. Popescu et al. (2003) and Kate et al. (2005) proposed to use SQL and FunQL to represent the meanings, respectively. Almost the same time, Zettlemoyer and Collins (2005) proposed to use Lambda Calculus and manually converted the Prolog logical forms to equivalent expressions in Lambda Calculus. Following their work, we adopt the standard 600/280 training/test split.

**ATIS** is a dataset of flight booking questions. It consists of 5,418 questions and their corresponding SQL queries. Zettlemoyer and Collins (2007) proposed to use Lambda Calculus to represent the meanings of natural language and automatically map these SQL queries to its equivalent logical forms in Lambda Calculus. Following the work of Kwiatkowski et al. (2011), we use the standard 4480/480/450 training/dev/test split.

**Job** is a dataset about job announcements posted in the newsgroup austin.jobs (Califf and Mooney, 1999). It consists of 640 utterances and their corresponding Prolog logical forms that query computer-related job postings. Similarly, in Geo, Zettlemoyer...
and Collins (2005) proposed using Lambda Calculus and manually converted them to equivalent expressions in Lambda Calculus. We use the same training-test split with them, containing 500 training and 140 test instances.

Since not all the four MRs that we introduce in Section 2.1 are used in these three domains, we semi-automatically translate logical forms in one MR into another. This effort enables researchers to explore MRs in more domains and make a fair comparison among them. Take the translation of Lambda Calculus to FunQL in ATIS as an example. We first design predicates for FunQL based on those defined in Lambda Calculus and implement an execution engine for FunQL. Then, we translate logical forms in Lambda Calculus to FunQL and compare the execution results to verify the correctness of the translation. In this process, we find that there is no ready-to-use Lambda Calculus execution engine for the three domains. Hence, we implement one for each domain. These engines, on the one hand, enable evaluations of semantic parsing approaches with both exact-match accuracy and execution-match accuracy. On the other hand, they enable exploration of weakly supervised semantic parsing with Lambda Calculus. In addition, we find some annotation mistakes in logical forms and several bugs in existing execution engines of Prolog and FunQL. By correcting the mistakes and fixing the bugs in the engines, we create a refined version of these datasets. Section A.1 in the supplementary material provides more details about the construction process.

We plan to cover more domains and more MRs in UNIMER. We have made UNIMER along with the execution engines publicly available.3 We believe that UNIMER can provide fertile soil for exploring MRs and addressing challenges in semantic parsing.

4 Experimental Setup

Based on UNIMER, we take the first attempt to study the characteristics of different MRs and their impact on neural semantic parsing.

4.1 Experimental Design

Meaning Representation Comparison. To understand the impact of MRs on neural semantic parsing, we first experiment with the two neural approaches described in Section 2.2 on UNIMER, and we compare the resulting performance of different MRs with two metrics: exact-match accuracy (a logical form is regarded as correct if it is syntactically identical to the gold standard),4 and execution-match accuracy (regarded as correct if a logical form’s execution result is identical to that of the gold standard).5

Program Alias. To explore the effect of program alias, we replace a different proportion of logical forms in a training set with their aliases (semantically equivalent but syntactically different logical forms), and we re-train the neural approaches to quantify its effect. To search for aliases of a logical form, we first derive multiple transformation rules for each MR. Then, we apply these rules to the logical form to get its aliases and randomly sample one. We compare the execution results of the resulting logical forms to ensure their equivalence in semantics. Table 3 presents three transformation rules for SQL. We provide a detailed explanation of transformation rules and examples for each MR in Section A.3 of the supplementary material.

Grammar Rules. To understand the grammar rules’ impact on grammar-based models, we provide two sets of grammar rules for each MR. Each set of rules can cover all the logical forms in the three domains. We compare the performance of models trained with different sets of rules. Specifically, Wong and Mooney (2006) and Wong and Mooney (2007) have induced a set of grammar rules for Prolog and FunQL in Geo. We directly use them in Geo and extend them to support logical forms in ATIS and Job. As for SQL, Bogin et al. (2019) have induced a set of rules for SQL in the Spider benchmark, and we adapt it to support the SQL queries in the three domains that we study.

---

3Our benchmark, execution engines and our implementation can be found on: https://github.com/JasperGuo/Unimer

4Following Dong and Lapata (2016), we sort the sub-expressions in the conjunction predicate of Lambda Calculus before comparison.

5It should be acknowledged that using the execution-match accuracy to evaluate a parser is not enough, as there exist spurious programs that lead to the same execution results with the gold standard but have different semantics.
When it comes to Lambda Calculus, we use the one induced by Yin and Neubig (2018). For comparison, we also manually induce another set of grammar rules for the four MRs. Section A.4 in the supplementary material provides definitions of all the grammar rules.

4.2 Implementations

We implement each approach with the AllenNLP (Gardner et al., 2018) and PyTorch (Paszke et al., 2019) frameworks. To make a fair comparison, we tune the hyper-parameters of approaches for each MR on the development set or through cross-validation on the training set, with the NNI platform.6 Due to the limited number of test data in each domain, we run each approach five times and take the average number. Section A.2 in the supplementary material provides the search space of hyper-parameters for each approach and the preprocessing procedures of logical forms.

Multiple neural semantic parsing approaches (Dong and Lapata, 2016; Iyer et al., 2017; Rabinovich et al., 2017) adopt the data anonymization techniques to replace entities in utterances with placeholders. However, the techniques are usually ad-hoc and specific for domains and MRs, and they sometimes require manual efforts to resolve conflicts (Finegan-Dollak et al., 2018). Hence, we do not apply data anonymization to avoid bias.

5 Experimental Results

5.1 Meaning Representation Comparison

Table 4 presents our experimental results on UNIMER. Since we do not use data anonymization techniques, the performance is generally lower than that shown in Table 1 and Table 8, but the performance is on par with the numbers reported in ablation studies of previous work (Dong and Lapata, 2016; Jia and Liang, 2016; Finegan-Dollak et al., 2018). We can make the following three observations from the table.

First, neural approaches exhibit notably different performance when they are trained to generate different MRs. The difference can vary by as much as 20% in both exact-match and execution-match metrics. This finding tells us that an apple-to-apple comparison is extremely important when comparing two neural semantic parsing approaches. However, we notice that some papers (Sun et al., 2020; Shaw et al., 2019; Chen et al., 2020) do not clearly note the MRs used in baselines and compare with them using different metrics; the attained result can be somewhat misleading.

Second, domain-specific MRs (Prolog, Lambda Calculus, and FunQL) tend to outperform SQL (domain-general) by a large margin. For example, in Geo, the execution-match accuracy of FunQL is substantially higher than that of SQL in all approaches. This result is expected because a lot of domain knowledge is injected into domain-specific MRs. Consider the logical forms in Table 2. There is a predicate tomorrow in all three domain-specific MRs, and this predicate can directly align to the description in the utterance. However, one needs to explicitly express the concrete date values in the SQL query; this requirement can be a heavy burden for neural approaches, especially when the values will change over time. In addition, a recent study (Finegan-Dollak et al., 2018) in Text-to-SQL has shown that domain-specific MRs are more robust against generating never-seen logical forms than SQL, because their surface forms are much closer to natural language.

Third, among all the domain-specific MRs, FunQL tends to outperform the others in neural approaches. In Geo, FunQL outperforms the other MRs in both metrics by a large margin. In Job, the grammar-based (w/ copy) model trained with FunQL achieves the state-of-the-art performance. One possible reason is that FunQL is more compact than the other MRs, due to its elimination of variables and quantifiers. Figure 2 shows box plots about the number of grammars rules in the AST of a logical form. We can observe that while FunQL has almost the same number of grammar rules with the other MRs (Table 5), it has much fewer grammar rules involved in a logical form than the others.

Figure 2: Statistics of logical forms in the training set of Prolog (P), Lambda Calculus (L), FunQL (F) and SQL (S). The y-axis indicates the number of production rules in each logical form.
Table 4: Experimental results of the seq2seq model and the grammar-based model. We highlight the best performance of each approach and MR in both metrics. The poor SQL exact-match accuracy in ATIS is caused by the different distribution of SQL queries in test set. Iyer et al. (2017) rewrote the SQL queries in test set to improve their execution efficiency.

Table 5: Statistics of logical forms in different MRs. ‘# Vo’ and ‘# Ru’ indicate the vocabulary size and the number of grammar rules of an MR, respectively.

Table 6: Relative decline of performance when 25% of logical forms in training data are replaced with aliases.

Figure 3: Execution-match accuracy of the seq2seq (w/ copy) model when different proportions of logical forms in training set are replaced with their aliases.

5.2 Program Alias

Figure 3 shows the execution-match accuracy of the seq2seq (w/ copy) model with different proportions of logical forms in the training data replaced with aliases. Since not all the logical forms have aliases, the curves in Figure 3 stop at different points. Among all the domain-specific MRs, FunQL has the fewest logical forms with aliases, while Prolog has the largest. For example, in Geo, only 42% of FunQL logical forms in the training set have aliases, while more than 70% of logical forms in Lambda Calculus and Prolog have aliases.

---

²We have experimented with the grammar-based model and observed similar results.
We find that the exact-match accuracy declines which could be extremely tedious. Second, the experimental results of Table 7 presents the execution-match accuracy of the grammar-based model trained with G2 for Prolog outperforms the program alias problem more seriously. The trends of Lambda Calculus and FunQL in ATIS are impressive, as their performance decreases only slowly. Selecting an MR with the less effect of program alias could be a better choice when we need to develop a semantic parser for a new domain, because we can save many efforts in defining annotation protocols and checking consistency, which could be extremely tedious. Second, the exact-match accuracy declines more seriously than execution-match. Table 6 provides the relative declines in both exact-match and execution-match metrics when 25% of logical forms are replaced. We find that the exact-match accuracy declines more seriously than execution-match, indicating that under the effect of program alias, exact-match may not be suitable as it may massively underestimate the performance. At last, given a large number of semantically equivalent logical forms, it would be valuable to explore whether they can be leveraged to improve semantic parsing (Zhong et al., 2018).

### 5.3 Grammar Rules

Table 7 presents the experimental results of the grammar-based (w/o Copy) model trained with different sets of grammar rules. As the table shows, there is a notable performance discrepancy between different sets of rules. For example, in ATIS, we can observe 2.5% absolute improvement when the model is trained with G2 for Lambda Calculus. Moreover, G2 is not always better than G1. While the model trained with G2 for Prolog outperforms G1 in Geo, it lags behind G1 in ATIS. This observation motivates us to consider what factors contribute to the discrepancy. We had tried to explore the search space of logical forms defined by different grammar rules and the distribution drift between the AST of logical forms in the training and test set. However, the exploration results cannot consistently explain the performance discrepancy. As our important future work, we would explore whether or not the discrepancy is caused by better alignments between utterances and grammar rules. Intuitively, it would be easier for decoders to learn the set of grammar rules having better alignments with utterances.

We can learn from these results that similar to traditional semantic parsers, properly transforming grammar rules for MRs can also lead to better performance in neural approaches. Therefore, grammar rules should be considered as a very important hyper-parameter of grammar-based models, and it is recommended to mention the used grammar rules in research papers clearly.

## 6 Related Work

### Meaning representations for semantic parsing

Recent work has shown that properly designing new MRs often helps improve the performance of neural semantic parsing. Except for the four MRs that we study, Liang et al. (2011); Liang (2013) presented DCS and lambda DCS for querying knowledge bases and demonstrated their advantages over Lambda Calculus. Guo et al. (2019) proposed SemQL, an MR for relational database queries, and they showed improvements in the Spider benchmark. Wolfson et al. (2020) designed an MR, named QDMR, for representing the meaning of questions through question decomposition. Instead of designing new MRs, Cheng et al. (2019) proposed a transition-based semantic parsing approach that supports generating tree-structured logical forms in either top-down or bottom-up manners. Their experimental results on various semantic parsing datasets using FunQL showed that top-down generation outperforms bottom-up in various settings. Our work aims to investigate the characteristics of different MRs and their impact on neural semantic parsing.

**Ungrounded semantic parsing.** Except for the grounded MRs studied in this work, there are also ungrounded MRs that are not tied to any particular applications, such as AMR (Banarescu et al.,...
2013), MRS (Copestake et al., 2005; Flickinger et al., 2017), and UCCA (Abend and Rappoport, 2013). Abend and Rappoport (2017) conducted a survey on ungrounded MRs to assess their achievements and shortcomings. Hershcovich et al. (2019) evaluated the similarities and divergences in the content encoded by ungrounded MRs and syntactic representation. Lin and Xue (2019) carried out a careful analysis on AMR and MRS to understand the factors contributing to the discrepancy in their parsing accuracy. Partly inspired by this line of work, we conduct a study on grounded MRs and investigate their impact on neural semantic parsing. Hershcovich et al. (2018) proposed to leverage annotated data in different ungrounded MRs to improve parsing performance. With UNIMER, we can explore whether it is feasible in grounded semantic parsing.

Extrinsic parser evaluation. Another line of research that is closely related to our work is extrinsic parser evaluation. Miyao et al. (2008) benchmarked different syntactic parsers and their representations, including dependency parsing, phrase structure parsing, and deep parsing, and evaluated their impact on an information extraction system. Oepen et al. (2017) provided a flexible infrastructure, including data and software, to estimate the relative utility of different types of dependency representations for a variety of downstream applications that rely on an analysis of grammatical structure of natural language. There has not been work on benchmarking MRs for grounded semantic parsing in neural approaches, to the best of our knowledge.

Weakly supervised semantic parsing. In this paper, we focus on supervised learning for semantic parsing, where each utterance has its corresponding logical form annotated. But the similar evaluation methodology could be applied to weakly supervised semantic parsing, which receives wide attention because parsers are only supervised with execution results and annotated logical forms are no longer required (Berant et al., 2013; Pasupat and Liang, 2015; Goldman et al., 2018; Liang et al., 2018; Mueller et al., 2019). We also notice that various MRs have been used in weakly supervised semantic parsing, and it would be valuable to explore the impact of MRs in such settings.

7 Conclusion

In this work, we propose UNIMER, a unified benchmark on meaning representations, based on established semantic parsing datasets; UNIMER covers three domains and four different meaning representations along with their execution engines. UNIMER allows researchers to comprehensively and fairly evaluate the performance of their approaches. Based on UNIMER, we conduct an empirical study to understand the characteristics of different meaning representations and their impact on neural semantic parsing. By open-sourcing our source code and benchmark, we believe that our work can facilitate the community to inform the design and development of next-generation MRs.

Implications. Our findings have clear implications for future work. First, according to our experimental results, FunQL tends to outperform Lambda Calculus and Prolog in neural semantic parsing. Additionally, FunQL is relatively robust against program alias. Hence, when developers need to design an MR for a new domain, FunQL is recommended to be the first choice. Second, to reduce program alias’ negative effect on neural semantic parsing, developers should define a concrete protocol for annotating logical forms to ensure their consistency. Specifically, given an MR, developers should identify as many as possible sources where program alias can occur. Take SQL as an example. To express the $\text{argmax}$ semantics, one can either use subquery or the OrderBy clause.\footnote{A concrete example is provided in Section A.3 in the supplementary material.}

Having identified these sources, developers need to determine using which expression in what context, e.g., $\text{argmax}$ is always expressed with subquery, and the unordered expressions in conjunctions are always sorted by characters.

Acknowledgments

We would like to thank Bei Chen for helpful discussions and thank the anonymous reviewers for their constructive comments. Ting Liu was partially supported by NSFC (61632015, 61721002) and State Grid R&D Program (5226SX1800FC).

References

Omri Abend and Ari Rappoport. 2013. UCCA: A semantics-based grammatical annotation scheme. In Proceedings of the 10th International Conference...
on Computational Semantics, pages 1–12, Potsdam, Germany. Association for Computational Linguistics.


A Supplemental Material

Algorithm 1: Translation of Lambda Calculus Logical Forms to FunQL

Input: A Lambda Calculus logical form p
Output: A FunQL logical form f

1. types = InferTypes(p);
2. tokens = Tokenize(p);
3. expressions, stack = [], [];
4. i = 0;
5. for t ∈ tokens do
   6. if t == ‘(’ then
      7. Append(stack, i);
   8. else if t == ‘)’ then
      9. j = Pop(stack);
     10. e = Search(expressions, j, i);
     11. Remove(e);
     12. ne = Translate(tokens[j : i], e, types);
     13. Append(expressions, ne);
     14. i+ = 1;
   end
15. f = expressions[0];

A.1 Details of Benchmark Construction

Geo. Since the four MRs we study have annotated logical forms and have execution engines except for Lambda Calculus in Geo, we directly use them (SQL taken from (Finegan-Dollak et al., 2018), Prolog taken from (Jia and Liang, 2016), FunQL taken from (Wong and Mooney, 2006), and Lambda Calculus taken from (Kwiatkowski et al., 2010)). We implement an execution engine with Haskell for Lambda Calculus. With these resources, we cross-validate the correctness of annotations and execution engines by comparing the execution results of logical forms. As a result, we found nearly 30 Prolog logical forms with annotation mistakes and two bugs in the execution engines of Prolog and FunQL.

ATIS. There are only annotated logical forms for Lambda Calculus and SQL in ATIS. We directly use Lambda Calculus logical forms provided by (Jia and Liang, 2016) and SQL queries provided by (Iyer et al., 2017). To provide annotations for FunQL and Prolog, we semi-automatically translate Lambda Calculus logical forms to equivalent logical forms in FunQL and Prolog. Algorithm 1 presents the pseudo code for translating a Lambda Calculus logical form to FunQL. Basically, we first
Table 8: Performance of semantic parsing approaches, where * denotes neural approaches and † denotes approaches using data anonymization techniques. ’λ’ denotes Lambda Calculus. ‘Exact’ and ‘Exec’ denote exact-match and execution-match, respectively.

<table>
<thead>
<tr>
<th>Approach</th>
<th>MR</th>
<th>Exact</th>
<th>Exec</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Geo</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kwiatkowski et al. (2010)</td>
<td>FunQL</td>
<td>84.3</td>
<td>-</td>
</tr>
<tr>
<td>Shaw et al. (2019)*</td>
<td>FunQL</td>
<td>89.3</td>
<td>-</td>
</tr>
<tr>
<td>Jia and Liang (2016)*</td>
<td>Prolog</td>
<td>-</td>
<td>89.3</td>
</tr>
<tr>
<td>- data augmentation</td>
<td>Prolog</td>
<td>-</td>
<td>85.0</td>
</tr>
<tr>
<td>Chen et al. (2020)*</td>
<td>Prolog</td>
<td>-</td>
<td>89.6</td>
</tr>
<tr>
<td>Kwiatkowski et al. (2010)</td>
<td>λ</td>
<td>87.9</td>
<td>-</td>
</tr>
<tr>
<td>Wang et al. (2014)</td>
<td>λ</td>
<td>90.4</td>
<td>-</td>
</tr>
<tr>
<td>Zhao and Huang (2015)</td>
<td>λ</td>
<td>88.9</td>
<td>-</td>
</tr>
<tr>
<td>Dong and Lapata (2016)*†</td>
<td>λ</td>
<td>87.1</td>
<td>-</td>
</tr>
<tr>
<td>Rabinovich et al. (2017)*†</td>
<td>λ</td>
<td>87.1</td>
<td>-</td>
</tr>
<tr>
<td>Dong and Lapata (2018)*†</td>
<td>λ</td>
<td>88.2</td>
<td>-</td>
</tr>
<tr>
<td>Sun et al. (2020)*†</td>
<td>λ</td>
<td>89.1</td>
<td>-</td>
</tr>
<tr>
<td>Iyer et al. (2017)*†</td>
<td>SQL</td>
<td>-</td>
<td>82.5</td>
</tr>
<tr>
<td><strong>ATIS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wang et al. (2014)</td>
<td>λ</td>
<td>91.3</td>
<td>-</td>
</tr>
<tr>
<td>Zhao and Huang (2015)</td>
<td>λ</td>
<td>84.2</td>
<td>-</td>
</tr>
<tr>
<td>Jia and Liang (2016)*</td>
<td>λ</td>
<td>83.3</td>
<td>-</td>
</tr>
<tr>
<td>- data augmentation</td>
<td>λ</td>
<td>76.3</td>
<td>-</td>
</tr>
<tr>
<td>Dong and Lapata (2016)*†</td>
<td>λ</td>
<td>84.6</td>
<td>-</td>
</tr>
<tr>
<td>Rabinovich et al. (2017)*†</td>
<td>λ</td>
<td>85.9</td>
<td>-</td>
</tr>
<tr>
<td>Yin and Neubig (2018)*†</td>
<td>λ</td>
<td>88.2</td>
<td>-</td>
</tr>
<tr>
<td>Dong and Lapata (2018)*†</td>
<td>λ</td>
<td>87.7</td>
<td>-</td>
</tr>
<tr>
<td>Cao et al. (2019)*†</td>
<td>λ</td>
<td>89.1</td>
<td>-</td>
</tr>
<tr>
<td>Sun et al. (2020)*†</td>
<td>λ</td>
<td>89.6</td>
<td>-</td>
</tr>
<tr>
<td>Shaw et al. (2019)*†</td>
<td>λ</td>
<td>87.1</td>
<td>-</td>
</tr>
<tr>
<td>Iyer et al. (2017)*†</td>
<td>SQL</td>
<td>-</td>
<td>79.2</td>
</tr>
<tr>
<td><strong>Jobs</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zettlemoyer and Collins (2005)</td>
<td>λ</td>
<td>79.3</td>
<td>-</td>
</tr>
<tr>
<td>Zhao and Huang (2015)</td>
<td>λ</td>
<td>85.0</td>
<td>-</td>
</tr>
<tr>
<td>Dong and Lapata (2016)*†</td>
<td>Prolog</td>
<td>90.0</td>
<td>-</td>
</tr>
<tr>
<td>Rabinovich et al. (2017)*†</td>
<td>Prolog</td>
<td>91.4</td>
<td>-</td>
</tr>
<tr>
<td>Chen et al. (2020)*†</td>
<td>Prolog</td>
<td>-</td>
<td>92.1</td>
</tr>
</tbody>
</table>

perform a type inference for variables in the logical form, since some predicates in Lambda Calculus can take different types of variables as input, e.g., for the predicate \( \text{to}(A, B) \), variable \( B \) can be either an airport or a city. Then, we tokenize the input logical form and recursively translate each sub-expression in the logical form into FunQL expressions based on the predicates we design for FunQL. Translation from Lambda Calculus to Prolog is performed in a similar way. The source code for translation is also available in our Github repository. We also implement execution engines for Lambda Calculus, Prolog, and FunQL.

**Job.** There are only annotated logical forms for Lambda Calculus and Prolog in Job. We directly use Prolog logical forms provided on the website.\(^9\) To provide annotations for FunQL, we semi-

\(^9\)http://www.cs.utexas.edu/ml/mldata/jobquery.html automatically translate Prolog logical forms to equivalent logical forms in FunQL with an algorithm similar to Algorithm 1. In terms of SQL, the logical forms in Job are relatively simple and can be expressed with the `SELECT`, `FROM` and `WHERE` clauses of SQL. We simply translate each sub-expression in Prolog to an expression in `WHERE` clause and use a conjunction to join the resulting expressions. Since we cannot find the annotated Lambda Calculus logical forms provided by (Zettlemoyer and Collins, 2005), we also translate the Prolog logical forms to Lambda Calculus. We implement execution engines for Lambda Calculus and FunQL.

A.2 Model Configuration

**Preprocessing of Prolog and Lambda Calculus**

As Prolog and Lambda Calculus have variables, we need to standardize their variable naming before training. Following (Jia and Liang, 2016), we preprocess the Prolog logical forms to the De Brujin index notation. We standardize the variable naming of Lambda Calculus based on their occurrence order in logical forms.

**Attention Copying Mechanism.** In Geo and Job, we use the standard copy mechanism, i.e., directly copying a source word to a logical form. In ATIS, following (Jia and Liang, 2016), we leverage an external lexicon to identify potential copy candidates, e.g., `slc:ap` can be identified as a potential entity for description “salt lake city airport” in utterance. When we copy a source word that is part of a phrase in the lexicon, we write the entity associated with that lexicon entry to a logical form.

**Hyper-Parameters.** For the seq2seq model, the embedding dimension of both source and target languages ranges over \{100, 200\}. We select a one-layer bi-directional LSTM as an encoder. The hidden dimension of the encoder ranges over \{32, 64, 128, 256\}. Similarly, a one-layer LSTM is selected as the decoder. Its hidden dimension is as same as the encoder. In terms of attention, we select bi-linear as the activation function, where the hidden dimension is 2 times that of the encoder. We employ dropout at training time with rate ranging over \{0.1, 0.2, 0.3\}. We select batch size from \{16, 32, 48, 64\}, and select learning rate from \{0.001, 0.0025, 0.005, 0.01, 0.025, 0.05\}. Following (Dong and Lapata, 2016), we use the RMSProp algorithm to update the parameters. The smoothing constant of RMSProp is 0.95. We initialize all
parameters uniformly at random within the interval $[-0.1, 0.1]$.

Similarly, for the grammar-based model, a one-layer bi-directional LSTM is used as an encoder and another LSTM is employed as a decoder. The layers of the decoder is selected from $\{1, 2\}$. The hidden dimension of the encoder ranges over $\{64, 128, 256\}$. The hidden dimension of the decoder is 2 times that of the encoder. The hidden dimension of both the grammar rule and non-terminal is selected from $\{64, 128, 256\}$. We also employ dropout in the encoder and decoder at training time with rate selected from $\{0.1, 0.2, 0.3\}$. We select batch size from $\{16, 32, 48, 64\}$, and select learning rate from $\{0.001, 0.0025, 0.005, 0.01, 0.025, 0.05\}$. We use the Adam algorithm to update the parameters.

For both models, gradients are clipped at 5 to alleviate the exploding gradient problem, and early stopping is used to determine the number of epochs. We provide the detailed configurations of the NNI platform in our Github repository.

### A.3 Search for Aliases for Logical Forms.

#### Algorithm 2: Search for Program Alias

**Input:** A logical form $p$  
**Output:** A set of aliases of the input logical form $\text{aliases}$

1. $\text{ast} = \text{Parse}(p)$;
2. $\text{aliases} = []$;
3. for $r \in \text{Rules}$ do
4. $\text{C} = \text{Apply}(\text{ast}, r)$;
5. for $c \in \text{C}$ do
6. if $\text{IsEquivalent}(p, c)$ then
7. $\text{Append(aliases, c)}$;
8. end
9. end

Algorithm 2 presents the way we search for aliases for a logical form. Transformation rules can be categorized into two groups based on whether they are domain-specific. Considering the following two logical forms:

- $(\lambda A : e (\exists B \ (\text{flight} B \ \text{fare} B A)))$
- $(\lambda A : e (\exists B \ (\text{flight} B) \ (\text{fare} B A)))$

they are semantically equivalent due to the multiple meaning definitions of `fare`. There are also domain-general transformation rules, e.g., permuting the expressions in the conjunction predicate:

- $(\lambda A : e (\exists B \ (\text{flight} B) \ (\text{fare} B A)))$
- $(\lambda A : e (\exists B \ (\text{fare} B A) \ (\text{flight} B)))$

In this work, we primarily consider domain-general transformation rules and only when there is limited aliases found by domain-general rules, we use domain-specific rules. Table 9 presents the transformation rules we used in Geo domains. Rules in ATIS are similar. We provide examples below to illustrate the rules.

#### Prolog Shuffle:

- $\text{answer}(A,(\text{area}(B,A),\text{const}(B,\text{stateid}(\text{Texas}))))$
- $\text{answer}(A,(\text{const}(B,\text{stateid}(\text{texas}))),(\text{area}(B,A)))$

#### Prolog Remove:

- $\text{answer}(\text{loc}(B,A),\text{city}(B),\text{const}(B,\text{cityid}(\text{Austin}_\text{r})))$
- $\text{answer}(\text{loc}(B,A),\text{const}(B,\text{cityid}(\text{Austin}_\text{l})))$

#### Prolog Merge:

- $\text{answer}(\text{state}(A),\text{loc}(B,A),\text{highest}(B,\text{place}(B)))$
- $\text{answer}(\text{highest}(B,(\text{place}(B),\text{loc}(B,A),\text{state}(A))))$

#### FunQL Swap:

- $\text{answer}(\text{count}(\text{major}(\text{city}(\text{loc}_2\text{(stateid}(\text{Texas}))))))$
- $\text{answer}(\text{count}(\text{city}(\text{major}(\text{loc}_2\text{(stateid}(\text{Texas}))))))$

#### FunQL Replace:

- $\text{answer}(\text{city}(\text{loc}_2\text{(largest}(\text{state}(\text{all})))))$
- $\text{answer}(\text{city}(\text{loc}(\text{2}(\text{largest}\text{one}(\text{area}\text{1}(\text{state}(\text{all}))))))$

#### SQL Argmax:

- $\text{SELECT city.city\_name FROM city WHERE city.population = (SELECT MAX(c1.population)\ FROM city as c1 WHERE c1.state\_name = ‘arizona’)}$ and $\text{city.state\_name = ‘arizona’};$

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prolog</strong></td>
<td></td>
</tr>
<tr>
<td>Shuffle</td>
<td>Shuffle expressions in conjunction</td>
</tr>
<tr>
<td>Remove</td>
<td>Remove redundant expressions</td>
</tr>
<tr>
<td>Merge</td>
<td>Put expressions into higher-order predicates</td>
</tr>
<tr>
<td><strong>Lambda Calculus</strong></td>
<td></td>
</tr>
<tr>
<td>Shuffle</td>
<td>Shuffle expressions in conjunction</td>
</tr>
<tr>
<td>Replace</td>
<td>Replace semantically equivalent predicates</td>
</tr>
<tr>
<td><strong>FunQL</strong></td>
<td></td>
</tr>
<tr>
<td>Shuffle</td>
<td>Shuffle expressions in conjunction</td>
</tr>
<tr>
<td>Remove</td>
<td>Remove redundant expressions</td>
</tr>
<tr>
<td>Swap</td>
<td>Swap unit relation predicate</td>
</tr>
<tr>
<td>Replace</td>
<td>Replace semantically equivalent predicates</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td></td>
</tr>
<tr>
<td>Shuffle</td>
<td>Shuffle expressions in Select, From, Where, and Having clauses</td>
</tr>
<tr>
<td>Argmax</td>
<td>Express Argmax/min with OrderBy and Limit clause instead of subquery</td>
</tr>
<tr>
<td>In2Join</td>
<td>Replace In clause with Join clause</td>
</tr>
</tbody>
</table>

Table 9: Transformation rules for Prolog, Lambda Calculus and FunQL.
<table>
<thead>
<tr>
<th>MR</th>
<th>Geo</th>
<th>ATIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prolog</td>
<td>98.0%</td>
<td>96.2%</td>
</tr>
<tr>
<td>Lambda Calculus</td>
<td>69.7%</td>
<td>95.3%</td>
</tr>
<tr>
<td>FunQL</td>
<td>42.5%</td>
<td>87.9%</td>
</tr>
<tr>
<td>SQL</td>
<td>38.5%</td>
<td>95.5%</td>
</tr>
</tbody>
</table>

Table 10: Number of logical forms in training set of Geo and ATIS that have aliases.

SELECT city.city\_name FROM city WHERE state\_name = ‘arizona’ ORDER BY city\_population DESC LIMIT 1;

**SQL In2Join:**

SELECT river.river\_name FROM river WHERE river\_traverse IN (SELECT state.state\_name FROM state WHERE state\_area = (SELECT MAX(s1.area) FROM state as s1));

Table 10 presents the number of logical forms in the training set of Geo and ATIS that have aliases, from which we can see that with these simple rules, we are able to find a lot of aliases. Since the logical forms in Geo are generally simpler than that of ATIS, the number of Geo is lower than ATIS.

A.4 Grammar

We present the grammar rules used in our experiments for Geo (Figure 4-11). The grammar rules for different MRs in ATIS follow a similar definition. It is important to note that although all logical forms in an MR can be modeled by both G1 and G2, they are not necessarily equivalent in expressiveness. For example, `answer(largest(place(all)))` is a syntactically correct but semantically incorrect FunQL logical form in Geo, because the predicate `largest` is not allowed to take places as input. However, G2 of FunQL still can accept this logical form, while G1 cannot, because G1 explicitly encodes type constraints in its rules. Nevertheless, the performance of G1 is lower than G2 in both Geo and ATIS domains, which is surprising. For all the grammar rules we present, G1 and G2 for Prolog and Lambda Calculus and equivalent. G1 and G2 for FunQL and SQL are not equivalent.

Figure 4: The grammar rules for Prolog in Geo induced by (Wong and Mooney, 2007) (G1).
Figure 5: The grammar rules that we induce for Prolog in Geo (G2).

Figure 6: The grammar rules that for Lambda Calculus in Geo (G1).
Figure 9: The grammar rules that we induce for FunQL in Geo (G2).
Figure 10: The grammar rules that we adapt from (Bogin et al., 2019) for SQL Geo (G1).
Figure 11: The grammar rules that we induce for SQL Geo (G2).