Schema Aware Semantic Reasoning for Interpreting Natural Language Queries in Enterprise Settings

Jaydeep Sen
IBM Research AI, India
jaydesen@in.ibm.com

Tanaya Babtiwale
NMIMS, India
tanbab98@gmail.com

Kanishk Saxena
SMVIT, India
kanishksaxena5@gmail.com

Yash Butala
IIT Kharagpur, India
yashbutala@iitkgp.ac.in

Sumit Bhatia
IBM Research AI, India
sumitbhatia@in.ibm.com

Karthik Sankaranarayanan
IBM Research AI, India
kartsank@in.ibm.com

Abstract

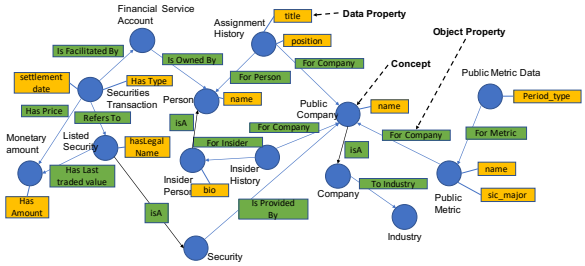
Natural Language Query interfaces allow the end-users to access the desired information without the need to know any specialized query language, data storage, or schema details. Even with the recent advances in NLP research space, the state-of-the-art QA systems fall short of understanding implicit intents of real-world Business Intelligence (BI) queries in enterprise systems as Natural Language Understanding remains an AI-hard problem. We posit that deploying ontology reasoning over domain semantics can help in achieving better natural language understanding for QA systems. In this paper, we specifically focus on building a Schema Aware Semantic Reasoning Framework that translates natural language interpretation as a sequence of solvable tasks by an ontology reasoner. We apply our framework on top of an ontology-based, state-of-the-art natural language question-answering system ATHENA, and experiment with 4 benchmarks focused on BI queries. Our experimental numbers empirically show that the Schema Aware Semantic Reasoning indeed helps in achieving significantly better results for handling BI queries with an average accuracy improvement of 30%

1 Introduction

Natural Language Query (NLQ) interfaces for information access (e.g., Alexa, Google Assistant, and Siri) have gained popularity in recent times as they allow the end-users to specify their information needs in natural language. In enterprise settings, systems with NLQ interfaces (Li and Jagadish, 2014; Saha et al., 2016; Sen et al., 2020) are especially desirable as they enable the business analysts to access domain-specific knowledge without having to learn specialized query languages such as SQL, SPARQL, or other customized query languages. NLQ interfaces also allow the end-user to explore and analyze the data without the need to learn about the storage mechanism and schema-specific details of the underlying system. Thus such systems can help boost analyst productivity and minimize training costs and time.

Current State of NLQ Systems: To retrieve required results from the underlying database, NLQ based systems need to translate the input natural language query to a structured query (e.g., SQL, SPARQL) that is then used to query the database. Recent advances in NLP research has produced many machine learning based systems which try to model the QA pipeline as a sequence-to-sequence architecture (Zhong et al., 2017a; Xu et al., 2017; Guo et al., 2019). However, they focus on rather simple queries on a single table or on a relatively simple schema with a few tables. For example, RAT-SQL (Wang et al., 2020) is a state-of-the-art on Spider (Yu et al., 2018) dataset, where Spider dev set has an average of only 4 tables per Database and less complex queries than typical BI queries seen with complex nesting (Sen et al., 2020). Therefore, such systems in their current capacity, are not yet aimed towards handling complex BI queries over large databases. An alternate approach is to utilize domain-specific ontologies and knowledge to translate the input natural language query to an intermediate meaning representation like OQL (Ontology Query Language) (Saha et al., 2016; Sen et al., 2020) or AMR (abstract meaning representation) (Banarescu et al., 2013) that can then be converted to the structured query (Saha et al., 2016; Li and Jagadish, 2014). However, these systems work well only if the user *explicitly* specifies the information need in the query. Such systems fail to capture and interpret the *implicit* intents implied in the query.

Proceedings of the 28th International Conference on Computational Linguistics, pages 1334–1345
Barcelona, Spain (Online), December 8-13, 2020



NL Query	Language Processor Output	Intended Interpretation
Which executives have more salary than Warren Buffet?	SELECT: executive WHERE: executive.salary > Warren Buffet.Salary	SELECT: executive WHERE: executive.salary > Warren Buffet.Salary
Which executives have more trading accounts than Warren Buffet?	SELECT: executive WHERE: executive.account > Warren Buffet.account	SELECT: executive WHERE: COUNT(executive.account) > COUNT(Warren Buffet.account)
Which executives have traded more stocks than Warren Buffet?	SELECT: executive WHERE: executive.TradedStocks > Warren Buffet.TradedStocks	SELECT: executive WHERE: SUM(executive.Transaction.StockCount) > SUM(Warren Buffet.Transaction.StockCount)

Figure 1: (a) Ontology Snapshot (b) Motivating Examples

Improving Query Interpretation by Reasoning: We posit that *ontology reasoning can serve as a useful tool for interpreting and inferring the implicit intents in natural language queries*. To understand the intuition, let us consider a financial domain analyst interested in obtaining a list of traders who traded more stocks as compared to a well-known trader X. This can be expressed by the two equivalent natural language queries:

Q1: List the traders buying stocks of total value more than the total value of stock bought by trader X.
Q2: List the traders who traded more stocks than trader X.

Here, note that **Q1** explicitly specifies that the system needs to return the list of traders such that the *value* of stocks traded by them is more than the *value of stocks* traded by trader X. On the other hand, this requirement is not made *explicit* in **Q2**, although to a person working in the financial domain, this *implicit* information is obvious, and in fact, **Q2** represents the commonly used language of the domain. While state-of-the-art NLQ systems can handle queries like **Q1** where the information required for computation is explicitly present in the query, they fail to answer queries like **Q2** where the user intent is implicit.

This is where we propose to apply a schema-aware semantic reasoning framework, which can reason that *more than* operation applies to numeric values and thus, can not be applied directly to entities of type *stock*. It can then use the domain knowledge to identify that *stocks* are associated with the sale and purchase transactions having monetary (numerical) values on which *more than* can be applied.

Our Focus: Admitting that the fully automated interpretation of natural language queries is AI-hard (Yampolskiy, 2013), we focus on a specific aspect of natural language query interpretation in enterprise settings where ontology reasoning can prove beneficial.

Capturing domain semantics: Consider the example questions in Figure 1 that are typical of queries a financial analyst may have. Note that the three questions seem very similar in their linguistic structure and use the same tokens except for salary (Q1), trading accounts (Q2), and stocks (Q3). From the perspective of natural language phrasing, all the queries look very similar in structure and hence will produce very similar outputs for any language processing operation. However, due to different semantics associated with the argument of `more than` operation, the system needs to perform very different operations to fetch the desired results for the user (intended interpretation column in Figure 1) For Q1, *more than* being a numeric comparison can be applied on `Salary`, which is a numeric field. However, for Q2, `trading accounts` is a concept (i.e., a collection of entities) and not a numeric field. The intended operation here is the `count` of the number of trading accounts. Likewise, in Q3, `more than` in the context of stocks correspond to not the count of individual stock symbols but the sum of all stock units transacted per transaction. Therefore, to interpret the query correctly, the system needs to utilize the domain knowledge to infer the implied operations requested in the query.

Our Contributions: We propose a framework for schema-aware semantic reasoning for natural language query interpretation. To capture implicit intents in the query, we construct a generic ontology for semantic reasoning that can be applied across different domains as a guide to deduce semantically valid meaning representations of natural language queries. We also propose novel algorithms (Section 3) to combine linguistic analysis of a query with domain-specific facts to translate query interpretation as a sequence of solvable tasks by an ontology reasoner such as consistency checking, instance retrievals, etc. over the domain ontology augmented with domain-specific logical facts. We implement the proposed approach on top of a state-of-the-art ontology-based NLQ interface ATHENA (Saha et al., 2016) and evaluate on multiple benchmark data sets (Section 4). We achieve 30% improvements in interpretation accuracy over ATHENA.

2 Related Work

Our work lies at the intersection of natural language query answering systems and applications of ontology reasoning. Therefore, we cover both the segments to study the recent research trends in them.

Natural language question answering(NL QA). NL QA in general refers to a large body of works segmented by the backend data store and resources used to answer the question. Natural language interfaces over linked data (Unger et al., 2014; Usbeck et al., 2015), RDF stores (Zou et al., 2014) are primarily meant for supporting question answering on text data. This is still an active area of research with many running challenges (QAL, 2016). On the other hand, Natural Language Interfaces to databases (NLIDB) are meant to support the NL question answering over structured data like databases. In recent days NLIDB systems have gained immense popularity and significant research exposure. NLIDB systems have mostly seen rule-based solutions (Popescu et al., 2003; Li and Jagadish, 2014; Saha et al., 2016) using database schema as one the primary resource. Although machine learning based systems (Zhong et al., 2017a; Utama et al., 2018) have been proposed recently, they often suffer from a lack of domain-specific training data to learn complex query classes and join paths among multiple tables. With the limited training data available, Seq2SQL (Zhong et al., 2017a) can handle only single table select and project queries without joins and therefore, is easily outperformed by state-of-the-art rule-based systems (Li and Jagadish, 2014; Saha et al., 2016). ATHENA (Saha et al., 2016) is one of the state-of-the-art NLIDB systems which outperformed its predecessor NALIR (Li and Jagadish, 2014) by introducing the use of domain ontology to resolve ambiguities in query interpretation.

Ontology Reasoning. Ontology reasoning has been widely used in data stores with an open-world assumption, where all facts may not be explicitly stated. Such data stores can broadly be called Knowledge Bases (KBs), where using ontology axioms over the existing facts can result in inferring lot more facts for the domain. This is known as Ontology-Based Data Access (OBDA) (Ortiz, 2013). OBDA systems and ontology reasoning have been extensively used across different tasks and domains (Ortiz, 2013). There are several available reasoners (Sattler and Matentzoglu, 2014) with varying support of constructs and expressivity in axioms along with a trade-off on time-bound efficiency. The most important task a reasoner does is to support question answering over an ontology by taking into account the implicit facts that can be inferred with the axioms. In question answering space, there has been some work on ontology reasoning for ambiguity resolution in dialogues (Estival et al., 2004) or combining answers from multiple knowledge bases by interpreting explicit operators like existential qualifiers (Waldinger et al., 2018). Further, ontology reasoning has also found successful applications in different aspects of natural language understanding such as modeling common sense reasoning via ontologies and subsequent reasoning tasks (Rashkin et al., 2018; Sap et al., 2018).

We build upon these two bodies of work and utilize the advances made in the field of ontology reasoning to address a specific challenge in natural language-based information access systems – interpreting implicit intents in the query by reasoning over domain knowledge.

3 System

In this section, we detail the proposed reasoning framework for natural language query interpretation. We start with the overall system architecture and then discuss each of the components in detail.

The proposed framework, as illustrated in Figure 2, can be divided into (i) an offline part and (ii) an online part. The offline part consists of components that are curated one time for a target domain and are used as resources later to facilitate reasoning for query interpretation. The online part consists of components that work on each input natural language query and help interpret the query in the context of the target domain by using the offline components for domain reasoning.

3.1 System Components

3.1.1 Domain Ontology

Domain Ontology is used to represent the target domain on which natural language queries are to be interpreted. The domain ontology captures target domain semantics in terms of different concepts in the domain, their attributes as data properties, and interactions with other concepts as object properties. Figure 1 captures a partial snapshot of the Finance domain ontology with

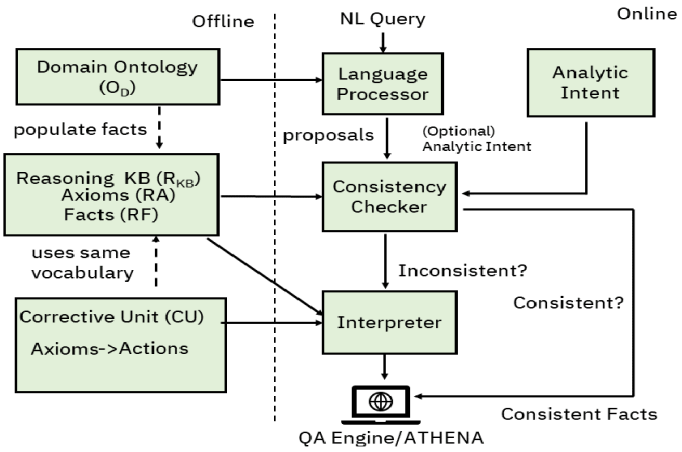


Figure 2: System Architecture

*CONCEPT*s like `SecuritiesTransaction` having *PROPERTY* such as `hasLegalName` or specifically *TIME_PROPERTY* like `hasSettlementDate`, and *RELATION* such as `hasLastTradedValue` with `MonetaryAmount`.

3.1.2 Reasoning Knowledgebase

We design *Reasoning Knowledgebase* (R_{KB}) to model the generic rules for making semantically valid interpretations, e.g. numeric aggregation (average, sum, etc.) can only be applied on a numeric property. Figure 3 shows a tiny fraction of R_{KB}, with three components

• **Concepts:** R_{KB} has concepts like $CONCEPTS, PROPERTIES$, $NUMERIC\_PROPERTIES$, $NUMERIC\_COMPARISON, NUMERIC\_AGGREGATION$, etc. which are generic entities to write rules for semantically valid interpretation.

• **TBox Axioms:** The TBox axioms in R_{KB} are used to describe the different types of relations between concepts and more importantly it enforces domain agnostic semantic constraints which are universally true for any valid interpretation. For example, the axiom $NUMERIC\_COMPARISON \subset MEASURE$ enforces that any numeric comparison has to be an instance of $MEASURE$. Another axiom $MEASURE = NUMERIC\_AGGREGATION \cup NUMERIC\_PROPERTY$ defines measure to be a union class of numeric aggregation and numeric properties. In effect, both these axioms enforce that any numeric comparison like more than, less than, etc. can be applied only on a numeric property or an aggregation. It also contains axioms about query-specific annotations to enforce (and later inferred via correction) intent completeness of an input query. For example, a query asking to retrieve top entities must also have an order to rank the entities.

• **ABox Facts:** Given a domain ontology, we collect the domain-specific facts which are useful for reasoning with R_{KB} axioms. For example, to work in the Finance domain, we need to know that *Executive* is not a $NUMERIC\_PROPERTY$ but a $CONCEPT$, hence $MORE\_THAN$ can not be applied directly to *Executive*. *ABox* facts are such domain-specific facts that we derive from a domain ontology (in Algorithm 1) and populate to R_{KB} for subsequent reasoning and inference (in Algorithm 2).

Concepts	CONCEPT, PROPERTY, NUMERIC_PROPERTY, MEASURE_PROPERTY, NUM_COMPARISON, MORE_THAN, LESS_THAN, TOP
TBox Axioms	hasKeyProp(CONCEPT, PROPERTY), hasMeasureProp(CONCEPT,PROPERTY), *MEASURE_PROPERTY, NUMERIC_PROPERTY* ⊆ *NUMERIC_PROPERTY* , {MORE_THAN, LESS_THAN} ⊆ NUMERIC_COMPARISON MEASURE := NUM_AGGREGATION U NUMERIC_PROPERTY CONCEPT ∩ PROPERTY = Φ SELECT ⊆ PROPERTY, NUMERIC_COMPARISON ⊆ MEASURE, NUM_AGGR ⊆ NUMERIC_PROPERTY, Query.∃hasTop ⊆ Query.∃hasMeasure
ABox Facts	CONCEPT(Executive), CONCEPT(SecuritiesTransaction), PROPERTY(stockCount), MEASURE_PROPERTY(stockCount), hasKeyProp(Executive, Name), hasMeasureProp(SecuritiesTransaction,stockCount)

Correction Detection Axioms	Corrective Actions
SELECT(C) ∩ CONCEPT(C) ∩ hasKEYPROPERTY(C,P)	+ SELECT(P), - SELECT(C)
NUM_COMPARISON(P) ∩ ¬NUMERIC_PROPERTY(P) ∩ hasCONCEPT(P,C)	+NUM_COMPARISON(C), -NUM_COMPARISON(P)
NUM_COMPARISON(C) ∩ CONCEPT(C) ∩ hasMeasureProp(C,M)	+NUM_AGGREGATION(SUM_m), +onProperty(SUM_m,M), +NUM_COMPARISON(SUM_m) , - NUM_COMPARISON(C)
NUM_COMPARISON(C) ∩ CONCEPT(C)	+COUNT(M), +NUM_COMPARISON(M) -NUM_COMPARISON(C)
Query.∃hasTop ∩ Query.∄hasMeasure ∩ TOP(C) ∩ CONCEPT(C) ∩ hasMeasureProp(C,M)	+MEASURE(M), +hasMEASURE(Query,M)

Figure 3: (a) Logical Snapshot of Reasoning KB (b) Logical Snapshot of Correction Unit

3.1.3 Correction Unit

While Reasoning Knowledgebase R_{KB} is modeled to catch semantically invalid interpretations, we design *Correction Unit* intending to detect the cause of semantic invalidity which can be corrected by inferring the implicit intent through action axioms. Therefore, the *Correction Unit* has axioms divided into two sets (1) Detection Axioms to detect the specific causes of invalid interpretations and (2) For every detection axiom, we have corresponding action axioms configured to take a set of actions to make the interpretation semantically valid.

As shown in Figure 3(Row 3), the detection axiom tries to detect if there is a *NUMERIC_COMPARISON* applied on a *CONCEPT C*, where the concept already has a configured *MEASURE_PROPERTY M* in the domain ontology. Note that, *NUMERIC_COMPARISON* on a *CONCEPT* is not semantically valid, thus the corresponding action axiom corrects the *NUMERIC_COMPARISON* to apply instead on an aggregation which is the SUM on *M*.

3.2 Query Flow

Although Figure 3 shows only a tiny fraction of R_{KB} and *Correction Unit* axioms, the key point of using axioms to model the semantic validity and subsequently its detection and correction is to translate the hard task of implicit intent understanding and producing semantically valid interpretations, into traditional reasoner tasks such as consistency check, satisfiability, and instance retrieval. We demonstrate that in Figure 4 via a running example which is the same query that we presented in motivation Section 1. We describe each of the online components to show how the offline components are used with an ontology reasoner for natural language query interpretation. Figure 4 reuses the example natural language query we presented during motivation in Section 1.

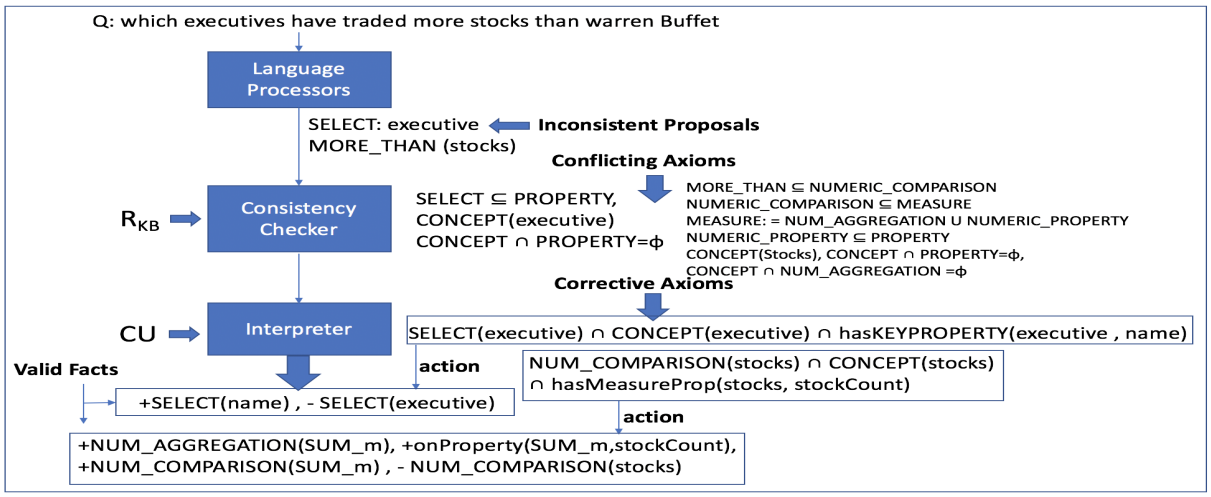


Figure 4: Illustrative Query Interpretation Flow

• **Language Processors** Language processors scan through the query applying general natural language processing tools and produce some logical annotations for different query clauses like Select, Where, Comparisons, Group By, etc. with associated tokens from the query. It is a standard layer in every question answering system. We chose to use the same language processor layer from the state-of-the-art ontology-based question answering system ATHENA (Saha et al., 2016). The system can choose to borrow annotators from other implementations as well. Because language processor annotations may or may not be correct for the target domain, we call them *proposals*. Figure 4 shows the *Proposals* as obtained from language processors applied to the example question.

• **Consistency Checker:** This acts as the first layer of the proposed system. Given a set of proposals, an ontology reasoner is used to check the *consistency* of the proposals for the axioms modeled in R_{KB}. Reasoner produces a binary output to signal if the proposals are consistent with R_{KB} or not.

• **Interpreter:** Given a set of proposals that can not produce a semantically valid interpretation to pass the *Consistency Checker*, the *Interpreter* makes use of *Correction Unit* axioms to detect and correct the

proposals. It uses an ontology reasoner to check if any of the detection axioms is *satisfiable*. If so, it uses the reasoner to *retrieve instances* of the class expression captured in the corresponding *action axiom* and performs the designated sequence of actions on those instances to convert the inconsistent proposals into consistent facts that can now produce semantically valid interpretations. Figure 4 shows the relevant axioms from the *Correction Unit* in detecting inconsistent proposals, as well as the corresponding corrective actions to convert them to valid facts over R_{KB}.

3.3 Algorithms

Having described individual components of our reasoning framework and illustrated their usage with examples, we now present the overview of the formal algorithms such that it can easily be implemented using core reasoning libraries.

Algorithm 1: Algorithm to initialize Reasoning Resources to be used for query interpretation

```
   Input: Domain Ontology O_D
   Output: Reasoning KB R_KB,
1  OWLOntology R_KB ←
     loadReasoningOntology(reasoning.owl);
2  foreach Concept c ∈ O_D do
3      R_KB.addAxiom(CONCEPT(c));
4      foreach Property p ∈ O_D do
5          R_KB.addAxiom(PROPERTY(p));
6  foreach Relation p ∈ O_D do
7      R_KB.addAxiom(RELATION(p));
8  foreach AnnotationAxioma ∈ O_D do
9      Annotation annot ← a.getAnnotation();
10     Element el ← a.getOntologyElement();
11     R_KB.addAxiom(annot(el)));
12 Return R_KB
```

Algorithm 2: Algorithm to detect and correct inconsistent proposals to produce valid facts

```
   Input: NL Query Q, Reasoning KB R_KB,
   Output: Set[Facts] interpretedProposals
1  Set[Fact] proposalSet = LanguageProcessor(Q);
2  R_KB.addFacts(proposalSet);
3  if Reasoner.isConsistent(R_KB) then
4      return proposals;
5  Set[Axioms] validProposals = {};
6  OWLOntology O_C ← loadCorrectionOntology(correctionAxioms.owl);
7  R_KB.addAxioms(O_C.getAxioms());
8  while (!Reasoner.isConsistent(R_KB)) do
9      foreach Axiom tbox ∈ O_C.getTBoxAxioms() do
10         foreach ClassExpression ce ∈ tbox.nestedClassExpression do
11             Set[Instance] correctionInstance =
                 Reasoner.getInstances(ce,R_KB)
12             if ce.isCorrectionDetectionAxiom() & correctionInstance ≠
                 empty then
13                 R_KB.removeAxioms(actionClass(correctionInstance))
14             if ce.isCorrectionActionAxiom() & correctionInstance ≠
                 empty then
15                 R_KB.addAxioms(correctedClass(correctionInstance))
16                 validProposals.addAll(correctedClass(correctionInstance))
17 return validProposals;
```

Algorithm 1 is built to set up the necessary offline resources to utilize ontology reasoning for query interpretation. Algorithm 2 further utilizes known reasoner tasks such as consistency checking for producing semantically valid facts for interpretation for each natural language query.

Algorithm 1 iterates over the different ontology resources like concepts, properties, and relations from the target domain ontology O_D to create domain-specific facts on R_{KB} vocabulary to ingest into R_{KB}(Line 3-Line 11). Note that, in Line 8, Algorithm 1 also reads various annotations present in O_D to ingest corresponding ABox facts into R_{KB}, such as annotating key properties or default measures for concepts, etc. as seen in Figure 3 ABox facts. These are typically provided by SMEs (Subject Matter Expert) to capture finer aspects of the domain. Facts like *hasKeyProp*, *hasMeasureProp*, etc. are produced for these SME provided annotations.

Algorithm 2 outlines the sequence of tasks performed to reason over a natural language query accurately as outlined in Section 3.1. For proposals obtained from a query Q that are inconsistent with R_{KB}, Correction ontology O_C is iterated over to detect the Detection axiom designated to the cause of inconsistency. The corresponding Action axiom is determined using conceptual linking employing a shared naming convention and index. The detected conflicting class expression is deleted, provided an action is found for the same. The Action axiom further inserts the corrected class into R_{KB}. The algorithm continues to search and correct inconsistent proposals until R_{KB} becomes consistent. Therefore, if a Query q requires N inferences, the algorithm complexity will be $O(N * P(|R_{KB}|))$ where $P(|R_{KB}|)$ represents polynomial time complexity of reasoner tasks on knowledgebase size $|R_{KB}|$. From our experiments, we know the number of inferences needed for a single query is typically upper bounded by 6, therefore, Algorithm 2 is effectively a polynomial-time algorithm on knowledge-base size $|R_{KB}|$.

Ontology	$\|C\|$	$\|P\|$	$\|R_F\|$	$\#R_{KB}\ facts$	$\#Query$	$\#Q_{Reasoning}$
FIBEN	152	664	159	866	144	109
GOSALES	25	156	31	230	112	92
QALD	214	2279	161	2868	50	28
Spider	4.1	25.9	3.3	37.4	1034	161

Table 1: Ontology statistics.

4 Experiments

In this section, we present a comprehensive evaluation of our proposed system. We describe in detail the experimental setup, benchmarks on which our prototype has been tested, followed by the set of evaluation metrics we have used, and finally the results with in-depth analysis.

As described in Section 3, we used our reasoning approach on top of existing state-of-the-art ATHENA. We have reused the existing language processors from ATHENA, whereas the algorithms with reasoning components e.g. Consistency Checker, Interpreter, etc. are implemented using JFact Reasoner (Palmisano, 2015) to create our prototype *ReasonedATHENA*. The modular architecture of our reasoning framework allows it to be built on top of any other state-of-the-art QA systems as well which rely on an intermediate level of annotations as ATHENA. This allows our proposed framework to be adaptable to different state-of-the-art implementations too.

4.1 Domains

To test the effectiveness of *ReasonedATHENA* we need benchmarks with BI queries. However, there is no specific benchmark already known to the community that specifically focuses on the challenges of BI queries. Therefore, we take 2 closest existing benchmarks in (1) Spider and (2) QALD6: QA on statistical modelling data. Apart from these, we create our own BI query benchmarks for 2 widely used domains in enterprise applications for business analytics as (3) Finance and (4) Sales. We describe each of these domains below:

• **Spider:** (Yu et al., 2018) is a widely used benchmark for NLIDB systems, and it is the dataset that comes closest to providing multi-table dependent question answering queries with an average of 4.1 tables in a domain. Unlike the relatively straightforward queries in WikiSQL (Zhong et al., 2017b), Spider provides a better degree of complexity involving aggregations, negations, numeric comparisons, etc. which are closer towards analytic queries seen in BI applications.

• **QALD6-Statistical question answering** (QAL, 2016): This task specifically focused on understanding implicit intents for answering statistical intent queries, which are also very close to analytic intent queries in BI applications.

• **Finance Benchmark Ontology: FIBEN** is a complex business intelligence benchmark ontology on Finance, first proposed in (Sen et al., 2019). It combines two standard ontologies in Finance, *(i)* FRO and *(ii)* FIBO. FIBEN models information about various financial metrics of companies, insiders working in companies, their stock transactions and holding accounts, etc., thus emulating a typical real-world data mart in Finance.

• **Sales Benchmark Data model: GOSALES (gos, 2020)** is a standard data model that is used as a benchmark in IBM's Cognos Business Intelligence Offerings (Browne and others, 2010). GOSALES contains information about various measures like sales target, product forecast across various dimensions like location, time, etc. The combinations of various measures and dimensions make it suitable for BI use cases.

Table 1 provides basic statistics for all the domain ontologies. In addition to the number of concepts and properties, Table 1 also shows the number of domain-specific facts added to the reasoning knowledge base prior to query interpretation by Algorithm 1.

4.2 Workload

We re-used the existing benchmark for QALD6 and dev set of queries for Spider. For FIBEN and GOSALES, we aimed to create a workload specifically focused on BI queries. We asked a group of 6 users including 3 domain SMEs (subject matter expert) and 3 business analysts to ask different types of

NL queries on these domains. These users were aware of the domains, their underlying data. Moreover, their job role made them experienced with typical day-to-day BI information needs on these domains. They were encouraged to try all variants of queries starting from simple and explicit queries to complex business intelligence queries (in natural language) needing implicit intent understanding and reasoning.

Table 1 also lists the number of queries we experimented with for each domain and how many of these queries needed reasoning. It shows that for FIBEN and GOSALES a large portion of queries (>80%) effectively need reasoning for semantic validation or/and inference of complete intents. Thus the query workloads becomes a suitable benchmark to address challenges in BI query interpretation. The ontologies and the query benchmarks are made public to promote further reuse[1].

4.3 Experimental Setup and Evaluation Metrics

4.3.1 Baselines:

We use two state-of-the-art baselines, one from rule-based solution paradigms and another from machine learning based approaches. They are as follows:

(Rule-Based) ATHENA: ATHENA becomes a natural baseline to compare the performance boost obtained by applying our reasoning framework on top of it. Also, ATHENA being a state-of-the-art NLIDB system, comparing our performance with ATHENA subsumes comparison with other rule-based NLIDB systems such as NALIR (Li and Jagadish, 2014).

(ML Based) IRNet: We also take another NLIDB system IRNet (Guo et al., 2019) which adopts a neural machine translation-based approach to be one of the state-of-the-art systems on Spider.

4.3.2 Metrics:

We use the following measures to evaluate the gains achieved by better query interpretation:

Accuracy: # of questions producing correct answers / # of questions asked to the system.

While *accuracy* is measured for the complete set of questions, we use *precision* and *recall* to measure the performance of domain reasoning specifically concerning the queries which need reasoning. We define *precision* and *recall* for domain reasoning in question answering as follows:

Precision: # of NL queries producing correct intent with domain reasoning / # of NL Queries which had some inferences made by domain reasoning

Recall: # of NL queries which produced correct intent with domain reasoning / # of NL Queries that actually needed domain reasoning [2]

Conceptually, *precision* measures the utility of the domain reasoning framework in doing correct inferences needed for accurate query interpretation, while *recall* measures how accurately the domain reasoning framework can detect the need for reasoning and respond with correct inferences.

Along with the traditional metrics of the QA community, we also want to measure the performance of domain reasoning module individually as an independent module in terms of soundness and completeness which we define as follows:

Soundness: # of correct inferences produced by domain reasoning / # inferences produced by domain reasoning

Completeness: # of correct inferences produced by domain reasoning / # of inferences needed to be produced by domain reasoning

4.4 Results

Table 2 captures the accuracy numbers of *IRNet*, *ATHENA*, and *ReasonedATHENA*. As seen in Table 2, domain reasoning indeed helps *ReasonedATHENA* to gain a significant leap in accuracy of almost 50% for GOSALES and FIBEN which are specifically concentrated on BI queries. This can be correlated with Table 1 findings where more than 80% of queries required domain reasoning. *ReasonedATHENA* achieves higher accuracy for Spider and QALD as well, although the difference in QALD is around 12%, as these benchmarks have a mix of non-BI queries which may not require reasoning.

[1] https://github.com/jdpsen/ReasoningForNLQ

[2] Queries needing reasoning are manually annotated after seeing the gold standard queries.

Ontology	IRNet	ATHENA	ReasonedATHENA
FIBEN	36.2	45.13	88.89
GOSALES	30	46.42	91.96
QALD	41.2	72	84
Spider	52.8	54.98	67.02

Table 2: Overall Accuracy percentage.

Table 3: Precision and Recall

Ontology	Precision	Recall
FIBEN	88.03	86.55
GOSALES	97.39	94.91
QALD	92.59	89.28
Spider	92.59	87.63

Table 4: Soundness and Completion

Ontology	Soundness	Completeness
FIBEN	97.31	95.90
GOSALES	98.54	93.11
QALD	94.11	91.42
Spider	95.41	90.11

IRNet adopts a grammar-based neural model that helps it to answer complex and cross-domain queries but due to its very limited reasoning capabilities, it fails to infer implicit intent to answer BI queries. ATHENA being an ontology-aware system can utilize SME populated configurations to handle some very basic levels of reasoning like choosing a configured key property for a concept in a select clause or doing aggregation for a configured measure property and etc. but *ATHENA* fails to extend it to more involved generic cases of reasoning modeled in R_{KB}, which are often required in analytic queries.

Table 3 shows the precision and recall numbers achieved by *ReasonedATHENA* for NL query interpretation. A precision value of $88-97\%$ can be considered reasonably high and signifies that most of the inferences done by domain reasoning framework produce the correct interpretation. Also, a high recall value of $86-94\%$ further indicates that the domain reasoning framework is also quite sensitive to detect any need for domain reasoning for producing correct interpretation and doing correct inferences whenever needed. A high value of precision and recall together gives the confidence that the domain reasoning framework can be used to infer semantically valid and correct intents without introducing noise in terms of erroneous inferences.

Table 4 shows the soundness and completeness values of the proposed domain reasoning framework in *ReasonedATHENA* as an independent module. These values show a good correlation with Table 3 numbers, which is expected given that accurate interpretation needs correct inferences. Note that, soundness and completeness numbers are higher than precision and recall numbers respectively. This too can be explained by the fact that a question most often needs multiple inferences to be done correctly in order to get to the right interpretation. Therefore, a single incorrect inference can make the complete query interpretation counted as wrong and thus, has a larger influence on the ratio of precision and recall numbers than soundness and completeness numbers.

We also analyzed the number of inferences needed for answering a query. Among the queries which needed some form of reasoning, Table 5 shows the distribution of such queries depending on the number of inferences needed to answer each query. For example, in *GOSALES* 61 queries (66%) need $2-3$ inferences as compared to 18 needing only 1 and 13 needing more than 3. The trend is similar for other benchmarks too except FIBEN which being the largest ontology requires a larger number of queries with more than 3 inferences. In general, the distribution of Table 5 shows a good percentage of queries require multiple inferences, although more than 3 inferences at a time are not so common. Table 6 captures a time analysis for all the domains. For each domain, it computes the average number of inferences needed to answer queries in that domain and also the average time taken to complete the inferences. Time taken to complete inferences is correlated with the number of inferences and also the domain complexity i.e. the size of facts populated in the domain. In correlation with Table 1, *FIBEN* being the most complex domain has the highest average number of inferences needed and thus the highest average time needed as well. Even in *FIBEN*, the average time needed per query is around 4 seconds, which is quite affordable to allow real-time query interpretation.

Table 5: # queries by # Inferences needed

Ontology	=1	2-3	>3
FIBEN	16	44	50
GOSALES	18	61	13
QALD	23	5	0
Spider	41	108	12

Table 6: Average # of Inferences and Average Time

Ontology	Avg #Inferences	Avg Time(in sec)
FIBEN	3.81	4.34
GOSALES	1.83	2.09
QALD	1.25	1.86
Spider	1.78	2.12

4.5 Lessons, Key Takeaways, Future Work

Generalizability: We have modeled the reasoning knowledge base (R_{KB}) and Correction unit axioms in a domain agnostic way and used target domain ontology only as a plug-in, so as to provide generalizability to any application domain. These axioms are a flexible artifact which can also be extended further to model more involved reasoning needs in QA across different use-cases. Although we have applied our reasoning framework over ATHENA, the modular design ensures the same reasoning framework can easily be adapted and applied on top of any other state-of-the-art QA system to infer semantically valid implicit intents. Moreover, the inherent rule-based nature of the proposed framework ensures that the requirement of a large amount of training data is curbed. The performance improvements we see in our experiments also hints that reasoning over domain semantics might be sufficient for a QA system to provide accurate interpretation and subsequent results efficiently.

Error Analysis reveals an important trade-off that serves as an alert to *NOT* design highly generic axioms while handling inconsistencies, as it may lead to incorrect or redundant inferences. We found that the queries where reasoning framework produced erroneous results can be categorized as follows:

Precision Fails: For some queries, the reasoning framework inferred facts that were incorrect, resulting in a drop in precision. For example, in *FIBEN*, for any numeric comparison on "Holding" concept, reasoner infers the correct interpretation to be compared with the $SUM(holding.stockCount)$. This is correct for most of the queries like "who has more holdings of IBM stocks than Warren Buffet". However, consider another query "How many persons have more than 1 holding account with MSFT stocks" in this case the query is genuinely asking to count the number of accounts holding MSFT stocks and correct interpretation in comparison with $COUNT(Account)$.

Recall Fails: For some queries, although some reasoning was necessary to get to the correct interpretation, our current reasoning framework has not yet modeled such cases. For example, consider the FIBEN domain query "On Dec 2019, whose stock valuation was worth the most". "Stock valuation" in this query is to be interpreted as $\sum Count(Stock) * Current\_Value(Stock)$. However, this is equivalent to learning a new concept called "stock valuation" for the domain, which we do not support yet.

Future Work: The cases which we could not handle yet with our reasoning framework offer avenues of possible future work. Ideally, an ontology reasoning framework should be capable of doing all kinds of common sense reasoning, new concept learning, etc. i.e. what a human can do over a domain. Also, the reasoning framework is well suited to provide more explainable models for query interpretation. In the future, we plan to use this for solving explainability for QA systems.

5 Summary

We proposed a novel schema-aware semantic reasoning framework where we have used ontology reasoning as a tool to help natural language query interpretation, specifically to interpret complex BI queries across domains. Ontology reasoning, being a well studied problem and with a theoretically sound solution framework, provides us a sound foundation to develop a robust query interpretation framework. For a quantitative evaluation, we have applied our reasoning framework over a state-of-the-art ontology-based question answering system ATHENA and observed significant improvements in accuracy and coverage of query interpretation across 4 benchmarks. Deep domain understanding and domain reasoning are some of the hard challenges faced by QA systems today, limiting their applicability in enterprise settings. Our proposed reasoning framework can provide the necessary bridge to use QA systems for BI queries in enterprise settings.

References

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria, August. Association for Computational Linguistics.

Dean Browne et al. 2010. *IBM Cognos Business Intelligence V10. 1: Intelligence Unleashed*. IBM Redbooks.

Dominique Estival, Chris Nowak, and Andrew Zschorn. 2004. Towards ontology-based natural language processing. In *Proceeedings of the Workshop on NLP and XML (NLPXML-2004): RDF/RDFS and OWL in Language Technology*, NLPXML '04, pages 59–66, Stroudsburg, PA, USA. Association for Computational Linguistics.

2020. GOSALES Schema. `https://www.ibm.com/support/knowledgecenter/en/SS9UM9_9.1.2/com.ibm.sampledata.go.ref.doc/content/GOSALESDW-content.html`, June.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database with intermediate representation. In *Proceeding of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*. Association for Computational Linguistics.

Fei Li and H. V. Jagadish. 2014. Constructing an Interactive Natural Language Interface for Relational Databases. *Proc. VLDB Endow.*, 8(1):73–84.

M Ortiz. 2013. Ontology based query answering the story so far. *CEUR Workshop Proceedings*, 1087, 01.

I Palmisano. 2015. Jfact dl reasoner. `http://jfact.sourceforge.net/`.

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*, IUI '03, pages 149–157. ACM.

2016. Qald-challenge. `http://qald.aksw.org/`.

Hannah Rashkin, Maarten Sap, Emily Allaway, Noah A. Smith, and Yejin Choi. 2018. Event2mind: Commonsense inference on events, intents, and reactions. In *ACL*, pages 463–473, Melbourne, Australia.

Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, et al. 2016. Athena: an ontology-driven system for natural language querying over relational data stores. *Proceedings of the VLDB Endowment*, 9(12):1209–1220.

Maarten Sap, Ronan LeBras, Emily Allaway, Chandra Bhagavatula, Nicholas Lourie, Hannah Rashkin, Brendan Roof, Noah A. Smith, and Yejin Choi. 2018. ATOMIC: an atlas of machine commonsense for if-then reasoning. *CoRR*, abs/1811.00146.

Uli Sattler and Nicolas Matentzoglu. 2014. List of reasoners (owl. cs) http://owl. cs. manchester. ac. uk/tools/list-of-reasoners. *URL http://owl. cs. manch ester. ac. uk/tools/list-of-reasoners/. Modified*, 1(09).

Jaydeep Sen, Fatma Ozcan, Abdul Quamar, Greg Stager, Ashish R. Mittal, Manasa Jammi, Chuan Lei, Diptikalyan Saha, and Karthik Sankaranarayanan. 2019. Natural language querying of complex business intelligence queries. In Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 1997–2000. ACM.

Jaydeep Sen, Chuan Lei, Abdul Quamar, Fatma Özcan, Vasilis Efthymiou, Ayushi Dalmia, Greg Stager, Ashish Mittal, Diptikalyan Saha, and Karthik Sankaranarayanan. 2020. Athena++ natural language querying for complex nested sql queries. *Proceedings of the VLDB Endowment*, 13(12):2747–2759.

Christina Unger, André Freitas, and Philipp Cimiano, 2014. *An Introduction to Question Answering over Linked Data*, pages 100–140. Springer International Publishing, Cham.

Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Lorenz Bühmann, and Christina Unger. 2015. Hawk — hybrid question answering using linked data. In *Proceedings of the 12th European Semantic Web Conference on The Semantic Web. Latest Advances and New Domains - Volume 9088*.

Prasetya Utama, Nathaniel Weir, Fuat Basik, et al. 2018. An end-to-end neural natural language interface for databases. *arXiv preprint arXiv:1804.00401*.

Richard Waldinger, Cleo Condoravdi, and Asuman Suenbuel. 2018. Natural language access: When reasoning makes sense. *Advances in Cognitive Systems*, 6:17–29.

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

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning.

Roman V. Yampolskiy, 2013. *Turing Test as a Defining Feature of AI-Completeness*, pages 3–17. Springer Berlin Heidelberg, Berlin, Heidelberg.

Tao Yu, Rui Zhang, Kai Yang, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *EMNLP*, pages 3911–3921.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017a. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017b. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

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.