Learning to Transpile AMR into SPARQL

Anonymous ACL submission

Abstract

We propose a transition-based system to transpile Abstract Meaning Representation (AMR) into SPARQL for Knowledge Base Question Answering (KBQA). This allows to delegate part of the abstraction problem to a strongly pre-trained semantic parser, while learning transpiling with small amount of paired data. We departure from recent work relating AMR and SPARQL constructs, but rather than applying a set of rules, we teach the BART model to selectively use these relations. Further, we avoid explicitly encoding AMR but rather encode the parser state in the attention mechanism of BART, following recent semantic parsing works. The resulting model is simple, provides supporting text for its decisions, and outperforms recent progress in AMR-based KBQA on LC-QuAD (F1 53.4), and QALD (F1 31.6), while exploiting the same inductive biases.

1 Introduction

Question Answering over Knowledge Bases (KBQA) (Zou et al., 2014; Vakulenko et al., 2019; Diefenbach et al., 2020) concerns the answering of natural questions by retrieving the information contained in a Knowledge Graph (KG). Compared to other automatic QA tasks, such as reading comprehension (Rajpurkar et al., 2018; Kwiatkowski et al., 2019; Clark et al., 2020), or free text generation (Lewis and Fan, 2019; Yin et al., 2016), KBQA has the advantage of producing answers backed against structured repositories of information, offering strong factual accuracy guarantees. Solving KBQA amounts to transforming a natural language question into some programming language, usually a query language such as SQL or SPARQL. It is thus considered a form of executable semantic parsing. We illustrate the KBQA task in Figure 1.

While some large KBQA datasets exist (Yu et al., 2018), the amount of paired examples, i.e. aligned natural language and query language pairs, is generally scarce (Trivedi et al., 2017; Usbeck et al., 2017). Furthermore, human language exhibits large variability, and obtaining enough training pairs for specific domains or infrequent natural language formulations requires large annotations investments. Often, real world implementations of KBQA end up containing a number of domain specific handmade rules that are costly to maintain and expand. The need to manipulate a formal representation in the target side with only a few examples, makes this task harder to learn compared to other QA tasks.

```
PREFIX dbr:http://dbpedia.org/resource/
PREFIX dbo:http://dbpedia.org/ontology/
PREFIX dbp:http://dbpedia.org/property/
SELECT DISTINCT ?uri WHERE {
  ?x <dbp:state> <dbr:Maharashtra>.
  ?x <dbo:sport> ?uri.
}
```

Figure 1: SPARQL graph for the question *Name some sports played in institutions of Maharashtra?*. `?uri` is the unknown variable (related to *sports*) and `?x` is an intermediate variable (related to *institution*), needed to relate the unknown with the KG entity (*Maharashtra*).

In order to mitigate the data availability problem (Kapanipathi et al., 2021) proposed to delegate part of the task to an Abstract Meaning Representation (AMR) parse that incorporates additional semantic information. This work identifies associations between certain AMR nodes and SPARQL variables as well as a associations between AMR sub-graphs and relations in a KG. Unfortunately, these AMR to SPARQL mappings are imperfect, suffering from coverage and granularity mismatch. Another concern with AMR-based approaches is the encoding of the AMR itself which can be challenging and implies an additional learning burden.

Taking these limitations into account, in this work we propose a new approach to leverage AMR parsing for KBQA that *learns to transpile* AMR

	Propbank/AMR	SPARQL mapping
a	`:e (n/amr-unknown)`	if e $\neq$:polarity then map n to unknown variable
b	`have-degree-91 :ARG1 amr-unknown :ARG5 n`	map n to unknown variable (override a)
c	`n :mode imperative :ARG1 n2`	if n is root then map $n2$ to unknown variable
d	`n :wiki c`	subgraph below n mapped to KG entity c
e	`n2 <ancestor of> n :wiki c`	map first n2 aligned to noun to (optional) secondary variable
f	`have-degree-91 :ARG2 n`	n is (optional) secondary variable (quantitative)
g	`have-quant-91 :ARG1-of n`	n is (optional) secondary variable (quantitative)
h	`:time`	n is (optional) secondary variable

Table 1: Relations between SPARQL and AMR constructs. This is a subset from (Kapanipathi et al., 2021). Note that `:wiki` can be attached above AMR entity subgraphs, unlike conventional AMR. See realignment in Section 4

into the SPARQL query language. The contributions can be summarized as follows

- In the spirit of transition-based semantic parsing, we develop state machine and an oracle that transpiles AMR into SPARQL and learn to imitate it with a BART (Lewis et al., 2019) based model for KBQA.
- This oracle leverages known relations (i.e. similarities) between AMR and SPARQL (Kapanipathi et al., 2021), but rather than applying them deterministically as in prior work, we teach the model when to use them.
- We show that it is not necessary to encode AMR directly, but rather encoding the transpiler state through attention masking as in (Fernandez Astudillo et al., 2020) suffices.
- The resulting transpiler outperforms (Kapanipathi et al., 2021) by 9 F1 points on LcQUAD and matches it on QALD, while being simpler and exploiting the same inductive biases.

2 AMR to SPARQL Machine and Oracle

Here we propose to apply the transition-based approach, well known in syntactic and semantic parsing, to learn to transpile AMR to SPARQL.

2.1 A Transition-based Transpiler

The objective is to learn to predict the SPARQL query s corresponding to the natural language question w, by transforming its AMR parse g.

At its core, a transition-based approach learns to predict a sequence of actions a from w, that applied to a non-parametric state machine yield s.

$$s = M(a, g) \tag{1}$$

What the actions are, needs to be determined for each problem. Generally, for transpiling AMR we can define a non-parametric oracle that given the original sentence, its AMR and the SPARQL, yields the actions

$$a = O(s, g, w). \tag{2}$$

The oracle is only needed to generate the tuples (w, g, a) for training. With these, one can use the oracle as a teacher to train the sequence to sequence model $p(a \mid w)$ and predict the SPARQL as

$$\hat{s} = M(\hat{a}, g) \tag{3}$$

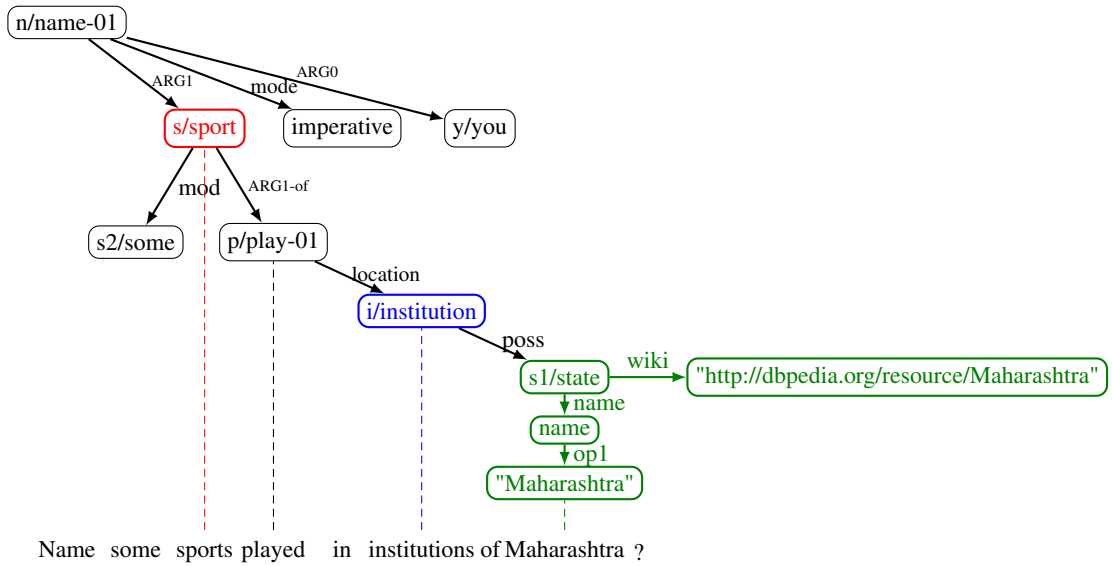
where g is obtained from an AMR parse of w and $\hat{a}$ is obtained with conventional decoding.

$$\hat{a} = \arg\max_{a} \{p(a \mid w)\} \tag{4}$$

It is often useful to use the state of the state machine i.e. the parser state to constrain the output vocabulary at every decoding step, and even at train time. In this way one can forbid the machine from considering invalid actions and make mistakes, or put additional strain in training. It has also been shown that Transformer models benefit from dedicating one or more attention heads to reflect the parser state (Fernandez Astudillo et al., 2020) and this applies also to pre-trained sequence to sequence models (Zhou et al., 2021b). In Section 3 we detail how we apply this strategy, and show that we can avoid encoding the AMR through this mechanism.

It is important to note that the use of a state machine generates a strong inductive bias, imposing some specific way in which the sequence to sequence problem can be solved. In this case we leverage the bias to make the transpiler aware of AMR path information, but it comes at the penalty of not fully being able to recover all SPARQL queries i.e. for some queries

$$s \neq M(O(s, g, w), g) \tag{5}$$



Path Stack:	Supporting Text:	Actions:	SPARQL:
		SELECT	SELECT DISTINCT ?s WHERE {
s1 i p s | s1 i | i p s	sports played institutions Maharashtra	REDUCE	
s1 i p s | s1 i | i p s	institutions Maharashtra	state	?i <dbp:state> <dbr:Maharashtra>.
s1 i p s | s1 i | i p s	sports played institutions	sport	?i <dbo:sport> ?s.
s1 i p s | s1 i | i p s		CLOSE	}

Figure 2: Oracle for the LCQUAD train set sentence: *Name some sports played in institutions of Maharashtra?*. Top: Sentence w aligned to its AMR graph g (input) and 3 relevant subgraphs identified by applying Table 1 (c, d, e): **unknown variable** (imperative root), **(optional) secondary variable** (entity-adjacent nominal), **linked entity** (wiki). Bottom box: All decoding timesteps including implicit machine state defined by the AMR path stack, explicit machine state defined by the supporting text, oracle action sequence a and resulting SPARLQ s (output).

Throughout this work, it is also assumed that the AMR parser provides alignments between sentence w and AMR graph g. (Zhou et al., 2021a).

2.2 Path-based Oracle and Machine

The work in (Kapanipathi et al., 2021), showed that certain AMR subgraphs can be deterministically mapped to SPARQL elements, such as KG entities or unknown variables. The path algorithm (Kapanipathi et al., 2021), identifies relevant paths between those subgraphs, and applies rules deterministically to obtain SPARQL variables and relations. These rules lead however to prediction errors, e.g. spurious secondary variables, and suffer from granularity mismatch between AMR and KG graph.

Here we propose an alternative approach. We consider all AMR paths obtained by applying only the subgraph mappings listed in Table 1 and highlighted in Figure 2. The goal is to learn to predict a KG relation for every path or to ignore it.

We implement a state machine $M(a, g)$ characterized by the stack of paths of the AMR graph g. To initialize the machine, we identify subgraphs in g representing the unknown variable to return in the query (red), entities in the KG (green) or optional secondary variables, that may not exist in the SPARQL (blue). We then fill a FIFO stack with the paths between these subgraphs, starting by each path between the unknown and entity, followed by the paths involved their related optional variables in Table 1 (e) and finally other optional variables.

The oracle $a = O(s, g, w)$ then produces one of the following actions[1] a_t to be played on the state machine

- {SELECT, ASK, COUNT}: Generate the query header from a closed vocabulary

[1]The actions names do not reflect the literal SPARQL code but are rather keywords indexing a more verbose string.

- <KG relation>: Produce the KG relation for path at the top of the stack and REDUCE
- REDUCE: Pop path at the top of the stack without predicting any KG relation
- {CLOSE, ASC, DESC}: Close or perform query post processing i.e. top/bottom-k search

The oracle starts by determining the header of the query and then proceeds over the stack of paths, predicting a KG relation for every path, or ignoring it (REDUCE). A KG relation is predicted when the path can be aligned to a SPARQL triple. Once the stack is empty, the oracle closes the machine or applies post-processing and closes. Figure 2 shows an example of a complete oracle action sequence.

One fundamental advantage of the proposed approach, is that it aligns by construction sentence w, AMR g and SPARQL s. This information can be used during decoding to restrict the output vocabulary of $p(a \mid w)$. For example, we can enforce header and closing operations only on a full and empty stack respectively. Further, we can also query the KG with nodes involved in the path on the top of the stack, to restrict the possible relations to predict. For example in Figure 2, bottom, to predict the KG relation `state` we restrict it to incoming or outgoing KG relations of the node `Maharashtra`. Finally, we also obtain textual cues to predict the KG relation, see **Supporting Text** in Figure 2. In Section 3 we describe how textual cues are incorporated into the model, allowing to avoid encoding AMR explicitly.

For completeness, Figure. 3 shows the oracle for a quantitative question requiring filtering by a secondary variable. In this case, the Propbank frame have-degree-91 indicates the possibility of a secondary variable *tall*. Due to SPARQL's lack of schema, sometimes KG relations exist that remove the need for secondary variables e.g. `largestCity` allows already to answer *Which is the largest city in x?*. The transpiler must thus learn when to REDUCE these additional variables. It is unlikely that the model can determine the KG structure from the limited data, but the constrained decoding plus some general regularities give the model an opportunity to learn when to reduce.

3 Transition Model with Inductive Bias

We parametrize $p(a \mid w)$ using an off-the-shelf sequence to sequence model such as the performant Transformer model (Vaswani et al., 2017) or its pretrained versions (Lewis et al., 2019; Raffel et al., 2019). The model leans to imitate the oracle: it predicts the header, then proceeds over the path stack of the state machine to predict KG relations and the closing actions.

3.1 Constrained Decoding

Since the target side is a formal description and it evolves according to a state-machine, it is possible to add inductive bias by masking the output vocabulary to prevent the model from selecting forbidden actions:

$$p(a_t \mid a_{<t}, w) \propto \exp(f(a_{<t}, w) + m(a_{<t}, w)) \tag{6}$$

where $f(a_{<t}, w)$ is e.g. a BART with the last softmax layer removed and $m(a_{<t}, w)$ is a mask based on the state of the machine i.e. a deterministic function of the input sentence w and action history $a_{<t}$, that can be set to $-\infty$ to forbid actions.

For the proposed model, masking is used to perform header actions only at the beginning of the action sequence and closing actions only at the end. In addition, when the AMR path at the top of the stack contains entity nodes, the KG is queried and the mask is set to restrict the actions to the appropriate KG relations. The relation's prefix [2] and direction is also obtained in this process.

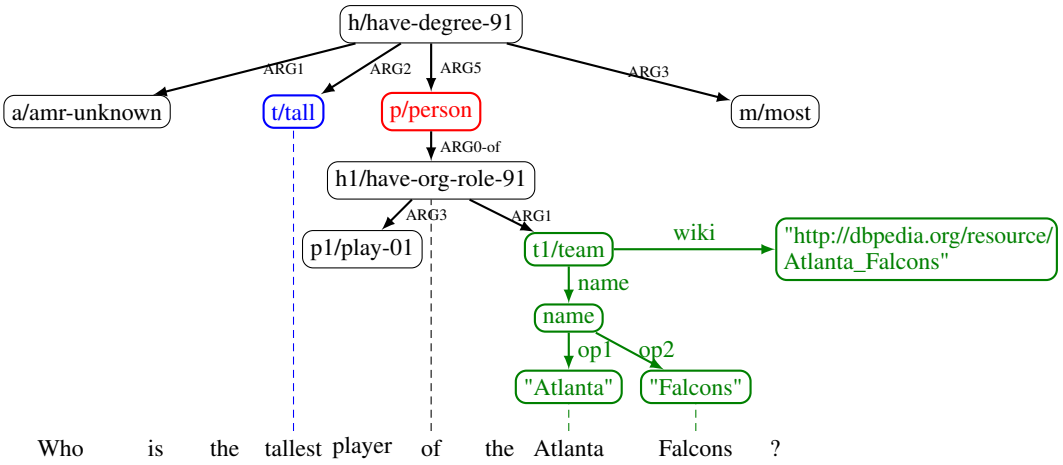
These constrains are applied at decoding time only. As shown in the experimental setup, constrained decoding has a fundamental effect on performance.

3.2 Encoding the Transpiler State

The mechanism described above allows the sequence to sequence model to become partially aware of KG topology, but it does not allow for a proper encoding of the machine state. Since the state involves proceeding over a stack of AMR paths, a lot of relevant information is lost. One possible option would be to feed the AMR as additional input to BART, but this would force the model to learn to interpret its structure, increase the input size by a factor above 2 (likely $3-5$) and thus making the model considerably slower.

As shown in (Fernandez Astudillo et al., 2020; Zhou et al., 2021a,b), notable gains can be attained by encoding the parser state in one or more attention heads of the Transformer. In their work, the

[2] PREFIX dbo: http://dbpedia.org/ontology/, PREFIX dbp: http://dbpedia.org/property/



Path Stack:	Supporting Text:	Actions:	SPARQL:
		SELECT	SELECT DISTINCT ?p WHERE {
t1 h1 p | t h p	of Atlanta Falcons	team	?p <dbo:team> <dbr:Atlanta_Falcons> .
t1 h1 p | t h p	tallest of	height	?p <dbo:height> ?t
t1 h1 p | t h p		DESC	} ORDER BY DESC(?t) OFFSET 0 LIMIT 1

Figure 3: Question from the QALD train set: *Who is the tallest player of the Atlanta Falcons?*. Top: Sentence w aligned to its AMR graph g (input) and 3 relevant subgraphs identified by by applying Table 1 (a, b, d, f): **unknown variable** (amr-unknown+have-degree-91), quantitative **(optional) secondary variable** (have-degree-91), **linked entity** (wiki). Bottom box: All decoding timesteps including implicit machine state defined by the AMR path stack, explicit machine state defined by the supporting text, oracle action sequence a and resulting SPARLQ s (output).

stack contains input words which are used to mask the cross-attention mechanism of one or two heads of the Transformer to attend only the stack contents. In our model, however, the stack contains AMR paths, rather than words. To resolve this, we propose to attend the words aligned to the nodes in the path at the top of the stack. Therefore, we are attending to the part of the sentence w aligned to the path of g for which we are going to predict a relation. As depicted in Figures 2 and 3, our approach provides an additional supporting text for every transpiler decision. The use of the supporting text helps the explainability of the model decisions, but also, as show in the experimental setup, plays an important role in performance.

4 Experimental Setup

4.1 Datasets

To evaluate our system, we used two standard KBQA datasets on DBpedia and followed the partitions in (Kapanipathi et al., 2021).

LC-QuAD 1.0 (Trivedi et al., 2017) is a dataset with 4,000 questions for training and 1,000 questions for test, created from templates. More than 80% of its questions contain two or more relations. Our modules are evaluated against a random sample of 200 questions from the training set. We also held out a second dev set of 200 questions to prevent overfit. LC-QuAD 1.0 predominantly focuses on the multi-relational questions, aggregation (e.g. COUNT) and simple questions.

QALD-9 (Usbeck et al., 2017) is a dataset with 408 training and 150 test questions in natural language, from DBpedia. Each question has an associated SPARQL query and gold answer set. We created a randomly chosen development set of 98 questions for evaluating individual modules and a secondary dev set of 63 questions to prevent overfit. QALD-9 contains much complex queries than LC-QuAD 1.0 including more realistic quantitative questions requiring secondary unknowns, time constructs and other.

4.2 AMR Parsing and State Machine

We used the Action Pointer Transformer (APT) transition-based parser (Zhou et al., 2021a) which provides alignments between surface tokens. As in most AMR parsers, entitity linking is delegated to BLINK (Wu et al., 2020) and applied as a post-

	LC-QuAD 1.0		QALD-9	
	Oracle (F1)	System (F1)	Oracle (F1)	System (F1)
All AMR Constructs	68.7	47.4	**67.0**	**53.6**
no time construct, Table 1 (h)	69.8	54.7	67.0	50.5
no quant. constructs, Table 1 (f, g)	68.7	54.6	68.2	47.5
no time and no quant. constructs	**69.8**	**56.5**	68.2	45.4
Gold EL	78.7	63.1	77.5	58.6

Table 2: Ablation study measuring the effect of oracle constructs and Entity Linking on oracle and trained model peformances. Measured by F1 KBQA SPARQL on the dev set.

		LC-QuAD 1.0			QALD-9		
		P	R	F1	P	R	F1
TransQA	mask 8 heads	**61.0**	**55.1**	**56.5**	**58.2**	**53.1**	**53.6**
TransQA	no mask	56.6	51.0	52.1	53.1	49.0	49.5
TransQA	mask 5 heads	60.0	55.1	56.0	54.1	50.0	50.5
TransQA	mask 12 heads	59.7	54.5	55.8	58.2	52.2	52.8
TransQA	no constrained decoding using KG	49.9	45.1	46.3	42.9	41.5	41.8

Table 3: Ablation study measuring the effect of constrained decoding and parser state encoding on BART. Measured by F1 KBQA SPARQL on the dev set.

processing stage for each AMR named entity subgraph. However, arround 30% of DBPedia entities are long spans of text, containing dates and locations and parsers often fail to produce the expected named entity subgraphs. To provide accurate linked entity subgraph detection, we run BLINK separately and then attach the `:wiki` edge to the most suitable node in the AMR, even if this is not a named entity subgraph head. Linking the AMR nodes to KG entities is attained by matching the span of text aligned to a subgraph with the mention via a greedy set of checks. Attachment to conventional named entities is attempted first. For the remaining 30% of the cases edit distance and fuzzy match search are used to find a suitable alignment.

The path stack defining the state of the state machine is determined from the aligned AMR graph, see Section 2.2. The same mechanism is used to produce the oracle action sequences for training[3].

4.3 Sequence to Sequence Model

We use BART (Lewis et al., 2020) implemented in (Wolf et al., 2020) to parametrize $p(a \mid w)$ and learn to imitate the oracle. We implement both constrained decoding at test time and parser state encoding by masking $8/16$ cross-attention heads both at train and test time, see Section 3.

For LC-QuAD 1.0 we used the train set comprising of 3,600 questions. The LC-QuAD 1.0 model was trained for 13 epochs with a learning rate of $5e^{-5}$. For development we built a QALD-9 model with the 3,600 LC-QuAD 1.0 and 247 QALD-9 examples from the designated train set. We trained for 14 epochs and used a learning rate of $4e^{-5}$. The final QALD-9 model is trained with all available data (4,000 LcQuAD 1.0 and 408 QALD-9 examples) including the dev set. We upsample the QALD-9 5 times to balance between the two datasets and we trained the model for 16 epochs. The upsampling ratio and the number of epochs are determined on the QALD-9 dev. For both models we set the max input sequence length to 64 tokens, max target sequence length to 32 tokens and beam size to 4. We used the Adam optimizer with standard parameters and trained on a V100 Nvidia GPU.

5 Results

In all experiments we refer to our system composed of the Oracle, the KBQA BART model, and the State Machine as TransQA. We analyze the impact of different components and we compare with prior work. We report the Macro Precision (P),

[3]We use spacy `https://spacy.io/` part-of-speech for Table 1, e

	LC-QuAD 1.0			QALD-9		
	P	R	F1	P	R	F1
NSQA (Kapanipathi et al., 2021)	44.8	45.8	44.5	31.4	32.2	30.9
EDGQA (Hu et al., 2021)	50.5	56.0	53.1	31.3	40.3	**32.0**
STaG-QA (Ravishankar et al., 2021)	76.5	52.8	51.4	-	-	-
TransQA (our system)	57.2	52.9	**53.3**	36.1	30.8	31.6

Table 4: KBQA SPARQL Test Result. Training data from Lc-QuAD 1.0 (4K examples) and QALD (408 example).

Macro Recall (R), and Macro F1 scores by comparing the gold answers to the answers that TransQA generates when executing its predicted SPARQL. QALD-9 provides the gold answer directly. LC-QuAD 1.0 provides the gold SPARQL which we execute to obtain the gold result.

5.1 System and Oracle Performance

Table 2 shows the ablation of AMR components and its effect on the SPARQL performance for both the oracle and our system. As expected, the whole approach is dependent on the Entity Linking performance of BLINK. The used model had a F1 score of 86.0 in both LC-QuAD 1.0 dev and test, 84.0 on QALD-9 dev and 72.3 on QALD-9 test, as measured against entities present in the queries. If we assume the EL performance is perfect, we get a large improvement both in oracle and trained model, gaining 9 points on both corpora. We also examine the performance gap between the oracle action sequence and the system sequence predicted with BART. There is a difference of 13 points between the oracle and the BART system on both datasets. This difference between the oracle and BART is more that 15 points when using the gold entities.

We ablate the time and quantitative constructs of the oracle detailed in Section 2.2 and Table 1. Results show that these constructs have a much larger effect on the trained model than on the oracle, indicating their importance for learning regularities. The LC-QuAD 1.0 dataset is template based and the SPARQL queries require simple conjunction of the matching KG triples. For superlative relations in LC-QuAD 1.0 the appropriate relation exists in the KG and there is no need for additional actions. For this reason, it does not benefit form the use of the corresponding AMR constructs. QALD-9, on the contrary, benefits equally both from time and quantitative subgraph handling, with a positive effect both in oracle and on the trained system.

It is worth noting that the trained system showed, on the dev set[4], the ability to handle complex regularities, despite the limited training data. For example it showed the ability to differentiate superlative questions that can be solved directly with a relation, such as `largestCity`, from situations that require an additional secondary variable.

5.2 Parser State Encoding

In Table 3 we show the importance of incorporating the state machine into the BART model on the LC-QuAD 1.0 and QALD-9 dev sets. Regarding encoding of the parser state through cross-attention masking, see 3.2, its use provides clear gains of 4 points on both corpora. Variations of the optimal number of heads (8/16) yield also sub-optimal results although by a smaller margin.

We also explore the effect of the constrained decoding using the KG grammar to restrict the action of every decoding step, described in Section 3.1. The use of constrained decoding has a very large effect of 10 points on LC-QuAD 1.0 and 12 points on QALD-9. This is likely because of the large reduction on possible decoding options, but also because it provides some knowledge about the KG structure.

5.3 Comparison with other Approaches

Table 4 compares the proposed approach with recent related works. Against the most directly related, NSQA (Kapanipathi et al., 2021) we obtain an increase of 9 F1 points on the LC-QuAD 1.0 dataset, and 0.7 F1 points for QALD-9. It should be noted that our approach exploits the same inductive biases, but eliminates deterministic transformations and the need for additional modules such as relation linkers, making it much simpler.

EDGQA is a recent approach making use of multiple rules to attain KBQA. Against it, we obtain

[4]We avoided result analysis on the test set to prevent corpus overfitting.

a modest increase on LC-QuAD and are 0.4 below on QALD. This slight gap is likely caused by differences in the entity linking approach, as discussed in the previous section. EDGQA uses an ensemble three entity linking tools: Dexter (Ceccarelli et al., 2014), EARL (Dubey et al., 2018) and Falcon (Sakor et al., 2019) as one entity retriever.

Finally we also compare our approach with the contemporaneous STaG-QA (Ravishankar et al., 2021), that uses a sketch approach and learns end-to-end. This work does not provide QALD-9 results, but provides LC-QuAD 1.0 which we outperform by almost 2 points. The authors also present LC-QuAD 1.0 results with a model pretrained with additional out-of-domain gold data (30,000 sentences) from LC-QuAD 2.0 and they obtain an F1 score of 53.6. This results is provided for completeness but is not directly comparable with our approach.

6 Related work

The work (Kapanipathi et al., 2021) introduces the Neuro-Symbolic Question Answering (NSQA) and is the most directly related to ours. NSQA also leverages AMR parses and a super-set of the mappings in Table 1. It however applies this mappings deterministically and proposes additional rules to deal with structural and granularity mismatch between the AMR graph and the KG. It also trains separate modules for relation linking and handling of logic and integrates all approaches into one single SPARQL hypothesis. By contrast this work makes uses of a small set of mappings between AMR and SPARQL and proposes a transition-based approach that learns to use them, resulting on a simpler system that is more performant on average while exploiting the same inductive biases.

EDGQA (Hu et al., 2021) is a rule based system for KBQA over DBpedia. They propose a custom Entity Description Graph (EDG) to represent the structure of complex questions, rather than relying on established formalists such as AMR. This formalism is also constructed using a rule-based question decomposition technique, which is likely to generalize worse than state-of-the-art AMR parsers. EDGQA also uses an ensemble of several entity linking models that is likely to provide an additional advantage, easily portable to our approach.

Another system making abundant use of rules is (Yih et al., 2015), which proposes a semantic parsing approach for KBQA specialized for Freebase. They describe a rule based system called STAGG (Staged Query Graph Generation) that iteratively expands the graph from entities by following KG relations using a similarity function.

More recently (Saparina and Osokin, 2021) proposes a system for KBQA and intermediate representation based on Question Decomposition Meaning Representation (QDMR) (Wolfson et al., 2020), a lightweight semantic parsing scheme, based on breaking down sentences. This is completed with a non-trainable transpiler component that transforms the intermediate representation into SPARQL queries.

All of the described systems can be seen as non learnable transpilers of custom or relatively unfrequent semantic formalisms. Compared to these we provide an approach that automatically learns transpilation, on top of the well established AMR.

The contemporaneous work (Ravishankar et al., 2021) proposes a two stage text to SPARQL system that first generates a query skeleton learned end-to-end with a sequence to sequence model, and then performs beam search to find optimal grounding of the skeleton into a target KG. In contrast, our approach introduces structure into the sequence to sequence modeling, allowing to easily relate SPARQL, AMR and input text while having a better performance on equal conditions.

Regarding the modeling of the parser state on Transformers, (Fernandez Astudillo et al., 2020) showed the benefit of masking cross attention heads to reflect the content of buffer and stack, while (Zhou et al., 2021b) demostrated that this still works for pre-trained sequence to sequence models. Here we take this technique one step further and show that it is possible avoid encoding AMR by using an extension of this approach and that it has a stronger impact in performance compared to Structured-BART (Zhou et al., 2021b).

7 Conclusions

We have introduced an approach for KBQA that transpiles a well established semantic representation, AMR, obtained from an off-the-shelf parser, into SPARQL. This transpiling operation is learned through a transition-based approach, rather than being rule-based as in previous approaches. AMR encoding is also avoided by encoding the parser state resulting in a simpler model that outperforms recent state-of-the art approaches on LC-QuAD.

References

Diego Ceccarelli, Claudio Lucchese, Salvatore Orlando, R. Perego, and Salvatore Trani. 2014. Dexter 2.0 - an open source tool for semantically enriching data. In *SEMWEB*.

Jonathan H. Clark, Eunsol Choi, Michael Collins, Dan Garrette, Tom Kwiatkowski, Vitaly Nikolaev, and Jennimaria Palomaki. 2020. TyDi QA: A benchmark for information-seeking question answering in typologically diverse languages. *Transactions of the Association for Computational Linguistics*, 8:454–470.

Dennis Diefenbach, Andreas Both, Kamal Deep Singh, and Pierre Maret. 2020. Towards a question answering system over the semantic web. *Semantic Web*, 11:421–439.

Mohnish Dubey, Debayan Banerjee, Debanjan Chaudhuri, and Jens Lehmann. 2018. Earl: Joint entity and relation linking for question answering over knowledge graphs.

Ramón Fernandez Astudillo, Miguel Ballesteros, Tahira Naseem, Austin Blodgett, and Radu Florian. 2020. Transition-based parsing with stack-transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1001–1007, Online. Association for Computational Linguistics.

Xixin Hu, Yiheng Shu, Xiang Huang, and Yuzhong Qu. 2021. Edg-based question decomposition for complex question answering over knowledge bases. In *The Semantic Web – ISWC 2021*, pages 128–145, Cham. Springer International Publishing.

Pavan Kapanipathi, Ibrahim Abdelaziz, Srinivas Ravishankar, Salim Roukos, Alexander Gray, Ramón Fernandez Astudillo, Maria Chang, Cristina Cornelio, Saswati Dana, Achille Fokoue, Dinesh Garg, Alfio Gliozzo, Sairam Gurajada, Hima Karanam, Naweed Khan, Dinesh Khandelwal, Young-Suk Lee, Yunyao Li, Francois Luus, Ndivhuwo Makondo, Nandana Mihindukulasooriya, Tahira Naseem, Sumit Neelam, Lucian Popa, Revanth Gangi Reddy, Ryan Riegel, Gaetano Rossiello, Udit Sharma, G P Shrivatsa Bhargav, and Mo Yu. 2021. Leveraging Abstract Meaning Representation for knowledge base question answering. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3884–3894, Online. Association for Computational Linguistics.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association of Computational Linguistics*.

Mike Lewis and Angela Fan. 2019. Generative question answering: Learning to answer the whole question. In *International Conference on Learning Representations*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

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*. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. In *ACL*.

Srinivas Ravishankar, June Thai, Ibrahim Abdelaziz, Nandana Mihindukulasooriya, Tahira Naseem, Pavan Kapanipathi, Gaetano Rossiello, and Achille Fokoue. 2021. A two-stage approach towards generalization in knowledge base question answering.

Ahmad Sakor, Isaiah Onando Mulang', Kuldeep Singh, Saeedeh Shekarpour, Maria Esther Vidal, Jens Lehmann, and Sören Auer. 2019. Old is gold: Linguistic driven approach for entity and relation linking of short text. 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 2336–2346, Minneapolis, Minnesota. Association for Computational Linguistics.

Irina Saparina and Anton Osokin. 2021. Sparqling database queries from intermediate question decompositions. In *EMNLP*.

Priyansh Trivedi, Gaurav Maheshwari, Mohnish Dubey, and Jens Lehmann. 2017. Lc-quad: A corpus for complex question answering over knowledge graphs.

Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Bastian Haarmann, Anastasia Krithara, Michael Röder, and Giulio Napolitano. 2017. 7th open challenge on question answering over linked data (qald-7). In *Semantic Web Challenges*, pages 59–69, Cham. Springer International Publishing.

Svitlana Vakulenko, Javier David Fernandez Garcia, Axel Polleres, M. de Rijke, and Michael Cochez. 2019. Message passing for complex question answering over knowledge graphs. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Tomer Wolfson, Mor Geva, Ankit Gupta, Matt Gardner, Yoav Goldberg, Daniel Deutch, and Jonathan Berant. 2020. Break it down: A question understanding benchmark. *Transactions of the Association for Computational Linguistics*, 8:183–198.

Ledell Yu Wu, Fabio Petroni, Martin Josifoski, Sebastian Riedel, and Luke Zettlemoyer. 2020. Scalable zero-shot entity linking with dense entity retrieval. In *EMNLP*.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1321–1331, Beijing, China. Association for Computational Linguistics.

Jun Yin, Xin Jiang, Zhengdong Lu, Lifeng Shang, Hang Li, and Xiaoming Li. 2016. Neural generative question answering. *ArXiv*, abs/1512.01337.

Tao Yu, Rui Zhang, Kai-Chou Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Z Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *EMNLP*.

Jiawei Zhou, Tahira Naseem, Ramón Fernandez Astudillo, and Radu Florian. 2021a. AMR parsing with action-pointer transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5585–5598, Online. Association for Computational Linguistics.

Jiawei Zhou, Tahira Naseem, Ramón Fernandez Astudillo, Young-Suk Lee, Radu Florian, and Salim Roukos. 2021b. Structure-aware fine-tuning of sequence-to-sequence transformers for transition-based AMR parsing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6279–6290, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Lei Zou, Ruizhe Huang, Haixun Wang, Jeffrey Yu, Wenqiang He, and Dongyan Zhao. 2014. Natural language question answering over rdf - a graph data driven approach. *Proceedings of the ACM SIGMOD International Conference on Management of Data*.