Learning to Synthesize Data for Semantic Parsing

Bailin Wang[†*]   **Wenpeng Yin**[‡]   **Xi Victoria Lin**[§*] and **Caiming Xiong**[‡]

[†] University of Edinburgh, [‡] Salesforce Research, [§] Facebook AI

bailin.wang@ed.ac.uk, {wyin,cxiong}@salesforce.com, victorialin@fb.com

Abstract

Synthesizing data for semantic parsing has gained increasing attention recently. However, most methods require handcrafted (high-precision) rules in their generative process, hindering the exploration of diverse unseen data. In this work, we propose a generative model which features a (non-neural) PCFG that models the composition of programs (e.g., SQL), and a BART-based translation model that maps a program to an utterance. Due to the simplicity of PCFG and pre-trained BART, our generative model can be efficiently learned from existing data at hand. Moreover, explicitly modeling compositions using PCFG leads to a better exploration of unseen programs, thus generate more diverse data. We evaluate our method in both in-domain and out-of-domain settings of text-to-SQL parsing on the standard benchmarks of GEOQUERY and SPIDER, respectively. Our empirical results show that the synthesized data generated from our model can substantially help a semantic parser achieve better compositional and domain generalization.

1 Introduction

Recently, synthesizing data for semantic parsing has gained increasing attention (Yu et al., 2018a, 2020; Zhong et al., 2020). However, these models require handcrafted rules (or templates) to synthesize new programs or utterance-program pairs. This can be sub-optimal as fixed rules cannot capture the underlying distribution of programs which usually vary across different domains (Herzig and Berant, 2019). Meanwhile, designing such rules also requires human involvement with expert knowledge. To alleviate this, we propose to learn a generative model from the existing data at hand. Our key observation is that programs (e.g., SQL)
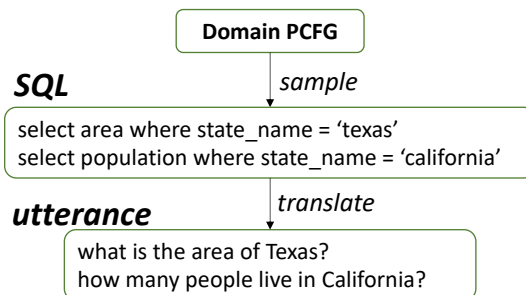

Figure 1: A two-stage generative process for synthesizing utterance-SQL pairs.

are formal languages that are intrinsically compositional. That is, the underlying grammar of programs is usually known and can be used to model the space of all possible programs effectively. Typically, grammars are used to constrain the program space during decoding of neural parsers (Yin and Neubig, 2018; Krishnamurthy et al., 2017). In this work, we utilize grammars to generate (unseen) programs, which are then used to synthesize more parallel data for semantic parsing.

Concretely, we use text-to-SQL as an example task, and propose a generative model to synthesize utterance-SQL pairs. As illustrated in Figure 1, we first employ a probabilistic context-free grammar (PCFG) to model the distribution of SQL queries. Then with the help of a SQL-to-text translation model, the corresponding utterances of SQL queries are generated subsequently. Our approach is in the same spirit as back-translation (Sennrich et al., 2016). The major difference is that the 'target language', in our case, is a formal language with known underlying grammar. Just like the training of a semantic parser, the training of the data synthesizer requires a set of utterance-SQL pairs. Hence, our generative model is unlikely to be useful if it is as data-hungry as a semantic parser. Our two-stage data synthesis approach, i.e. the PCFG and the translation model, is designed to be more sample-efficient compared to a neural semantic parser. To achieve better sample efficiency, we use the non-

---

2760

neural parameterization of PCFG (Manning and Schütze, 1999) and estimate it via simple counting. For the translation model, we use the pre-trained text generation model BART (Lewis et al., 2020). We sample synthetic data from the generative model to pre-train a semantic parser. The resulting parameters can presumably provide a strong compositional inductive bias in the form of initializations.

We conduct experiments on two text-to-SQL parsing datasets, namely GEOQUERY (Zelle and Mooney, 1996) and SPIDER (Yu et al., 2018b). In the query split of GEOQUERY, where training and test sets do not share SQL patterns, synthesized data helps boost the performance of a base parser by a large margin of 12.6%, leading to better *compositional generalization* of a parser. In the cross-domain [1] setting of SPIDER, synthesized data also boosts the performance by 3.1% in terms of execution accuracy, resulting in better *domain generalization* of a parser. Our work can be summarized as follows:

- We propose to efficiently learn a generative model that can synthesize parallel data for semantic parsing.

- We empirically show that the synthesized data can help a neural parser achieve better compositional and domain generalization. Our code and data are available at https://github.com/berlino/tensor2struct-public.

2 Related Work

Data Augmentation   Data augmentation for semantic parsing has gained increasing attention in recent years. Dong et al. (2017) use back-translation (Sennrich et al., 2016) to obtain paraphrase of questions. Jia and Liang (2016) induce a high-precision SCFG from training data to generate more new "recombinant" examples. Yu et al. (2018a, 2020) follow the same spirit and use a hand-crafted SCFG rule to generate new parallel data. However, the production rules of these approaches usually have low coverage of meaning representations. In this work, instead of using SCFG that accounts for rigid alignments between utterance and programs, we use a two-stage approach that implicitly models the alignments by taking advantage of powerful conditional text generators such

as BART. In this way, our approach can generate more diverse data. The most related work to ours is GAZP (Zhong et al., 2020) which synthesizes parallel data directly on test databases in the context of cross-database semantic parsing. Our work complements GAZP and shows that synthesizing data indirectly in training databases can also be beneficial for cross-database semantic parsing. Crucially, we learn the distribution of SQL programs instead of relying on handcrafted templates as in GAZP. The induced distribution helps a model explore unseen programs, leading to better compositional generalization of a parser.

Generative Models   In the history of semantic parsing, grammar-based generative models (Wong and Mooney, 2006, 2007; Zettlemoyer and Collins, 2005; Lu et al., 2008) have played an important role. However, learning and inference of such models are usually expensive as they typically require grammar induction (from text to logical forms). Moreover, their grammars are designed specifically for linguistically faithful languages, e.g., logical forms, thus not suitable for programming languages such as SQL. In contrast, our generative model is more flexible and efficient to train due to the two-stage decomposition.

3 Method

In this section, we explain how our method can be applied to text-to-SQL parsing.

3.1 Problem Definition

Formally, the labeled data for text-to-SQL parsing is given as a set of triples (x, d, y), and each triple represents an utterance x, the corresponding SQL query y and relational database d. A probabilistic semantic parser is trained to maximize $p(y|x, d)$. The goal of this work is to learn a generative model of $q(x, y|d)$ given databases such that it can synthesize more data (i.e., triplets) for training a semantic parser $p(y|x, d)$. Note that we use different notations q and p to represent the *generative model* and the *discriminative parser*, respectively, where $p(y|x, d)$ is not a posterior distribution of q. Instead, p is a separate model trained with different parameterization with q. This is primarily due to the intractability of posterior inference of $q(y|x, d)$. Specifically, we use a two-stage process to model the generation of utterance-SQL pairs as follows:

$$q(x, y|d) = q(y|d)q(x|y, d) \qquad (1)$$

```
sql = (select select, cond? where)
select = (agg* aggs)
agg = (agg_type agg_id, column col_id)
agg_type = NoneAggOp | Max | Min
cond = And(cond left, cond right)
     | Or(cond left, cond right)
     | Not(cond c)
```

Figure 2: A simplified ASDL grammar for SQL, where "sql, select, cond, agg" stands for variable types, "where, agg_id" for variable names, and "And, Or, Not" for constructor names.

where $q(y|d)$ models the distribution of SQLs given a database, and $q(x|y, d)$ models the translation process from SQL to utterances.

3.2 Database-Specific PCFG: $q(y|d)$

We use abstract syntax trees (ASTs) to model the underlying grammar of SQL, following Yin and Neubig (2018) and Wang et al. (2020b). Specifically, we use ASDL (Wang et al., 1997) formalism to define ASTs. To illustrate, Figure 2 shows a simplified ASDL grammar for SQL. The ASDL grammar of SQL can be represented by a set of context-free grammar (CFG) rules, as elaborated in the Appendix. By assuming the strong independence of each production rule, we model the probability of generating a SQL as the product of the probability of each production rule $q(y) = \prod_{i=}^{N} q(T_i)$. It is well known that estimating the probability of a production rule via maximum-likelihood training is equivalent to simple counting, which is defined as follows:

$$q(N \rightarrow \zeta) = \frac{C(N \rightarrow \zeta)}{\sum_{\gamma} C(N \rightarrow \gamma)} \qquad (2)$$

where C is the function that counts the number of occurrences of a production rule.

3.3 SQL-to-utterance Translation: $q(x|y, d)$

With generated SQL queries at hand, we then show how we map SQLs to utterances to obtain more paired data. We notice that SQL-to-utterance translation, which belongs to the general task of conditional text generation, shares the same output space with summarization and machine translation. Fortunately, pre-trained models (Devlin et al., 2019; Radford et al., 2019) using self-supervised methods have shown great success for conditional text generation tasks. Hence, we take advantage of a contemporary pre-trained model, namely BART (Lewis et al., 2020), which is an encoder-decoder model that uses the Transformer architecture(Vaswani et al., 2017).

To obtain a SQL-to-utterance translation model, we fine-tune the pre-trained BART model with our parallel data, with SQL being the input sequence and utterance being the output sequence. Empirically, we found that the desired translation model can be effectively obtained using the SQL-utterance pairs at hand, although the original BART model is designed for text-to-text translation only.

3.4 Semantic Parser: $p(y|x, d)$

After obtaining a trained generative model $q(x, y|d)$, we can sample synthetic pairs of (x, y) for each database d. The synthesized data will then be used as a complement to the original training data for a semantic parser. Following Yu et al. (2020), we use the strategy of first pre-training a parser with the synthesized data, and then fine-tuning it with the original training data. In this manner, the resulting parameters encode the compositional inductive bias introduced by our generative model. Another way to view pre-training is that a parser $p(y|x, d)$ is essentially trained to approximate the posterior distribution of $q(y|x, d)$ via massive samples from $q(x, y|d)$.

4 Experiments

We show that our generative model can be used to synthesize data in two settings of semantic parsing. We also present an ablation study for our approach.

In-Domain Setting We first evaluate our method in the conventional in-domain setting where training and test data are from the same database. Specifically, we synthesize new data for the GEO-QUERY dataset (Zelle and Mooney, 1996) which contains 880 utterance-SQL pairs on the database of U.S. geography. We evaluate in both question and query split, following Finegan-Dollak et al. (2018). The traditional question split ensures that no utterance is repeated between the train and test sets. This only tests limited generalization as many utterances correspond to the same SQL query; query split is introduced to ensure that neither utterances nor SQL queries repeat. The query split tests compositional generalization of a semantic parser as only fragments of test SQL queries occur in the training set.

Out-of-Domain Setting Then we evaluate our method in a challenging out-of-domain setting where the training and test databases do not overlap. That is, a parser is trained on some *source*

Model	Question Split	Query Split
seq2tree (Dong and Lapata, 2016)	62 [†]	31 [†]
GECA (Andreas, 2020)	68 [†]	49 [†]
template-based (2018)	55.2	-
seq2seq (Iyer et al., 2017)	72.5	-
Base Parser♣	70.9	49.5
Base Parser♣ + Syn Pre-Train	**74.6**	**62.1**
w.o. trained PCFG	72.4	54.8
w.o. pre-trained BART	71.5	53.9

Table 1: Execution accuracies on GEOQUERY. Methods with [†] measure exact match accuracy. *w.o.* stands for ablating a certain component.

Model	SET MATCH	EXECUTION
RAT-SQL♠ (Wang et al., 2020b)	69.7	-
RYANSQL♠ (Choi et al., 2020)	70.6	-
IRNet◇ (Guo et al., 2019)	61.9	-
GAZP (Zhong et al., 2020)	59.1	59.2
BRIDGE♠ (Lin et al., 2020)	70.0	68.0
Base Parser♣	70.4	69.4
Base Parser♣ + Syn Pre-Train	**71.8**	**72.5**
w.o. trained PCFG	71.4	72.3
w.o. pre-trained BART	70.6	70.8

Table 2: Set match and execution accuracies on SPIDER. ♠ stands for models with BERT-large, ◇ for BERT-base, ♣ for Electra-base.

databases but evaluated in unseen *target databases*. Concretely, we apply our method to the SPIDER (Yu et al., 2018b) dataset where the training contains utterance-SQL pairs from 146 source databases and the test set contains data from a disjoint set of target databases. In this out-of-domain setting, we synthesize data in the source databases in the hope that it can promote its domain generalization to unseen target databases.

Training As mentioned in Section 3.4, we use pre-training to augment a semantic parser with synthesized data. Specifically, we use the following four-step training procedure: 1) train a two-stage generative model, namely $q(x, y|d)$, 2) sample new data from it, 3) pre-train a semantic parser $p(y|x, d)$ using the synthesized data, 4) fine-tune the parser with the target training data. In the in-domain setting, one PCFG and translation model is trained. In the out-of-domain setting, a separate PCFG is trained on each source database assuming that each database has a different distribution of SQL queries. In contrast, a single translation model is trained and shared across source databases. We use RAT-SQL (Wang et al., 2020b) as our base parser.

The size of the synthesized data is always proportional to the size of the original data. We tune the ratio in $\{1, 3, 6, 12\}$, and find that 3, 6 works best for GEOQUERY and SPIDER respectively. We use the RAT-SQL implementation from Wang et al. (2020a) which supports value prediction and evaluation by execution. We train it with the default hyper-parameters. For the SQL-to-utterance translation model, we reuse all the default hyperparameters from BART (Lewis et al., 2020). Both models are trained using NVIDIA V100.

4.1 Main Results

For GEOQUERY, we report execution accuracy on the test sets of the question and query split; for SPI-

DER, we report exact set match (Yu et al., 2018b) along with execution accuracy on the dev set. The main results are shown in Table 1 and 2. First, we can see that compared with previous work, our base parser achieves the best performance, confirming that we are using a strong base parser to test our synthesized data.

With the pre-training using synthesized data, the performance of the base parsers is boosted in both GEOQUERY and SPIDER. In GEOQUERY, the pre-training results in the margin of 12.6% in the query split. This is somewhat expected as our generative model, especially $q(y|d)$ directly models the composition underlying SQL queries, which helps a parser generalize better to unseen queries. Moreover, our sampled SQL queries cover around 15% test SQL queries of the query split, partially explaining why it is so beneficial for the query split. In SPIDER, the pre-training boosts the performance by 3.1% in terms of execution accuracy. Although our model does not synthesize data directly for target databases (which are unseen), it still helps a parser achieve better domain generalization. This contradicts the observation by Zhong et al. (2020) that synthesizing data in source databases is useless, even harmful without careful consistency calibration. We attribute this to the pre-training strategy we use, as in our preliminary experiments we found that directly mixing the synthesized data with the original training data is indeed harmful.

4.2 Ablation Study

We try to answer two questions: a) *whether it is necessary to learn a PCFG*; b) *whether pre-trained translation model, namely BART, is required for success*. To answer the first question, we use a randomized version of $q(y|d)$ where the probability of production rules are uniformly distributed, instead of being estimated from data in Equation (2). As

Sampled SQLs (y)	Generated Utterances (x)
SELECT length FROM river WHERE traverse = "new york"	What is the length of the river whose traverse is in New York city?
SELECT Sum(length) FROM river WHERE traverse = "colorado"	What is the total length of the rivers that traverse the state of Colorado?
SELECT state_name FROM border_info WHERE border = "wyoming"	What are the names of the states that have a border with Wyoming?
SELECT state_name FROM city WHERE population = "mississippi"	What are the names of all cities in the state of Mississippi?
SELECT Min(state_name) FROM state WHERE state_name = "mississippi"	What is the minimum state name of the state with the name Mississippi?
SELECT capital FROM state WHERE population = 15000	What are the capitals of states with population of 150000 or more?

Table 3: Positive (top) and negative (bottom) examples of synthesized paired data for GEOQUERY.

shown in Table 1 and 2, this variant (*w.o.* trained PCFG) still improves the base parsers, but with a smaller margin. This shows that a trained PCFG model is better at synthesizing useful SQL queries. To answer the second question, we use a randomly initialized SQL-to-utterance translation model instead of BART. As shown in Table 1 and 2, this variant (*w.o.* pre-trained BART) results in a drop in performance as well, indicating that pre-trained BART is crucial for synthesizing useful utterances.

4.3 Qualitative Analysis

Table 3 shows examples of synthesized paired data for GEOQUERY. In the positive examples, the sampled SQLs can be viewed as recombinations of SQLs fragments observed in the training data. For example, `SELECT Sum(length)` and `traverse = "colorado"` are SQL fragments from separate training examples. Our PCFG combines them together to form a new SQL, and the SQL-to-utterance model successfully maps it to a reasonable translation. The negative examples consist of two kinds of errors. First, the PCFG generated semantically invalid SQLs which cannot be mapped to reasonable utterances. This error is due to the independence assumption made by the PCFG. For instance, when a column and its corresponding entity is separately sampled, there is no guarantee that they form a meaningful clause, as shown in `population = "mississippi"`. To address this, future work might consider more powerful generative models to model the dependencies within and across clauses in a SQL. Second, the SQL-to-utterances model failed to translate the sampled SQLs, as shown in the last example.

5 Conclusion

In this work, we propose to efficiently learn a generative model that can synthesize parallel data for semantic parsing. The synthesized data is used to pre-train a semantic parser and provide a strong inductive bias of compositionality. Empirical results on GEOQUERY and SPIDER show that the pre-training can help a parser achieve better compositional and domain generalization.

Acknowledgements

References

Jacob Andreas. 2020. Good-enough compositional data augmentation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7556–7566, Online. Association for Computational Linguistics.

DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol Shin. 2020. Ryansql: Recursively applying sketch-based slot fillings for complex text-to-sql in cross-domain databases. *arXiv preprint arXiv:2004.03125*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany. Association for Computational Linguistics.

Li Dong, Jonathan Mallinson, Siva Reddy, and Mirella Lapata. 2017. Learning to paraphrase for question answering. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 875–886, Copenhagen, Denmark. Association for Computational Linguistics.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-SQL evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360, Melbourne, Australia. Association for Computational Linguistics.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-SQL in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, Florence, Italy. Association for Computational Linguistics.

Jonathan Herzig and Jonathan Berant. 2019. Don't paraphrase, detect! rapid and effective data collection for semantic parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3810–3820, Hong Kong, China. Association for Computational Linguistics.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973, Vancouver, Canada. Association for Computational Linguistics.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Berlin, Germany. Association for Computational Linguistics.

Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1516–1526, Copenhagen, Denmark. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging textual and tabular data for cross-domain text-to-sql semantic parsing. *CoRR*, abs/2012.12627.

Wei Lu, Hwee Tou Ng, Wee Sun Lee, and Luke S. Zettlemoyer. 2008. A generative model for parsing natural language to meaning representations. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 783–792, Honolulu, Hawaii. Association for Computational Linguistics.

Christopher Manning and Hinrich Schütze. 1999. *Foundations of statistical natural language processing*. MIT press.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Bailin Wang, Mirella Lapata, and Ivan Titov. 2020a. Meta-learning for domain generalization in semantic parsing. *arXiv preprint arXiv:2010.11988*.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020b. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.

Daniel C Wang, Andrew W Appel, Jeffrey L Korn, and Christopher S Serra. 1997. The zephyr abstract syntax description language.

Yuk Wah Wong and Raymond Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 439–446, New York City, USA. Association for Computational Linguistics.

Yuk Wah Wong and Raymond Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 960–967, Prague, Czech Republic. Association for Computational Linguistics.

Pengcheng Yin and Graham Neubig. 2018. TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation. In *Proceedings of the 2018 Conference on Empirical Methods in*

Natural Language Processing: System Demonstrations, pages 7–12, Brussels, Belgium. Association for Computational Linguistics.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. 2020. Grappa: Grammar-augmented pre-training for table semantic parsing.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018a. SyntaxSQLNet: Syntax tree networks for complex and cross-domain text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1653–1663, Brussels, Belgium. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018b. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, page 658–666, Arlington, Virginia, USA. AUAI Press.

Victor Zhong, Mike Lewis, Sida I. Wang, and Luke Zettlemoyer. 2020. Grounded adaptation for zero-shot executable semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6869–6882, Online. Association for Computational Linguistics.

A CFG Rules

Following Yin and Neubig (2018), we represent ASDL grammar of SQLs using a set of production rules, as illustrated in Figure 3.

Formally, a production rule T is denoted as $N \to \zeta$, where N represents a non-terminal variable type, ζ represents a sequence of terminal or non-terminals. We can derive a set of production rules from our pre-defined ASDL grammar by instantiating original ASDL statements. For example, "sql = (select select, cond? where)" is instantiated into two rules: "sql → select" and "sql →

```
sql → select;   sql → select, cond;
select → agg;   select → agg, agg;
agg → agg_type, column;
agg_type → NoneAggOp;
agg_type → Min;   agg_type → Max;
cond → And;   cond → Or;   cond → Not;
```

Figure 3: Context-free grammars that represent the ASDL grammar in Figure 2 of the main paper. Only variable types are used in the production rules.

select, cond". With pre-defined production rules, a SQL can be transformed into a sequence of production rules. For example, the SQL query "select max(age)" can be represented by the sequence:

(1) sql → select
(2) select → agg
(3) agg → agg_type, column
(4) agg_type → Max
(5) column → age