SYGMA: A System for Generalizable and Modular Question Answering Over Knowledge Bases

Sumit Neelam[†], **Udit Sharma**[‡], **Hima Karanam, Shajith Ikbal, Pavan Kapanipathi, Ibrahim Abdelaziz, Nandana Mihindukulasooriya, Young-Suk Lee, Santosh Srivastava**[*], **Cezar Pendus, Saswati Dana, Dinesh Garg, Achille Fokoue, G P Shrivatsa Bhargav, Dinesh Khandelwal, Srinivas Ravishankar**[*], **Sairam Gurajada**[*], **Maria Chang, Rosario Uceda-Sosa, Salim Roukos, Alexander Gray, Guilherme Lima, Ryan Riegel, Francois Luus**[*], **L Venkata Subramaniam**

IBM Research

Abstract

Knowledge Base Question Answering (KBQA) involving complex reasoning is emerging as an important research direction. However, most KBQA systems struggle with generalizability, particularly on two dimensions: (a) across multiple knowledge bases, where existing KBQA approaches are typically tuned to a single knowledge base, and (b) across multiple reasoning types, where majority of datasets and systems have primarily focused on multi-hop reasoning. In this paper, we present SYGMA, a modular KBQA approach developed with goal of generalization across multiple knowledge bases and multiple reasoning types. To facilitate this, SYGMA is designed as two high level modules: 1) KB-agnostic question understanding module that remain common across KBs, and generates logic representation of the question with high level reasoning constructs that are extensible, and 2) KB-specific question mapping and answering module to address the KB-specific aspects of the answer extraction. We evaluated SYGMA on multiple datasets belonging to distinct knowledge bases (DBpedia and Wikidata) and distinct reasoning types (multi-hop and temporal). State-of-the-art or competitive performances achieved on those datasets demonstrate its generalization capability.

1 Introduction

The goal of Knowledge Base Question Answering (KBQA) systems is to answer natural language (NL) questions by retrieving and reasoning over facts in Knowledge Base (KB). KBQA has gained significant popularity in recent times due to its practical real-world applications and associated research challenges (Fu et al., 2020). However, research in this area has primarily focused so far on *single/multi-hop* reasoning on a *single* knowledge base (Trivedi et al., 2017; Usbeck et al., 2017; Yih et al., 2016). As a result, most of the methods developed are tuned to a restricted set of reasoning types on a single knowledge base.

In this paper, we present a modular approach for KBQA called **SYGMA** (**Sy**stem for **G**eneralizable and **M**odular question **A**nswering over knowledge bases), that is built on a framework adaptable to multiple KBs and multiple reasoning types. Such a system hold promise for many interesting applications namely: answering complex questions involving multiple types of reasoning, unifying of knowledge across multiple KBs, and so on. However, it poses challenges in terms of: (a) Handling different representations of information in different KBs. For example, Figure 1 shows how information related to question *"Who was roman emperor before Nero?"* is represented in two different KBs. Wikidata represents properties of facts such as *replaces*, *temporal* and *spatial* with reification[1]. On the other hand, DBpedia manages to represent it as a simple fact with a relationship to existing entity nodes. (b) Handling a variety of complex reasoning types such as temporal and spatial in a common framework. Table 1 shows examples of questions that require different types of reasoning. Most of the research so far has focused on questions involving multi-hop reasoning (Bordes et al., 2015; Dubey et al., 2019; Berant et al., 2013), with relatively less past work on other types of reasoning. To our knowledge, there is no past work trying to handle multiple reasoning types within the same framework.

In SYGMA, we tackle these challenges through: (1) a modular design of the system and (2) using λ-calculus based intermediate representations. Modular design offers flexibility to isolate sub-tasks that need adaptation, thus avoiding data-intensive end-to-end adaptation. Motivated by NSQA (Kapanipathi et al., 2021) and ReTraCk (Chen et al., 2021),

[†]sumit.neelam@in.ibm.com, [‡]udit.sharma@in.ibm.com
[*]This work was done when authors were at IBM.

[1]Information about facts are explicitly mapped by making facts as primitive nodes.

Findings of the Association for Computational Linguistics: EMNLP 2022, pages 3866–3879
December 7-11, 2022

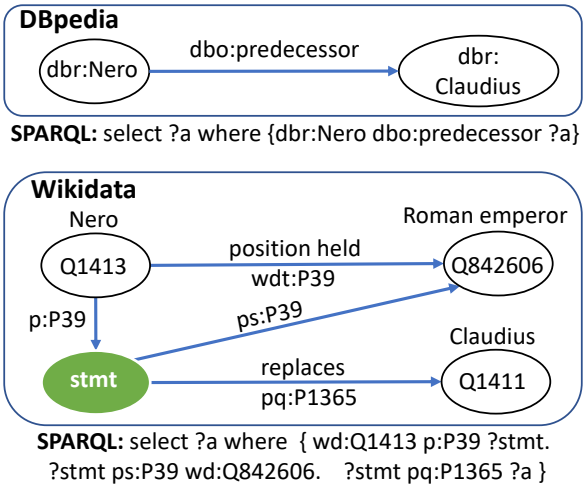

Figure 1: Nero's predecessor representation in DBpedia vs. Wikidata with SPARQL query to retrieve it.

Category	Example
Single-hop	*Who directed Titanic Movie?* **SPARQL:** select distinct ?a where { wd:Q44578 wdt:P57 ?a}
Multi-hop	*Which movie is directed by James Cameron starring Leonardo DiCaprio?* **SPARQL:** select distinct ?a where {?a wdt:P57 wd:Q42574. ?a wdt:P161 wd:Q38111. }
Temporal	*Who was the US President during cold war?* **SPARQL:** in Figure 3

Table 1: Examples of Single-hop, Multi-hop and Temporal reasoning questions on Wikidata.

we use a pipeline of sub-modules namely: Abstract Meaning Representation (Banarescu et al., 2013), Entity Linking, and Relation Linking. λ-calculus based representation is used as a common framework to represent and handle a variety of reasoning types and knowledge representation. We demonstrate the generalizability of SYGMA through experimental evaluations using a variety of datasets that include: (a) DBpedia and Wikidata as knowledge bases, and (b) multi-hop and temporal as the reasoning types.

A summary of our main contributions:

- A modular approach, called SYGMA, for generalizable KBQA that uses λ-calculus based intermediate logical representations towards enabling adaptation to: (a) multiple knowledge bases, specifically DBpedia and Wikidata, and (b) multiple reasoning types, multi-hop and temporal reasoning.
- Experimental results show SYGMA is able to achieve its generalization goal, also achieving state-of-the-art performance on WebQSP-WD, and competitive performance on LC-QuAD 1.0 and QALD-9 datasets. We also report baseline accuracies on Simple WebQuestions (Diefenbach et al., 2017b) and temporal question answering dataset called TempQA-WD (Neelam et al., 2022).

2 Related Work

Early work on automatic question answering considered extracting answers from text (Hirschman and Gaizauskas, 2001; Voorhees, 2000). Organizing knowledge in a structured format (unlike unstructured knowledge in text) started gaining momentum with focus on semantic web, leading to creation of large scale KBs such as Freebase (Bollacker et al., 2008), DBpedia (Lehmann et al., 2014) and Wikidata (Pellissier Tanon et al., 2016). KBQA has emerged out of these developments, as a natural language interface to structured knowledge resources. While early work on KBQA has focused on simple factoid questions (Yahya et al., 2012; Bordes et al., 2015), latest advances in NLP and knowledge representation, has moved the focus to complex questions (Hu et al., 2018; Luo et al., 2018).

Among the existing approaches for complex KBQA, the most successful ones in recent times are semantic parsing based (Singh et al., 2018; Kapanipathi et al., 2021; Zou et al., 2014; Hu et al., 2021), where a question understanding module is used to convert natural language questions into their corresponding logical forms. The logical form serves to provide a structured way of representing and executing reasoning needed to answer complex questions. Different systems use different parsing techniques for question understanding, for example, (Luo et al., 2018) use dependency parsing and (Kapanipathi et al., 2021) use Abstract Meaning Representation (AMR). From design point of view, KBQA systems can be grouped into: (a) end-to-end trainable (Sorokin and Gurevych, 2018; Jia et al., 2018a; Saxena et al., 2020, 2021; Jia et al., 2021; Mavromatis et al., 2021) and (b) modular (Kapanipathi et al., 2021; Hu et al., 2021; Zou et al., 2014). Modular approaches are typically easily adaptable and interpretable.

With the goal of generalization across multiple KBs and multiple reasoning types, our approach SYGMA is designed in a modular fashion. Moreover, it also uses AMR parse of the question and

transforms that further into a λ-calculus based representation. As will be seen later in the paper, these design aspects are aimed at providing a flexible framework to adapt SYGMA to new KBs and new reasoning types. In contrast, most of the past KBQA approaches are tuned to a specific KB and a specific set of reasoning types. For example, NSQA (Kapanipathi et al., 2021) and EDGQA (Hu et al., 2021) are tuned to work only on DBpedia, whereas GGNN (Sorokin and Gurevych, 2018) is tuned to work on Wikidata. Likewise, NSQA (Kapanipathi et al., 2021) and EDGQA (Hu et al., 2021) target only single/multi-hop reasoning, whereas TEQUILA (Jia et al., 2018b) and CronKGQA (Saxena et al., 2021) target only temporal reasoning.

KBQA Datasets: Many question answering datasets have been built over time to evaluate KBQA systems: Free917 (Cai and Yates, 2013), SimpleQuestions (Bordes et al., 2015), WebQuestions (Berant et al., 2013), QALD-9 (Usbeck et al., 2017), LC-QuAD 1.0 (Trivedi et al., 2017), LC-QuAD 2.0 (Dubey et al., 2019), CRONQUESTIONS (Saxena et al., 2021), TimeQuestions (Jia et al., 2021). Most of these datasets are built with goal of evaluating a specific KBQA approach. As a result, these datasets are also typically restricted to specific KB and reasoning type combinations. For example, QALD-9 is a dataset build for DBpedia to evaluate multi-hop reasoning and TempQA-WD is a dataset to evaluate temporal reasoning on Wikidata.

3 SYGMA: System Description

Figure 2 shows overall architecture of SYGMA. Motivated by Kapanipathi et al. (2021), SYGMA is designed as a modular system, where multiple sub-modules performing distinct sub-tasks are stitched together in a pipeline. Each of these sub-modules are built independent of each other with sub-task specific data. Such a design offers greater flexibility to push ahead our goal of generalization, because it is possible to isolate those sub-tasks that need adaptation and restrict adaptation effort only to the corresponding sub-modules, unlike data-intensive end-to-end trained systems.

KB Generalization: Towards the goal of KB generalization, the sub-modules are grouped into two processing stages, as shown in Figure 2: 1) *KB-Agnostic Question Understanding* that transforms Natural Language (NL) question into a KB-agnostic logical representation of the question, and 2) *KB-Specific Question Mapping and Reasoning* that maps the elements of the KB-agnostic logical representation onto the vocabulary of the KB to first build a KB-specific logical representation of the question and then transform that further into a SPARQL query that when executed over the KB would fetch the answer. Figure 3 gives an illustration of the outputs generated at different intermediate stages of the pipeline for an example NL question. Note that, in our effort to achieve generalization across KBs, it is not possible to avoid the KB-specific components completely out from the system. Through above design, all we try to achieve is to minimize the adaptation effort by pushing all KB-specific processing towards the end of the pipeline, so the preceding part of the pipeline would remain common across all the KBs and the efforts needed to support new KBs is limited only to the later part of the pipeline.

Reasoning Type Generalization: Support for generalization across reasoning types is achieved through: (1) use of λ-calculus for logical representation that captures different reasoning types as higher order functions, and (2) incorporation of transformation heuristics within logical representation modules that spans across both the stages of the pipeline, as shown in Figure 2. λ-calculus provides a flexible framework not only to handle a variety of reasoning types but also to handle representation differences across KBs. Over that, the transformation modules incorporate necessary high-level knowledge about different reasoning constructs and their transformations that are needed to handle specifics of the different reasoning types. Note that we chose to adopt such a neuro-symbolic approach of combining acquired knowledge with the learned knowledge in a modular fashion. On this aspect, we would like to highlight that end-to-end trained systems do not have any distinctive advantage, because to acquire such a knowledge through data is difficult and would need large amount of manual effort in terms of data collection and labeling, to cover a wide range of reasoning types.

Before we get into the details of different modules, next we describe Lambda calculus, the formalism used in our system for logical representation.

3.1 Lambda Calculus

λ-calculus, by definition, is considered the smallest universal programming language that expresses

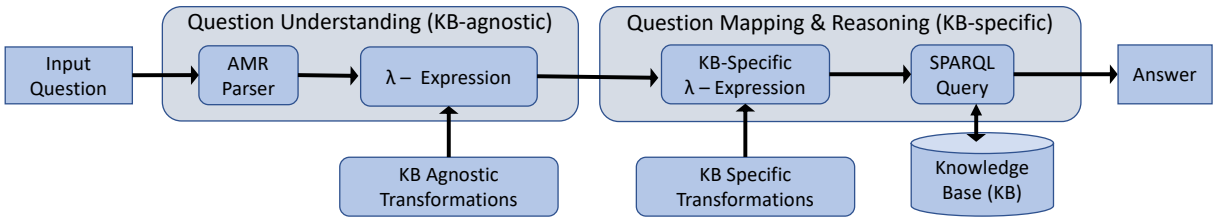

Figure 2: Architecture of SYGMA that shows the pipeline with modules and intermediate representations.

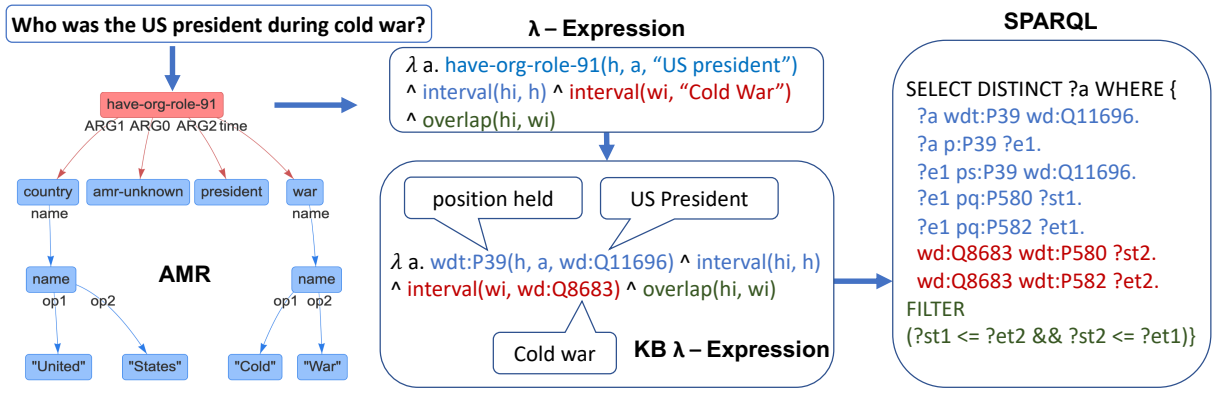

Figure 3: An illustration of the outputs at the intermediate stages of the pipeline in SYGMA.

any computable function. In this work, we have adopted *Typed λ-Calculus* (Zettlemoyer and Collins, 2012) because it supports addition of new higher order functions (a requirement in our framework to handle multiple reasoning types). We use constants, logical connectives (like AND, OR, NEGATION) and functions (like argmin, argmax, count) as in (Zettlemoyer and Collins, 2012). Apart from these, we have also added a few other functions to support different reasoning types. E.g., to support temporal reasoning, we have added temporal functions such as *interval*, *overlap*, *before*, *after*, and so on. *interval* is used to represent time interval associated with an event[2]. *overlap*, *before* and *after* are used to represent comparison of time intervals of events. Below is an example NL question and its logical form, called λ-expression:

Question:	When was Barack Obama born?
Logical Form:	*λt. born(b,"Barack Obama") ∧ interval(t, b)*

Here, *b* is used to denote event *born(b, "Barack Obama")* and *interval(t, b)* represents the time interval of the event denoted by *b*. *t* corresponds to unknown variable, as given in the expression by *λt*.

[2]Note that, in this paper, we call facts that are true for a specific duration of time as events. For example *birth of Barack Obama* is an event.

3.2 KB-Agnostic Question Understanding

The goal here is to represent the semantics of the question logically in a KB-independent manner, i.e., to derive intermediate λ-expression of the question that is common across all the KBs. This is achieved in two steps: first the NL question is mapped onto its Abstract Meaning Representation (AMR), followed by further mapping onto corresponding λ-expression.

3.2.1 AMR

We use AMR (Abstract Meaning Representation) (Banarescu et al., 2013) to mediate between NL questions and their corresponding λ-expressions, because it provides an efficient intermediate form to further derive logical representation of the question semantics. AMR encodes the meaning of a sentence into a rooted directed acyclic graph where nodes represent concepts and edges represent relations. We adopt an action pointer transformer architecture of Zhou et al. (2021) for transition-based AMR parsing and self-training technique of Lee et al. (2020, 2022). We used a single AMR parsing model for all of our experiments which is trained on the combination of human annotated treebanks and a synthetic AMR corpus. Human annotated treebanks include AMR3.0 and 958 questions sentences (250 QALD train + 627 LC-QuAD 1.0 train + 81 TempQA-WD) annotated in-house. The syn-

thetic AMR includes around 100k self-trained sentences (∼27k from LC-QuAD 1.0/LC-QuAD 2.0 and ∼70k from SQuAD-2 (Rajpurkar et al., 2018) training data sets). Figure 3 shows an example AMR for the question *Who was US president during cold war?* Note that this representation has encoded cold war event as sub-event under the *time* edge. It also explicitly captures the before/after constraints as part of the *time* edge. If there are no constraints appearing under *time* edge, we treat that by default as overlap constraint.

3.2.2 KB-Agnostic Lambda Expression

A recursive algorithm is used to transform AMR into KB-agnostic λ-expression. Starting from root AMR node, other nodes deep/across the AMR graph (i.e., linked elements, entity/relation mentions) are traversed in a recursive fashion. While being at each node during the recursion, appropriate transformation heuristics as listed in Table 2 are applied to generate component λ-expression corresponding to that node. The transformations heuristics are basically high-level constructs, implemented as functions to map each AMR node into corresponding component λ-expression, by applying appropriate rule based on the current AMR frame and its components. At the end of recursion, the final λ-expression is generated as a conjunction of component λ-expressions, together with a projection variable and the required reasoning functions. AMR unknown/imperative constructs are used to identify the projection variable. Transformations as listed in the table includes: templates for base reasoning, templates covering specific reasoning types (e.g., temporal), templates for how AMR constructs can be used to isolate events and so on. A complete list of templates is given in Appendix A. Figure 3 shows an illustrative example of λ-expression constructed from AMR of a sample question. Note that λ-expression nicely represents semantic decomposition of the question, i.e., a logical composition of multiple sub-queries linked together through variables, that when resolved satisfying the reasoning constraints would obtain the answer.

3.3 KB-Specific Question Mapping and Reasoning

KB-specific segment encompasses a module to transform KB-agnostic λ-expression into KB-specific λ-expression and another module to transform that further into a SPARQL. Note that, to adapt SYGMA to new KBs, these modules need to be rebuilt for each KB added.

3.3.1 KB-Specific Lambda Expression

KB-specific λ-expression is structurally similar to the KB-agnostic λ-expression, except that all the mentions of entities and relations are mapped to the corresponding KB entities and relations. The process of such mapping is described below.

Entity Linking: The goal of Entity Linking is to map entity mentions in the KB-agnostic λ-expression of the question onto their corresponding KB entities. We use a recently proposed zero-shot entity linking approach called BLINK (Wu et al., 2020). For a question where entity mentions are already identified, bi-encoder piece of the BLINK is used to predict top-K entities. For this prediction, we use a pre-trained model built with entity dictionary of 5.9M English Wikipedia entities with mappings to their corresponding Wikidata and DBpedia entities. More details on the model is described in (Wu et al., 2020).

Relation Linking: The goal of relation linking is to map the relation mentions in the KB-agnostic λ-expression onto their corresponding KB-specific relations. To achieve this, we use state-of-the-art AMR-based relation linking approach as in (Naseem et al., 2021). We use their pre-trained models built for Wikidata and DBpedia. As described in (Naseem et al., 2021), this takes question text and its AMR graph as the input and returns a ranked list of KB relations. KB-specific λ-expression is constructed from KB-agnostic λ-expression by retaining its structure and appropriately modifying the Relation Slots and other details such as number of relations; their subjects, objects and surface forms. For the example question "Who was the US President during cold war?", as in Figure 3, Wikidata KB relations predicted are (given in brackets): *position held (P39)*, *start time (P580)*, *end time (P582)*.

3.3.2 SPARQL Query

This module maps KB-specific λ-expressions into SPARQL query through a deterministic approach. λ-expression contains one or more terms such that each term T_i is comprised of one or more predicates connected via $\wedge$ or $\vee$. Each construct in λ-expression is mapped to its equivalent SPARQL construct as per Table 3. Most of these rules are generic across KBs, except for a few rules that require KB-specific predicates during translation. For

Type	AMR A = (v/frame ...)	Lambda Expression L = ψ(v)
Base	(**v**/frame :arg0(**v0**/frame0) ... :argn(**vn**/framen))	frame(**v**, **v0**, ... **vn**) $\wedge$ ψ(**v0**) $\wedge$ ... $\wedge$ ψ(**vn**)
Base	(**v**/frame :**arg1(a/amr-unknown)** ... :argn(vn/framen))	λa. ψ(**v**)
Numerical	(**v**/frame :arg0(**v0**/frame0 :**quant(a/amr-unknown)**) ... :argn(vn/framen))	count(λ**v0**. ψ(**v**))
Temporal	(**v**/frame :arg0(v0/frame0) ... :argn(vn/framen) **:time(a/amr-unknown))**	λev. ψ(**v**) $\wedge$ interval(ev, **v**)
Temporal	(**v**/frame :arg0(a/amr-unknown) ... :argn(vn/framen) **:time(b/before** :op1(**n**/nested-frame)))	argmax(λa. ψ(**v**) , λa. λev. ψ(**n**) $\wedge$ interval(ev, **v**) $\wedge$ interval(en, **n**) $\wedge$ **before**(ev, en), 0, 1)
Temporal	(**v**/frame :arg0(a/amr-unknown) ... :argn(vn/framen) **:time(n**/nested-frame))	λa. ψ(**v**) $\wedge$ ψ(**n**) $\wedge$ interval(ev, **v**) $\wedge$ interval(en, **n**) $\wedge$ **overlap**(ev, en)
Spatial	(**b/be-located-at-91** :arg0(a/amr-unknown), **:mod(s/south)** :op1(**n**/nested-frame))	λa. ψ(**b**)$\wedge$ ψ(**n**) $\wedge$ coordinate(cb, **b**) $\wedge$ coordinate(cn, **n**) $\wedge$ **south**(cb, cn)

Table 2: Translation of AMR into KB-agnostic λ-expression

example, table contains rule for handling temporal reification for Wikidata, in which, *start time(P580)*, *end time(P582)*, or *point in time(P585)* connected to intermediate statement node are used to get the time interval. This can be altered to support other temporal KBs. A complete list of rules is given in Appendix A.

4 Evaluation

4.1 Implementation Details

In our implementation, we use Flow Compiler[3] (Chakravarti et al., 2019) to build the pipeline, that stitches together individual modules exposed as gRPC services on single virtual machine with 32 cores and 128GB memory. We employ single AMR and entity linking service across all datasets, and one service each for DBpedia and Wikidata for relation linking purpose. We use ANTLR[4] grammar to define λ-expressions, that includes rules to capture basic predicates, logical connectives (like *and, or, not*), basic functions (like *argmin, argmax, min, max*), temporal functions (like *interval, overlap, before, after, teenager, now, age*) and so on. Defined ANTLR grammar is used to check generated λ-expression syntax and to parse it as well. Module to transform KB-Specific λ-expression to SPARQL is implemented in Java using Apache Jena[5] SPARQL modules, that first creates SPARQL objects and then generates the final SPARQL query to run on target KB end-point[6]. The rest of the modules are implemented in Python and are exposed as gRPC services. We use GERBIL[7] (Usbeck et al., 2019) to compute performance metrics from pairs of gold answers and system generated answers (computed by the pipeline). We use standard performance metrics typically used for KBQA systems, namely macro precision, macro recall and F1.

4.2 Datasets

Our choice of datasets for evaluation is driven by the key goal of our approach, i.e., generalizability across KBs and reasoning types. In this paper, we perform evaluation on the following: 1) two different KBs: DBpedia and Wikidata[8], 2) two different reasoning types: multi-hop and temporal. Given there is no previous work (to our knowledge) on generalizability of KBQA systems, we could not find any dataset that covers all the evaluation dimensions listed above. As a result, we decided to evaluate our system on multiple datasets, each spanning one of the dimensions above. Table 6 lists 5 datasets we use for evaluation. Their details are: 1) QALD-9 (Usbeck et al., 2017) has 408 training and 150 test questions, 2) LC-QuAD 1.0 (Trivedi et al., 2017) has 4000 train and 1000 test set questions 3) SWQ-WD (Diefenbach et al., 2017b) has 14894 train and 5622 test set questions, 4) WebQSP-WD (Sorokin and Gurevych, 2018) has 2880 train and 1033 test set questions, and 5) TempQA-WD has no train set questions but 175 dev and 664 test questions. Note that TempQA-WD is a relatively new dataset and derived from TempQuestions (Jia et al., 2018a) dataset. We have chosen TempQA-WD because it is on Wikidata whereas original TempQuestions is on Freebase. The DBpedia specific datasets mentioned above (QALD-9 and LC-QuAD 1.0) are on different versions of DBpedia and hence

[3]https://github.com/IBM/flow-compiler
[4]https://www.antlr.org
[5]https://jena.apache.org/
[6]https://query.wikidata.org/
[7]https://github.com/dice-group/gerbil

[8]We did not consider adapting to Freebase as it is discontinued, hence no longer actively maintained and not up-to-date.

Type	Expression/Predicate E	SPARQL S = ϕ(E)
λ abstraction	λ**x.T**	SELECT DISTINCT ?**x** WHERE { ϕ(**T**) }
Count expression	count(λ**x.T**)	SELECT (COUNT(?**x**) AS ?c) WHERE { ϕ(**T**) }
Argmax expression	argmax(λ**x.T1**, λ**x**.λ**y**. **T2**, ***O***, ***L***)	SELECT DISTINCT ?**x** WHERE { ϕ(**T1**) ϕ(**T2**) } ORDER BY DESC(?**y**) LIMIT ***L*** OFFSET ***O***
KB Predicate	**<pred_iri>**(i, **s**/**<s_iri>**, **o**/**<o_iri>**)	?**s**/**<s_iri>** **<pred_iri>** ?**o**/**<o_iri>**.
Interval predicate for reified facts	wdt:**PID**(i, **s**/**<s_iri>**, **o**/**<o_iri>**) $\wedge$ interval(**e**, i)	?**s**/**<s_iri>** p:**PID** ?x. ?x ps:**PID** ?**o**/**<o_iri>**. ?x *pq:P580* ?$\mathbf{e}_{start}$. ?x *pq:P582* ?$\mathbf{e}_{end}$.
Overlap predicate	overlap(**e1, e2**)	FILTER(?$\mathbf{e1}_{start}$ <= ?$\mathbf{e2}_{end}$ && ?$\mathbf{e2}_{start}$ <= ?$\mathbf{e1}_{end}$)
Before predicate	before(**e1, e2**)	FILTER(?$\mathbf{e1}_{end}$ <= ?$\mathbf{e2}_{start}$)
After predicate	after(**e1, e2**)	FILTER(?$\mathbf{e1}_{start}$ >= ?$\mathbf{e2}_{end}$)

Table 3: Translation of KB-Specific λ-expression into SPARQL

DataSet →	WebQSP-WD			SWQ-WD			TempQA-WD		
System ↓	P	R	F1	P	R	F1	P	R	F1
GGNN	0.27	0.32	0.26	0.32	0.38	0.33	0.08	0.21	0.09
SYGMA	**0.32**	**0.36**	**0.31**	**0.42**	**0.55**	**0.44**	**0.32**	**0.34**	**0.32**

Table 4: SYGMA's Performance on Wikidata across reasoning types. P-Precision, R-Recall

DataSet →	LC-QuAD 1.0			QALD-9		
System ↓	P	R	F1	P	R	F1
WDAqua	0.22	0.38	0.28	0.26	0.26	0.25
gAnswer	n/a	n/a	n/a	0.29	0.32	0.29
QAMP	0.25	0.50	0.33	n/a	n/a	n/a
NSQA	0.44	0.45	0.44	0.31	0.32	0.31
EDGQA	**0.50**	**0.56**	**0.53**	**0.31**	**0.40**	**0.32**
SYGMA	0.47	0.48	0.47	0.29	0.30	0.29

Table 5: SYGMA's Performance on DBpedia. Baseline numbers are taken from Hu et al. (2021), Kapanipathi et al. (2021). P-Precision, R-Recall

Datasets	Knowledge Base	Reasoning
QALD-9	DBpedia	Multi-hop
LC-QuAD 1.0	DBpedia	Multi-hop
SWQ-WD	Freebase, Wikidata	Single-hop
WebQSP	Freebase, Wikidata	Multi-hop
TempQA-WD	Freebase, Wikidata	Temporal

Table 6: SYGMA relevant KBQA datasets and features

we used the appropriate version of DBpedia as per the dataset during evaluation. Notice that, we have performed temporal reasoning on Wikidata only, because we analysed Wikidata and DBpedia and found that temporal information is captured structurally well in Wikidata using reification in comparison to DBpedia. Moreover, all the recent temporal QA datasets are based on Wikidata.

Although LC-QuAD 2.0 dataset is a potential candidate for evaluation, since it is both on DBpedia and Wikidata, we could not use it because we found some inconsistencies in question texts. Also, although TimeQuestions (Jia et al., 2021) is potential candidate dataset, we could not use it as well for evaluation because of the discrepancies we noted. This dataset maps only answer entities to the corresponding Wikidata entities and do not validate if the answers are up-to-date and if all the required facts needed to retrieve that answer are present in the Wikidata KB or not. On the other hand, we used TempQA-WD for evaluation as it is manually validated and has SPARQL queries to refresh the answers. Note that answers to temporal questions can change over time. To validate the extent of the discrepancy we compared the answers for the overlapping set of 743 questions between TempQA-WD and TimeQuestions, and found that only 50.9% of the answers matching.

4.3 Baselines

We compare our system with various baselines systems supporting question answering on DBpedia and Wikidata. The λ-expression generation and the transformation heuristics of the baseline system are tuned with 200 questions from SWQ-WD train set, 200 from LC-QuAD 1.0 train set, 175 from TempQA-WD dev set. Evaluation is on test sets of all five data sets.

EDGQA (Hu et al., 2021) (current state-of-the-art on both LC-QuAD 1.0 and QALD-9 datasets), NSQA (Kapanipathi et al., 2021), WDAqua (Diefenbach et al., 2017a), gAnswer (Zou

et al., 2014) and QAMP (Vakulenko et al., 2019) systems provides baseline for DBpedia based datasets. For Wikidata based datasets we use GGNN (Sorokin and Gurevych, 2018) as the baseline as it is the only known benchmark for WebQSP-WD dataset. To evaluate SWQ-WD dataset, we trained GGNN system on its train data. To evaluate on TempQA-WD dataset, we trained GGNN on the combination of train set of WebQSP-WD and the dev set of TempQA-WD. We used GGNN default parameters[9] for both datasets training.

5 Results & Discussion

Table 4 and 5 show performance comparison of SYGMA against all the baselines (described in Section 4.3) on all the datasets (described in Section 4.2). We achieve state-of-the-art performance on all the Wikidata datasets WebQSP-WD, SWQ-WD and TempQA-WD, while achieving reasonable performance on DBpedia datasets QALD-9 and LC-QuAD 1.0. SYGMA achieves second best accuracy on LC-QuAD 1.0. Note that GGNN is the only comparable baseline for WebQSP-WD dataset. Performance of GGNN on TempQA-WD dataset clearly points to the limitation of end-to-end trained systems, i.e., the requirement of large amount of training data to adapt to a new reasoning need.

Note that SYGMA is the only system able run on all the datasets, which indeed demonstrate the main goal of our work. Although it is not able to achieve top performance on DBpedia datasets, generalizability across KBs and reasoning types with performance comparable to state-of-the-art is the key distinguishing aspect of SYGMA.

5.1 Ablation Study

We also performed module-level evaluation of SYGMA and ablation studies as described below.

5.1.1 AMR

Table 7 show the performance of the AMR parser on the 5 development sets. Evaluation metrics used are *Smatch* (standard measure used to measure AMRs) and *Exact Match* (% of questions that match fully with ground truth AMR).

5.1.2 Entity Linking

Table 8 shows independent performance of entity linking on different datasets. Question level accuracy is computed by considering the correctness of all the entities in each question, whereas mention level accuracy is computed on each mention. Note that entity linking performance for Wikidata datasets is low in comparison to DBpedia. This is because all those datasets are adopted from corresponding Freebase dataset and the way entities are represented in these two KBs is different. Also, research on entity linking for Wikidata is gaining momentum only recently.

Dataset	KB	Smatch	Exact Match
LC-QuAD 1.0	DBpedia	87.6	30.0
QALD-9	DBpedia	89.3	41.8
WebQSP-WD	Wikidata	88.0	43.8
SWQ-WD	Wikidata	83.0	37.8
TempQA-WD	Wikidata	89.6	39.8

Table 7: AMR parser performance on dev sets

Dataset	KB	Accuracy (%)	
		MentionLevel	QuestionLevel
LC-QuAD 1.0	DBpedia	86.8 (91.9)	84.0 (90.5)
QALD-9	DBpedia	89.5 (94.3)	89.8 (93.9)
TempQA-WD	Wikidata	74.0 (82.5)	57.1 (69.7)
SWQ-WD	Wikidata	72.3 (83.2)	70.1 (81.6)

Table 8: Entity linking performance on dev sets with gold mentions, Hits@5 scores in parentheses.

A closer look into the results show that accuracy on nominal entities (which BLINK is not trained on) drags down the question level accuracy on TempQA-WD dataset. For SWQ-WD dataset, the question size becomes bottleneck as that leads to lack of adequate context. For the question "*what is John Steppling's place of birth?*" taken from SWQ-WD dataset, due to insufficient context, EL incorrectly links *John Steppling* to Wikidata entity *Q16150539*(playwright) instead of *Q3182525*(actor). Also, for TempQA-WD dataset question "*who won best Actor when Alfred Junge won Best Art Direction?*", correct entity for *Best Art Direction* is *Q22253131*(Academy Award), but EL wrongly returns *Q28805401*(Guldbagge Award)

5.1.3 Relation Linking

Table 9 shows independent performance of the relation linking module. Metrics here point to the fact that relation linking is still a challenging task, especially in case of multi-hop reasoning and temporal reasoning, where the query graph is disconnected across events. Similar to entity linking, relation linking performance for Wikidata is low in comparison to DBpedia because it is relatively new.

[9] https://github.com/UKPLab/coling2018-graph-neural-networks-question-answering

One of the reasons, relation linking suffers is becuase of lack of context, E.g., for SWQ-WD dataset question "*Where is Zenon Grocholewski from?*", correct relation is *P27*(country of citizenship) for entity *Q189728*, but RL returns *P19*(place of birth) as output. Further, current relation linking fails to capture non-temporal reified relation in Wikidata. Consider TempQA-WD question "*who played Will Scarlett in the 1991 Robin Hood?*", Wikidata stores link between *Robin Hood(Q689658)* and *Will Scarlet(Q339019)* through reification using relation *P161*(cast member) and *P453*(character role), which are not considered by current module.

Dataset	KB	Precision	Recall	F1
LC-QuAD 1.0	DBpedia	0.52	0.50	0.50
QALD-9	DBpedia	0.55	0.53	0.53
TempQA-WD	Wikidata	0.43	0.43	0.42
SWQ	Wikidata	0.67	0.68	0.67

Table 9: Relation linking performance on dev sets.

Empirical results on individual modules convey that Entity Linking and Relation Linking perform relatively better on DBpedia in comparison to Wikidata. But it does not reflect on overall performance on DBpedia based datasets(QALD-9 performance is relatively low). One of the reasons is AMR performance is low(Exact Match 30%) for LC-QuAD 1.0 compared to Wikidata based datasets(Exact Match 40%) Additionally, as mentioned earlier, KB-Agnostic λ module is developed looking at only LC-QuAD 1.0 dataset from DBpedia, but not with QALD-9, resulting in partially handling comparative and superlative questions.

5.1.4 Pipeline Ablation Study

We also performed pipeline ablation study to analyze the impact of individual SYGMA modules on the overall performance, using TempQA-WD dev set questions and 100 dev set questions from LC-QuAD 1.0. For this, we manually annotated those questions with ground truths for all the intermediate modules. Table 10 shows the results. In the table, GT-x denote the case where ground truth up to module x is fed directly into the system. For example, GT-AMR denotes directly feeding of the AMR ground truth into λ-expression generation module. The results show a large jump in accuracy (in both the datasets) when fed directly with the ground truth entities (GT-EL) and ground truth relations (GT-RL). This points to the need for improved entity/relation linking on both KBs.

	TempQA-WD			LC-QuAD 1.0		
	P	R	F1	P	R	F1
NO GT	0.47	0.50	0.47	0.50	0.52	0.50
GT-AMR	0.50	0.51	0.50	0.51	0.53	0.51
GT-λ	0.52	0.53	0.52	0.52	0.53	0.52
GT-EL	0.60	0.62	0.60	0.63	0.66	0.64
GT-RL	0.92	0.93	0.92	0.99	0.98	0.98
GT-KB-λ	0.93	0.93	0.93	1.0	0.99	0.99
GT-Sparql	1.0	1.0	1.0	1.0	1.0	1.0

Table 10: Ablation Study on TempQA-WD and LC-QuAD 1.0 dev sets. P-Precision, R-Recall

6 Conclusion

In this paper, we described SYGMA, a KBQA system that generalize across KBs and reasoning types. The key distinguishing features of our approach are its modular design and its use of AMR/λ-Calculus based question understanding and decomposition module to obtain KB-agnostic logical representation of the question. It also includes KB-specific question mapping and reasoning modules and transformation heuristics to handle multiple KBs and multiple reasoning types. An experimental evaluation of SYGMA on datasets spanning multiple KBs and reasoning types indeed demonstrates its generalization capabilities. Among many potential opportunities for future work, we believe improving the individual performance of relation linking module is critical and holds promise for significantly improving the overall performance.

7 Limitations

Although modular design of our system SYGMA is a beneficial from the point-of-view of our generalization goal, it makes it more sensitive to the performance of the individual modules, because non-overlapping errors across all the modules can affect the overall performance. Currently, our system is using only the top-1 result from entity linking and relation linking modules, while top-k ranked list returned from these modules are available. Extending system to use top-k can potentially improve the overall accuracy. One of the weak links in our system is relation linking module. As shown through independent evaluation of this module, the performance of the overall system can potentially be improved significantly with improved relation linking alone. Especially, given KBQA efforts on Wikidata has started only recently, enity linking and relation linking on Wikidata are still not matured enough and have good scope for improvement.

References

L. Banarescu, C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of Linguistic Annotation Workshop 2013*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA. Association for Computational Linguistics.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, page 1247–1250, New York, NY, USA. Association for Computing Machinery.

Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. 2015. Large-scale simple question answering with memory networks. *CoRR*, abs/1506.02075.

Qingqing Cai and Alexander Yates. 2013. Large-scale semantic parsing via schema matching and lexicon extension. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 423–433, Sofia, Bulgaria. Association for Computational Linguistics.

Rishav Chakravarti, Cezar Pendus, Andrzej Sakrajda, Anthony Ferritto, Lin Pan, Michael Glass, Vittorio Castelli, J William Murdock, Radu Florian, Salim Roukos, and Avi Sil. 2019. CFO: A framework for building production NLP systems. 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): System Demonstrations*, pages 31–36, Hong Kong, China. Association for Computational Linguistics.

Shuang Chen, Qian Liu, Zhiwei Yu, Chin-Yew Lin, Jian-Guang Lou, and Feng Jiang. 2021. ReTraCk: A flexible and efficient framework for knowledge base question answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 325–336, Online. Association for Computational Linguistics.

Dennis Diefenbach, Kamal Singh, and Pierre Maret. 2017a. Wdaqua-core0: A question answering component for the research community. In *Semantic Web Evaluation Challenge*, pages 84–89. Springer.

Dennis Diefenbach, Thomas Pellissier Tanon, Kamal Deep Singh, and Pierre Maret. 2017b. Question answering benchmarks for wikidata. In *Proceedings of the ISWC 2017 Posters & Demonstrations and Industry Tracks co-located with 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 23rd - to - 25th, 2017.*

Mohnish Dubey, Debayan Banerjee, Abdelrahman Abdelkawi, and Jens Lehmann. 2019. *LC-QuAD 2.0: A Large Dataset for Complex Question Answering over Wikidata and DBpedia*, pages 69–78.

Bin Fu, Yunqi Qiu, Chengguang Tang, Yang Li, Haiyang Yu, and Jian Sun. 2020. A survey on complex question answering over knowledge base: Recent advances and challenges. *arXiv preprint arXiv:2007.13069*.

Lynette Hirschman and Rob Gaizauskas. 2001. Natural language question answering: The view from here. *Natural Language Engineering*, 7:275 – 300.

Sen Hu, Lei Zou, and Xinbo Zhang. 2018. A state-transition framework to answer complex questions over knowledge base. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2098–2108, Brussels, Belgium. 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 - 20th International Semantic Web Conference, ISWC 2021, Virtual Event, October 24-28, 2021, Proceedings*, volume 12922 of *Lecture Notes in Computer Science*, pages 128–145. Springer.

Zhen Jia, Abdalghani Abujabal, Rishiraj Saha Roy, Jannik Strötgen, and Gerhard Weikum. 2018a. Tempquestions: A benchmark for temporal question answering. In *Companion Proceedings of the The Web Conference 2018*, WWW '18, page 1057–1062, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.

Zhen Jia, Abdalghani Abujabal, Rishiraj Saha Roy, Jannik Strötgen, and Gerhard Weikum. 2018b. Tequila: Temporal question answering over knowledge bases. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, page 1807–1810, New York, NY, USA. Association for Computing Machinery.

Zhen Jia, Soumajit Pramanik, Rishiraj Saha Roy, and Gerhard Weikum. 2021. Complex temporal question answering on knowledge graphs. In *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, pages 792–802. ACM.

Pavan Kapanipathi, Ibrahim Abdelaziz, Srinivas Ravishankar, Salim Roukos, Alexander Gray, Ramon Astudillo, Maria Chang, Cristina Cornelio, Saswati Dana, Achille Fokoue, et al. 2021. Leveraging abstract meaning representation for knowledge base

question answering. *Findings of the Association for Computational Linguistics: ACL*.

Young-Suk Lee, Ramon Fernandez Astudillo, Tahira Naseem, Revanth Gangi Reddy, Radu Florian, and Salim Roukos. 2020. Pushing the limits of amr parsing with self-learning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*.

Young-Suk Lee, Ramón Astudillo, Hoang Thanh Lam, Tahira Naseem, Radu Florian, and Salim Roukos. 2022. Maximum bayes smatch ensembel distillation for amr parsing. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5379–5392.

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, SÃ¶ren Auer, and Christian Bizer. 2014. Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 6.

Kangqi Luo, Fengli Lin, Xusheng Luo, and Kenny Zhu. 2018. Knowledge base question answering via encoding of complex query graphs. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2185–2194, Brussels, Belgium. Association for Computational Linguistics.

Costas Mavromatis, Prasanna Lakkur Subramanyam, Vassilis N. Ioannidis, Soji Adeshina, Phillip R. Howard, Tetiana Grinberg, Nagib Hakim, and George Karypis. 2021. Tempoqr: Temporal question reasoning over knowledge graphs.

Tahira Naseem, Srinivas Ravishankar, Nandana Mihindukulasooriya, Ibrahim Abdelaziz, Young-Suk Lee, Pavan Kapanipathi, Salim Roukos, Alfio Gliozzo, and Alexander Gray. 2021. A semantics-aware transformer model of relation linking for knowledge base question answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 256–262, Online. Association for Computational Linguistics.

Sumit Neelam, Udit Sharma, Hima Karanam, Shajith Ikbal, Pavan Kapanipathi, Ibrahim Abdelaziz, Nandana Mihindukulasooriya, Young-Suk Lee, Santosh K. Srivastava, Cezar Pendus, Saswati Dana, Dinesh Garg, Achille Fokoue, G. P. Shrivatsa Bhargav, Dinesh Khandelwal, Srinivas Ravishankar, Sairam Gurajada, Maria Chang, Rosario Uceda-Sosa, Salim Roukos, Alexander G. Gray, Guilherme Lima, Ryan Riegel, Francois P. S. Luus, and L. Venkata Subramaniam. 2022. A benchmark for generalizable and interpretable temporal question answering over knowledge bases. *CoRR*, abs/2201.05793.

Thomas Pellissier Tanon, Denny Vrandečić, Sebastian Schaffert, Thomas Steiner, and Lydia Pintscher. 2016. From freebase to wikidata: The great migration. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, page 1419–1428, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers*, pages 784–789. Association for Computational Linguistics.

Apoorv Saxena, Soumen Chakrabarti, and Partha P. Talukdar. 2021. Question answering over temporal knowledge graphs. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 6663–6676. Association for Computational Linguistics.

Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. 2020. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4498–4507, Online. Association for Computational Linguistics.

Kuldeep Singh, Andreas Both, Arun Sethupat, and Saeedeh Shekarpour. 2018. Frankenstein: a platform enabling reuse of question answering components. In *European Semantic Web Conference*, pages 624–638. Springer.

Daniil Sorokin and Iryna Gurevych. 2018. Modeling semantics with gated graph neural networks for knowledge base question answering. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3306–3317. Association for Computational Linguistics.

Priyansh Trivedi, Gaurav Maheshwari, Mohnish Dubey, and Jens Lehmann. 2017. Lc-quad: A corpus for complex question answering over knowledge graphs. In *Proceedings of the 16th International Semantic Web Conference (ISWC)*, pages 210–218. Springer.

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 Evaluation Challenge*, pages 59–69. Springer International Publishing.

Ricardo Usbeck, Michael Röder, Michael Hoffmann, Felix Conrads, Jonathan Huthmann, Axel-Cyrille Ngonga Ngomo, Christian Demmler, and Christina Unger. 2019. Benchmarking question answering systems. *Semantic Web*, 10(2):293–304.

Svitlana Vakulenko, Javier Fernández, Axel Polleres, Maarten de Rijke, and Michael Cochez. 2019. Message passing for complex question answering over

knowledge graphs. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM2019*, pages 1431–1440, Beijing, China. ACM.

Ellen M. Voorhees. 2000. Overview of the TREC-9 question answering track. In *Proceedings of The Ninth Text REtrieval Conference, TREC 2000, Gaithersburg, Maryland, USA, November 13-16, 2000*, volume 500-249 of *NIST Special Publication*. National Institute of Standards and Technology (NIST).

Ledell Wu, Fabio Petroni, Martin Josifoski, Sebastian Riedel, and Luke Zettlemoyer. 2020. Scalable zero-shot entity linking with dense entity retrieval. In *EMNLP*, pages 6397–6407.

Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, Maya Ramanath, Volker Tresp, and Gerhard Weikum. 2012. Natural language questions for the web of data. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, page 379–390, USA. Association for Computational Linguistics.

Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 201–206, Berlin, Germany. Association for Computational Linguistics.

Luke S. Zettlemoyer and Michael Collins. 2012. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *CoRR*, abs/1207.1420.

Jiawei Zhou, Tahira Naseem, Ramon Fernandez Astudillo, and Radu Florian. 2021. Amr parsing with action-pointer transformer. *https://arxiv.org/abs/2104.14674*.

Lei Zou, Ruizhe Huang, Haixun Wang, Jeffrey Xu Yu, Wenqiang He, and Dongyan Zhao. 2014. Natural language question answering over rdf: A graph data driven approach. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, page 313–324, New York, NY, USA. Association for Computing Machinery.

A Appendix

A.1 AMR to Lambda Translation

Table 11 shows the translation rules for transforming AMR frames (high level) into corresponding λ-expressions. Type denotes where the rule is applied. We have shown four different types of reasoning used in the SYGMA. More such transformations can be added to support additional reasoning types. These templates rely on the AMR constructs to derive the required reasoning functions.

- **Base Templates:** These transformations capture general multi-hop question meaning in terms of conjunction of different predicates coming from the AMR graph. These also include simple projection of variables that question is expecting to output or boolean(true/false) output in case of boolean questions. Second base rule gives an example of simple projection based on *amr-unknown*. If there are no *amr-unknown* variables then the question becomes boolean question and its return type is decided as true/false.

- **Numerical Templates:** These transformations capture all the numerical reasoning that is supported by the system. This category includes simple count questions, min/max or argmin/argmax or comparative questions. We use the AMR modifiers or the frames like *have-degree-91* to derive the required numerical reasoning to be performed for these type of questions. For example AMR's *quant* modifier for the unknown variable results in count operator in λ-expression, which gets translated to SPARQL count question later in the pipeline. Following are example questions for numerical reasoning:

count:	How many languages are spoken in Turkmenistan?
min/max:	When did Romney first run for president?
argmin/argmax:	What is the highest mountain in Italy?
comparative:	Is Lake Baikal bigger than the Great Bear Lake?

- **Temporal Templates:** These are the Transformations that capture the temporal constraints coming in the question. We rely on AMRs temporal constructs like *time/date-entity* etc. to decide if a question needs any temporal reasoning. Depending on additional constraints on the time edge like before/after/ordinal, we decide on the kind of reasoning to be performed. In the Table 11, we described all the temporal constructs that we encountered while working on the TempQA-WD dataset.

Type	AMR A = (v/frame . . .)	Lambda Expression L = ψ(v)
Base	(**v**/frame :arg0(**v0**/frame0) . . . :argn(**vn**/framen))	frame(**v**, **v0**, . . . **vn**) $\wedge$ ψ(**v0**) $\wedge$ . . . $\wedge$ ψ(**vn**)
Base	(**v**/frame :**arg1(a/amr-unknown)** . . . :argn(vn/framen))	λa. ψ(**v**)
Numerical	(**v**/frame :arg0(**v0**/frame0 :**quant(a/amr-unknown)**) . . . :argn(vn/framen))	count(λ**v0**. ψ(**v**))
Numerical	(**v**/frame :arg0(a/amr-unknown) ... :argn(vn/framen) **:mod(f/first)**)	min(λa. ψ(**v**), 0, 1)
Numerical	(**v**/frame :arg0(a/amr-unknown) ... :argn(vn/framen) **:mod(f/last)**)	max(λa. ψ(**v**), 0, 1)
Numerical	(**v**/frame :arg0(**v0**/frame0:mod(a/amr-unknown)) . . . :argn (**vn**/frame :arg1-of(h2/**have-quant-91** :arg3(l/**most**))))	argmax(λ**v0**. ψ(**v**), λ**v0**. λ**vn**. ψ(**vn**), 0, 1)
Numerical	(**v**/frame :arg0(**v0**/frame0:mod(a/amr-unknown)) . . . :argn (**vn**/frame :arg1-of(h2/**have-quant-91** :arg3(l/**least**))))	argmin(λ**v0**. ψ(**v**), λ**v0**. λ**vn**. ψ(**vn**), 0, 1)
Numerical	(**v**/frame :arg0(a/amr-unknown) . . . :argn(**vn**/framen :arg1-of (h2/**have-degree-91** :arg3(m/**more**) :arg4(**vm**/framem :arg1-of(**n**/nested-frame))))	λa. ψ(**v**) $\wedge$ ψ(**n**) $\wedge$ cmp(**vn**, **vm**, $>$)
Numerical	(**v**/frame :arg0(a/amr-unknown) . . . :argn(**vn**/framen :arg1-of (h2/**have-degree-91** :arg3(m/**less**) :arg4(**vm**/framem :arg1-of(**n**/nested-frame))))	λa. ψ(**v**) $\wedge$ ψ(**n**) $\wedge$ cmp(**vn**, **vm**, $<$)
Temporal	(**v**/frame :arg0(v0/frame0) . . . :argn(vn/framen) **:time(a/amr-unknown))**	λev. ψ(**v**) $\wedge$ interval(ev, **v**)
Temporal	(**v**/frame :arg0(a/amr-unknown) . . . :argn(vn/framen) **:time(b/before** :op1(**n**/nested-frame)))	argmax(λa. ψ(**v**) , λa. λev. ψ(**n**) $\wedge$ interval(ev, **v**) $\wedge$ interval(en, **n**) $\wedge$ **before**(ev, en), 0, 1)
Temporal	(**v**/frame :arg0(a/amr-unknown) . . . :argn(vn/framen) **:time(a/after** :op1(**n**/nested-frame)))	argmin(λa. ψ(**v**) , λa. λev. ψ(**n**) $\wedge$ interval(ev, **v**) $\wedge$ interval(en, **n**) $\wedge$ **after**(ev, en), 0, 1)
Temporal	(**v**/frame :arg0(a/amr-unknown) . . . :argn(vn/framen) **:time**(**n**/nested-frame))	λa. ψ(**v**) $\wedge$ ψ(**n**) $\wedge$ interval(ev, **v**) $\wedge$ interval(en, **n**) $\wedge$ **overlap**(ev, en)
Temporal	(**v**/frame :arg0(a/amr-unknown) :argn(vn/framen) :**ord(o/ordinal-entity** :value **x**))	argmin(λa. ψ(**v**) , λa. λev. interval(ev, **v**), **x**+1, 1)
Temporal	(**v**/frame :arg0(a/amr-unknown) :argn(vn/framen) :**ord(o/ordinal-entity** :value **-1**))	argmax(λa. ψ(**v**) , λa. λev. interval(ev, **v**), 0, 1)
Temporal	(**v**/frame :arg0(a/amr-unknown) . . . :argn(vn/framen) **:time(n/now))**	λa. ψ(**v**) $\wedge$ interval(ev, **v**) $\wedge$ interval(en, now()) $\wedge$ overlap(ev, en)
Temporal	(**v**/frame :arg0(a/amr-unknown) . . . :argn(vn/framen) **:time(d/date-entity** :month mm :day dd :year yyyy))	λa. ψ(**v**) $\wedge$ interval(en, date("dd-mm-yyyy")) $\wedge$ interval(ev, **v**) $\wedge$ overlap(ev, en)
Temporal	(**v**/frame :arg0(a/amr-unknown) . . . :argn(vn/framen) **:time(t/teenager** :domain(**n**/nested-frame)))	λa. ψ(**v**) $\wedge$ interval(ev, **v**) $\wedge$ teenager(en, **n**) $\wedge$ overlap(ev, en)
Spatial	(**b/be-located-at-91** :arg0(a/amr-unknown), **:mod(s/south)** :op1(**n**/nested-frame))	λa. ψ(**b**)$\wedge$ ψ(**n**) $\wedge$ coordinate(cb, **b**) $\wedge$ coordinate(cn, **n**) $\wedge$ **south**(cb, cn)

Table 11: AMR to KB-agnostic λ-expression Translation Rules (full rules)

Since we saw teenager being very prominent construct in many question, we explicitly added a this as a function to support that reasoning type. We also use synonyms like prior/precedes etc. to be treated similar to before temporal reasoning construct present in the table. These synonyms are captured as part of the KB-Agnostic Transformation module. Below are different examples of temporal questions:

overlap:	Who was the US president during the cold war?
before/after:	Who was London mayor before Boris Johnson?
ordinal:	Who was the first host of the tonight show?

- **Spatial Templates:** We have added a single template only to show how special kind of spatial reasoning queries like north of/ above a certain region or south of /below a certain region can be extended into our current framework. Please note that current SYGMA doesn't support any of these constructs, but can be easily extended in future.

A.2 KB Specific Lambda to SPARQL translation

Each KB-specific λ-expression construct is mapped to an equivalent SPARQL construct, as templates given in Table 12. λ-expressions in our setup can be broadly grouped into 6 categories: λ *abstraction*, *argmin*, *argmax*, *min*, *max* and, *count* expression. Table 12 gives mapping for each of them. λ *abstraction* template takes care of simple multi-hop rules and projection scenar-

Type	Expression/Predicate E	SPARQL S = ϕ(E)
λ abstraction	λ**x.T**	SELECT DISTINCT ?**x** WHERE { ϕ(**T**) }
Count expression	count(λ**x.T**)	SELECT (COUNT(?**x**) AS ?c) WHERE { ϕ(**T**) }
Argmin expression	argmin(λ**x.T1**, λ**x**.λ**y**. **T2**, ***O***, ***L***)	SELECT DISTINCT ?**x** WHERE { ϕ(**T1**) ϕ(**T2**) } ORDER BY ?**y** LIMIT ***L*** OFFSET ***O***
Argmax expression	argmax(λ**x.T1**, λ**x**.λ**y**. **T2**, ***O***, ***L***)	SELECT DISTINCT ?**x** WHERE { ϕ(**T1**) ϕ(**T2**) } ORDER BY DESC(?**y**) LIMIT ***L*** OFFSET ***O***
Min expression	min(λ**x.T**, ***O***, ***L***)	SELECT DISTINCT ?**x** WHERE { ϕ(**T**) } ORDER BY (?**x**) LIMIT ***L*** OFFSET ***O***
Max expression	max(λ**x.T**, ***O***, ***L***)	SELECT DISTINCT ?**x** WHERE { ϕ(**T**) } ORDER BY DESC(?**x**) LIMIT ***L*** OFFSET ***O***
KB Predicate	**<pred_iri>**(i, **s**/**<s_iri>**, **o**/**<o_iri>**)	?**s**/**<s_iri>** **<pred_iri>** ?**o**/**<o_iri>**.
Interval predicate for reified facts	wdt:**PID**(i, **s**/**<s_iri>**, **o**/**<o_iri>**) $\wedge$ interval(**e**, i)	?**s**/**<s_iri>** p:**PID** ?x. ?x ps:**PID** ?**o**/**<o_iri>**. ?x *pq:P580* ?$\mathbf{e}_{start}$. ?x *pq:P582* ?$\mathbf{e}_{end}$.
Interval predicate for non-reified facts	wdt:**PID**(i, **s**/**<s_iri>**, **x**) $\wedge$ interval(**e**, i)	?**s**/**<s_iri>** wdt:**PID** ?**x**. BIND (?**x** AS ?$\mathbf{e}_{start}$) BIND (?**x** AS ?$\mathbf{e}_{end}$)
Now predicate	now(**e**)	BIND (now() AS ?$\mathbf{e}_{start}$) BIND (now() AS ?$\mathbf{e}_{end}$)
Overlap predicate	overlap(**e1, e2**)	FILTER(?$\mathbf{e1}_{start}$ <= ?$\mathbf{e2}_{end}$ && ?$\mathbf{e2}_{start}$ <= ?$\mathbf{e1}_{end}$)
Before predicate	before(**e1, e2**)	FILTER(?$\mathbf{e1}_{end}$ <= ?$\mathbf{e2}_{start}$)
After predicate	after(**e1, e2**)	FILTER(?$\mathbf{e1}_{start}$ >= ?$\mathbf{e2}_{end}$)

Table 12: KB-Specific λ-expression to SPARQL Translation Rules (full rules)

ios while translating to SPARQL. Next set of templates like *min/max/argmin/argmax/count* provide the base numerical reasoning capabilities. These constructs can be clubbed with other types of reasoning. For example, temporal before/after utilize argmax/argmin on date attribute to get the desired effect of sorting and picking the desired entity.

λ-expression contains one or more terms such that each term T_i is comprised of one or more predicates connected via $\wedge$ or $\vee$. Translation of different predicates is also present in Table 12. Predicates used in λ-expression can be broadly categorized into *KB predicate*, *interval* predicate, and *temporal* predicate. *KB predicate* is directly mapped to triple pattern in SPARQL, whereas the *interval* predicates create an interval consisting of start time and end time for a given event. Table 12 has Wikidata specific rules which can be used for constructing intervals. *start time(P580)*, *end time(P582)* or *point in time(P585)*, in the reified events case, are used for creating the interval. For non-reified events other temporal properties connected with the entities are used for getting the interval. *Teenager* predicate and *now* make use of *date of birth(P569)* and *current time(now())* respectively for creating the interval. Each of the Wikidata specific rules can be mapped to target KB accordingly. Temporal predicates include *overlap*, *before*, and *after* which make use of SPARQL FILTER condition to filter out the intervals that do not fall under given conditions.