# Optimization of Large-Scale Knowledge Forward Reasoning Based on OWL 2 DL Ontology

Lingyun Cui[1], Tenglong Ren[1], Xiaowang Zhang[1,2(✉)], and Zhiyong Feng[1,2]

[1] College of Intelligence and Computing, Tianjin University, Tianjin 300350, China
{cly1213,tenglongren,xiaowangzhang,zyfeng}@tju.edu.cn
[2] Tianjin Key Laboratory of Cognitive Computing and Application, Tianjin, China

**Abstract.** This paper focuses on the performance of optimized forward reason systems. The main characteristics of forward reasoning are that it is sensitive to the update of data, has a high cost of precomputation closure, and can not be closely related to the characteristics of the specific query. Therefore, it usually makes reason on irrelevant data. Processing this data reduces the performance of the reason system and consumes a lot of memory resources. Backward reasoning can make up for this defect to a certain extent, but its inherent defect of the high cost of online query rewriting cannot make it efficient in reasoning tasks. We design an efficient reason method, which can effectively combine the advantages of forward reason and backward reason to ensure the completeness of reason as much as possible. It can not only reduce the processing cost caused by data updates and desensitize semantic data to a certain extent but also avoid the high cost caused by query rewriting and greatly reduce the cost of precomputation closure. Finally, we implement the proposed method on a prototype of a forward reason system named SUMA-F and compare it with the current forward reason systems with better performance on various datasets of different sizes. Experiments show that the SUMA-F has high reasoning efficiency, is better than other systems, and has high scalability on large-scale datasets.

**Keywords:** Forward reasoning · Ontology · RDF data

## 1 Introduction

The World Wide Web produces a vast amount of data that humans cannot process efficiently and computers cannot understand well. Tim Berners-Lee et al. [1] propose the concept of the Semantic Web, and the development of the Semantic Web makes knowledge expression modes such as knowledge graph and ontology widely used in the research field of query answering. The most critical feature in semantics is the relationship between concept attributes and concepts. Ontology takes the concept of interconnection through attributes as its core and provides a semantic framework for language understanding and generation.


H. Gao et al. (Eds.): CollaborateCom 2022, LNICST 461, pp. 380–399, 2022.
https://doi.org/10.1007/978-3-031-24386-8_21

This plays an important role in the semantic ontology system, which can represent events and objects through various complex semantic combinations, and its grammar is specially designed to express complex lexical meanings so that the ontology can use as few basic concepts as possible to construct descriptions of complex objects and procedures in a composite manner. Chen et al. [7] propose an unsupervised attribute network embedding framework to solve basic and compound relations in attribute networks. It considers the relationship between users and attributes, analyzes all first-order combinations to obtain composite relationships, and outperforms the current state-of-the-art baseline methods. RDFS was proposed because knowledge graphs cannot describe knowledge at the architectural level. RDFS defines inclusion relationships between concepts or roles, domain and value constraints for roles, and implements the classification of individuals. OWL with stronger expressive ability adds cardinality restriction, equivalent individuals, and other knowledge descriptions based on RDFS. In the query answering system, the implicit knowledge contained in the explicit knowledge can be obtained by ontology reasoning, which enriches the original data and returns more abundant results in the query.

There is a lot of research in the field of ontological reasoning, such as [3,6,8, 18,20,24], which are based on forward materialization. In the process of forward reasoning, the facts in the knowledge base are expanded according to the reason rules to obtain implicit knowledge. The original and newly reasoned data are used as the target data to repeat this reason process continuously. When no new data is generated, the reasoning process end. This whole process is called computing a forward closure. It is worth noting that the process of forward materialization does not change because of the query, and the query operation is simple and efficient. The disadvantage of forward reasoning is also obvious. The data heavily influence it, and it repeats the operation when the data is updated. The forward reason system is inefficient for frequent database updates.

Unlike forward reasoning, backward reasoning is query-driven and matches rules backwards. Backward reasoning is not sensitive to the update of data. It is calculated for specific queries and extends queries according to rules. Backward reasoning is less about data processing and more about searching rules and expanding queries. Ontop [5] is a pure backward reason system based on query rewriting. Although backward reasoning can be well applied in the context of frequent database updates, its operation is relatively complex, and the cost of calculating queries is high. In [19,22] for backward reasoning, advanced pruning optimization algorithm and dynamic optimization algorithm are proposed to reduce the cost of query computation in backward reasoning.

Because of the inherent defects of forward and backward reasoning, it becomes a challenge to design a method that strikes a balance between forward and backward reasoning, that is, between the high cost of computing a complete closure for all data and the huge cost of complex computations for each query. We propose an efficient forward materialization algorithm, which combines backward reasoning to optimize forward reasoning, reducing the knowledge of precomputation closure and the size of forward matching rules. We propose an efficient query parsing algorithm that maintains a key resource pool (KR-POOL) to hold

all the resource entities related to the corresponding query. It can not only do personalized reasoning according to a specific query but also break all the possible knowledge involved in a specific query into the form of one element into the KR-POOL. Compared with the traditional backward chaining algorithm, the algorithm reduces the time complexity of the query rewriting process by simplifying the operation of query processing. We implement all algorithms in the forward reasoning system and ensure completeness of reasoning.

Next, we introduce the three works we have made in this paper and introduce the specific implementation details of these three contributions in later chapters.

- We propose an efficient query parsing algorithm to maintain a key resource pool named KR-POOL, simplifying the complexity of query expansion and other specific query computation.
- We propose a forward reasoning algorithm to reduce the computational closure scale of forward materialization, which can materialize all the data related to the query and dynamically screen the rules to reduce the search scale of rules during reasoning. Optimizing the data and rules effectively reduces the reasoning time, and completeness is guaranteed.
- We implement our method in a forward reason system named SUMA-F and test the effectiveness and scalability of the system on the UOBM [13] dataset with the standard query of the UOBM dataset and achieved good results.

This paper reviews related work in Sect. 2. In Sect. 3, we present a preliminary definition of some of the basics involved in reasoning. We describe the algorithm principle in detail in Sect. 4. In Sect. 5, the architecture diagram of the forward reasoning system SUMA-F is presented. Experimental results are presented in Sect. 6, and conclusions are drawn in Sect. 7.

## 2 Related Work

With the development of the Semantic Web, related works in ontology reasoning can be divided into forward reasoning, backward reasoning, and hybrid methods based on forward materialization and query rewriting.

Jena [6] and Sesame [3] are relatively early systems that support RDF (Resource Description Framework) data reason, and the reasoning module is only a part of them. They are mainly used for small-scale data reasons on a single machine. Due to the design principle and hardware limitations, their scalability and computational power are poor, and they cannot process large-scale RDF data. Based on forward reason, Pellet's reasoning algorithm, adopted by Pellet [20] performs deductive reasoning on the original dataset offline according to the ontology, expresses the implicit ontology information as new knowledge obtained by explicit reasoning, and expands the original input dataset. In the online stage, query the expanded data set directly. However, the Tableau algorithm adopted by Pellet has high time and space complexity. Pellet is only suitable for processing small and medium-sized data. The main working principle of PAGOdA is to delegate the

heavy computational load to the datalog reasoner [14,16] and only use the hypertable algorithm [15] when necessary. Sequoia [8] is a consequence-based forward reasoning engine with support for nominals, implementing the calculus of logical $\mathcal{ALCHOIQ}^{+}$. Ontologies are classified by ontology preprocessing methods. SUMA [17,18] improves the n-step materialization model, restricts the reasoning to a finite number of steps, and proposes a partial materialization algorithm that can reliably and completely support root conjunctive queries as well as boolean queries with cyclic and fork-shaped structures.

Query rewriting techniques are also used in reason systems such as [5,9]. Instead of explicitly calculating all the implicit information according to the ontology, Ontop [5] rewrites the query according to the ontology and mapping, and the rewritten query explicitly contains the implicit information in the ontology. Since this query rewriting algorithm is performed online, it has a high time cost. Meanwhile, the mapping used in query rewriting requires human intervention input. In [4], the RDFS entailment rule set is divided into two subsets, and a query rewriting algorithm is proposed for query answering on knowledge graphs without reasoning. The method proposed in [10] can clearly describe the mapping relationship so that the database data can be better interpreted as ontology data.

There are also hybrid methods that combine the forward reason mechanism with the backward reason mechanism. [11] can rewrite the query while still computing the canonical model. [12] mainly filters out false answers using a filtering mechanism. The above approaches all have the disadvantage of being limited to lightweight ontology languages. QueryPie [22] implements a backward reasoning system through the proposed hybrid reasoning approach. The reasoning method computes a part of the forward closure, and the rest of the reasoning part is processed dynamically during the query parsing process. Shi Hui et al. [19] propose a scalable backward chaining-based reasoner, in which the optimization algorithm mentioned in query expansion sorts according to the number of variables contained in the query body clause and then reduces the reason time.

The above are some contributions to the field of ontology reasoning. Some are based on forward reasoning to compute the closure of database data uniformly, and the input query is pattern matched on the data after forward materialization to get the result. Others take the query as the target, extending and rewriting the query in reverse, reasoning about the data in a process contrary to the forward materialization.

Inspired by previous research, our method combines the backward reasoning method to design a forward materialization algorithm so that the forward reasoning system can no longer calculate too many redundant results and reduce redundant computational operations. Perform reverse parsing of the query to parse out the key information in the query, match the rules in the process, and calculate the semantic data containing the necessary explicit information and the semantic data that may reason useful implicit information. The query parsing algorithm for reverse parsing query is designed, and a KR-POOL and reason rule set that are constantly updated with the parsing process is maintained. A rule filtering algorithm is designed, which expands backward according to the query pattern, searches the effective rules recursively, and screens the key rules for the query from

a large number of rules in the rule base. Finally, the redundancy of large-scale semantic data is removed to optimize the data scale while ensuring completeness.

## 3 Preliminaries

We describe some necessary background knowledge in this section, such as RDF graphs, SPARQL queries, Description Logic (DL), and OWL Horst rules.

**Definition 1 (RDF Graph).** *U, B, and L exist, which are three disjoint infinite sets. U is a Uniform Resource Identifier (URI), B represents a blank node, and L is literals. A finite set of RDF triples* $(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$ *constitute an RDF graph, where s represents the subject, p represents the predicate, and o represents the object. A triple (s, p, o) is a statement of fact that s and o satisfy relation p or that the corresponding value of s with respect to property p is o.*

Figure 1 shows an RDF statement indicating that Joe knows Jane and an RDF graph consisting of multiple RDF statements.

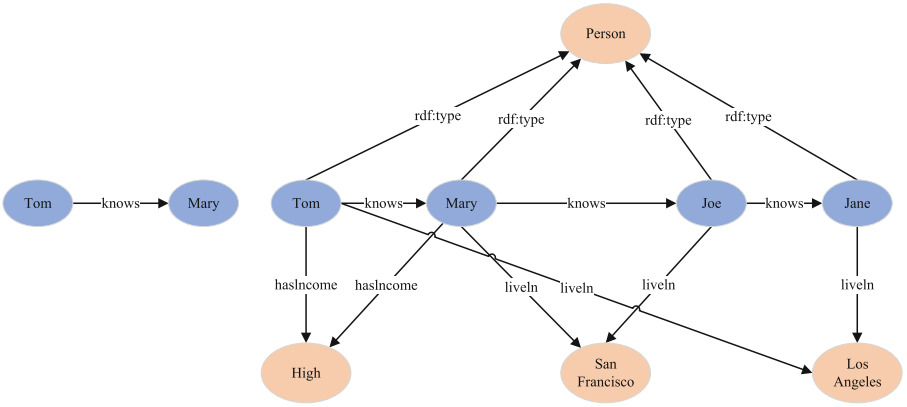



**Fig. 1.** Figure on the left side of an RDF statement, by multiple RDF statements consisting of RDF Graph on the right.

**SPARQL Query.** Q is a quadruple of the form (*q-type, m-dataset, pattern, sort-map*). *q-type* represents four query types: SELECT, ASK, CONSTRUCT, DESCRIBE. *m-dataset* specifies the target dataset for pattern matching. *pattern* P searches the input dataset for a specific subgraph and returns the result set of the map. *sort-map* is used to sort the set of mappings produced by pattern matching while returning a specified mapping window.

**Description Logic** can also be called concept representation language and term logic, which gives formal logic-based semantics. DL consists of concepts, relations, and individuals. Concepts describe the common properties of a set

of individuals, interpret concepts as unary predicates of sets of objects, and interpret relations as binary relations between objects. It is characterized by applying a large number of constructors to simple concepts, eventually creating more complex concepts. At the core of description logic is reasoning, that is, the knowledge that is implicitly represented from knowledge explicitly contained in a knowledge base.

The DL knowledge base $\mathcal{K}$ is usually composed of two parts: Tbox ($\mathcal{T}$) and Abox ($\mathcal{A}$). Among them, $\mathcal{T}$ is a set of assertions related to concepts and relations, describing the properties of concepts and relations. $\mathcal{A}$ is a collection of instance assertions that specify attributes of individuals or relationships between individuals. It consists of concept assertion and relation assertion. The most basic description language in DLs is ALC, and other description languages extend based on ALC. The symbols involved and their description are shown in Table 1, and the syntax and semantics of ALC are shown in Table 2 and Table 3.

**Table 1.** List of notations

| Notation | Description |
|---|---|
| a,b,c,d,e | Individual names |
| a,b,z | Concept names |
| C | Concepts |
| p,s | Role names |
| r | Roles |

DL are the basis for the standard Web ontology languages OWL and OWL 2. OWL provides powerful expressive capabilities, including OWL Lite, OWL DL, and OWL Full, which increase expressive capabilities and computational complexity in turn. The latest version of OWL, OWL 2, is also divided into OWL 2 DL and OWL 2 Full. OWL 2 Full has the strongest expressive power, but it is undecidable. OWL 2 DL adds a few restrictions to the combination of OWL 2 Full and RDFS, preserving decidability. $\mathcal{SROIQ}$ is the underlying

**Table 2.** Syntax of ALC

| Atomic concept | Description |
|---|---|
| A | Atomic concept |
| $\bot$ | The notion that any explanation is empty |
| $\top$ | The notion that contains any other concept |
| $\neg$A | Negation |
| $A \sqcup B$ | Take the union of concepts |
| $A \sqcap B$ | Take the intersection of concepts |
| $\exists R.A$ | Existential quantifier restriction |
| $\forall R.A$ | Universal quantifier restriction |

**Table 3.** Semantic of ALC

| Atomic concept | Semantic |
|---|---|
| $\bot$ | $\bot^I = \emptyset$ |
| $\top$ | $\top^I = \Delta$ |
| ¬A | $(\neg A)^I = \Delta / A^I$ |
| A ⊔ B | $(A \sqcup B)^I = A^I \cup B^I$ |
| A ⊓ B | $(A \sqcap B)^I = A^I \cap B^I$ |
| ∃R.A | $(\exists R.A)^I = \{a \in \Delta \mid \exists b \in \Delta ((a,b) \in R^I \wedge b \in A^I)\}$ |
| ∀R.A | $(\forall R.A)^I = \{a \in \Delta \mid \forall b \in \Delta ((a,b) \in R^I \rightarrow b \in A^I)\}$ |

logic of OWL 2 DL, and this section will focus on the background knowledge of $\mathcal{SROIQ}$.

$\mathcal{SROIQ}$ $\mathcal{K}$ is composed of RBox $\mathcal{R}$, TBox $\mathcal{T}$, and ABox $\mathcal{A}$, the concept of which is defined as $C := \bot \mid \top \mid \neg A \mid \{a\} \mid \geq mR.A \mid \exists R.A$.

A $\mathcal{SROIQ}$ $\mathcal{T}$ contains the concept inclusion axiom $C_1 \sqcap \cdots \sqcap C_n \sqsubseteq C$, the disjoint axiom $Dis(C_1, C_2)$, and the equivalent concept $C_1 \equiv C_2$.

The RBox is a finite set containing the role inclusion axioms or disjoint axioms. Role inclusion axioms are represented as $R_1 \sqsubseteq R_2$ or $R_1 \circ R_2 \sqsubseteq R_3$, and disjoint axioms are represented as $Dis(R_1, R_2)$. Inverse roles and symmetric roles are denoted as Inv(R) and Sym(R), also satisfy Inv(R) = $R^-$ and Inv(R) ≡ R if a role is symmetric. Also, the transitive role is represented as Trans(R), and $R \circ R \sqsubseteq R$ if a role is a transitive role. Fun(R) represents the functional role.

A $\mathcal{SROIQ}$ $\mathcal{A}$ without *unique name assumption* (UNA) includes individual inequality $\neq$ and individual equality $a \doteq b$.

**Reason Rules.** The rule we use for reason is the combination of the two fragments of OWL Horst and RDFS, shown in Table 6 in the Appendix. Each rule has at least one triple as antecedent, which triggers the outcome of the rule as long as the antecedent of the rule can be satisfied, and there is only one outcome. The OWL Horst fragment [21] is a more complex fragment. Its use is more common and can also be called pD* rule set.

## 4 Optimize Forward Reasoning with Queries

We describe in this section the implementation details of the proposed algorithms and techniques for optimizing the forward reasoning process. Because forward reasoning is affected by semantic data scale and rules, we optimize forward reasoning by combining terminological triples to filter rules and data optimization for large-scale semantic data driven by given queries. The terminological triple pattern represents those triple schemas that use terms from RDFS or OWL vocabularies as predicates or objects. We propose a query parsing algorithm and design two modules: rule filtering and data optimization.

### 4.1 Query Parsing Algorithm

We get the relevant SPARQL query before reasoning and then target the query for reverse rule and data filtering. First, we need to parse the query to obtain a collection of multiple query patterns. Second, parse the query patterns according to the query parsing algorithm to get the key information we need in the query and generate the key resource pool KR-POOL for further optimization. KR-POOL contains resource entities that have determined values resolved from the query. We only need to search for variables that meet the conditions based on these resource entities and then generate the solution map. The above process is shown in Algorithm 1.

**Algorithm 1.** Query Parsing Algorithm

**Input:** queryPath: the path of query
**Output:** queryPatternList: a collection of patterns for the query

```
query.readPath(queryPath)
List⟨String⟩ queryList = query.getQueryList();
queryList.forEach(queryString→{
BufferedReader br = new BufferedReader(new StringReader(queryString));
String content = br.readLine();
Boolean start = false;
while content ≠ '}' do
   if 'PREFIX' ∈ content  then
      preReplace = content.split(" ");
      preReplaceMap.put(preReplace[1],preReplace[2]);
   end if
   if '{' ∈ content then
      start = true
   end if
   if start then
      QueryModeConvert.convertQueryToModeList(content)→R;
   end if
end while
});
```

**Example 1.** Table 4 represents a SPARQL query, {p1,p2,p3,p4} is the query pattern set of this query, where pi represents the $i^{th}$ query pattern (i =1,2,3,4). After we get the query pattern set of this query, we perform pattern parsing according to the query pattern set. After parsing, it returns more fine-grained key resource entities than the original query pattern. We put these key resource entities into KR-POOL to form The initial set. KR-POOL is continuously updated in subsequent expansion operations. Figure 2 shows the key resource pool (KR-POOL) generated after parsing, which contains the most direct resources required by the query.

**Table 4.** An example of SPARQL query

| SPARQL Query Q |
|---|
| PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#><br>PREFIX benchmark: <http://semantics.crl.ibm.com/univ-bench-dl.owl#><br>SELECT ?x<br>WHERE {<br>?x rdf:type Publication . //p1<br>?x publicationAuthor ?y . //p2<br>?y rdf:type Faculty . //p3<br>?y isMemberOf <http://www.Department0.University0.edu> //p4<br>} |

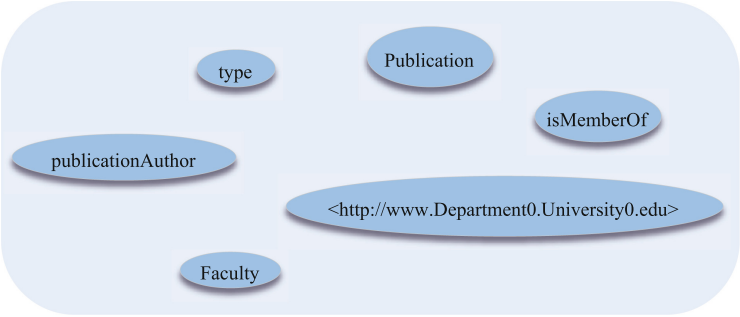



**Fig. 2.** An example of KR-POOL

### 4.2 Generate Reason Rule Set

In the process of knowledge reasoning, we first limit the range of resources we need according to the query to some extent, but this is not complete. A lot of implicit knowledge is not represented, and while the scope is defined, some rules are no longer needed because they do not lead to new results. However, reasoning on these rules will undoubtedly cause a waste of time, so we expand the key resource pool and filter the rules simultaneously to form a final reason rule set. The specific implementation details of our algorithm are shown in Algorithm 2.

**Example 2.** Assume that the knowledge base has two semantic data: *Publication0 type Article* and *Publication1 type Publication*. The existing terminological triplet is *Article SubclassOf Publication*. The KR-POOL obtained by query parsing has two resource entities, *type* and *Publication*. The rules we can match are R1, R2, R5, O13a, O14, and O15. According to the existing conditions, R5 is finally added to the reason rule set. The updated knowledge base is *Publication0 type Article*, *Publication1 type Publication*, and *Publication1 type Publication*.

**Algorithm 2.** Rule Filtering Algorithm

**Input:** OWLOntoMap: consists of OWL ontology
queryPatternList: consists of query patterns for the query
**Output:** R: extends resource
FO: filter Owl

```
queryPatternList.Rs→R
queryPatternList.Ro→R
queryPatternList.Rp→R
while resource = R.next do
  if resource ∈ OWLOntoMap.keySet() then
    OWLOntoMap→FO
    OWLOntoMap.get(resource )→R
  end if
end while
```

The terminological triples and semantic data in the above examples are simple cases. A real situation, however, is far more complicated than this, so the rules of selection and matching are more complex. Therefore, how to effectively screen and match rules is a key challenge.

We illustrate the process of rule filtering with query pattern p4. Rule filtering is a process in which the query is first decomposed in reverse, then matched with the rules, and the rules that can produce new results are put into the reason rules for the following reason. 1) The resource entities in KR-POOL are matched with the results of rules in the rule table, and the variable values in the antecedents of rules are determined by combining the known terminological triple to obtain a new query pattern after binding. 2) The newly obtained query pattern is taken as the target of the new round. Combined with terminological triple and rule results, the rule antecedents are found by matching recursion. During this process, the matching rules are added to the reason rule set, and the newly added resource entities are added to the KR-POOL until no new matching rules or resource entities are generated. For example, in the dashed line in Fig. 4, *isMemberOf* obtains the new query mode *University0 hasMember ?y* according to the rules O7 and terminological triple *hasMember OWL:inverseOf isMemberOf*, when we continue the recursion with the newly obtained query pattern, we find that the resulting reason rule set and the resource entity are not updated, and we consider the branch terminated. The final reason rule set is O7, R4. Figure 3 shows the updated key resource pool.

### 4.3 Optimizing Semantic Data

Rule filtering generates a set of reason rules by excluding rules that do not reason the results required by the relevant query. Therefore, in the process of reasoning on large-scale semantic data, it can effectively reduce invalid reasoning and shorten the reasoning time. Furthermore, with the final KR-POOL, our proposed data optimization algorithm can optimize large-scale semantic data in

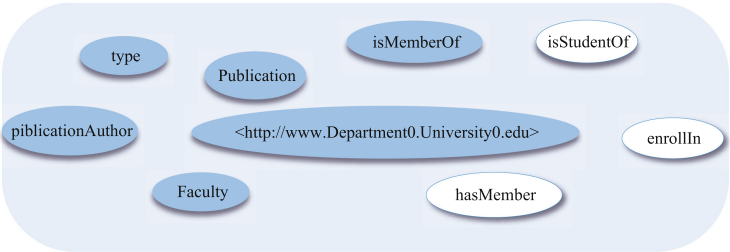



**Fig. 3.** An example of updated KR-POOL

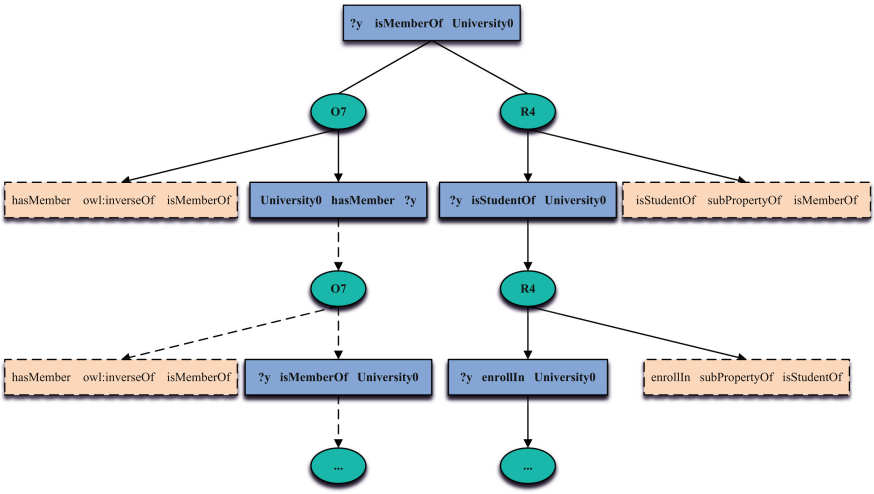



**Fig. 4.** The process of generating reason rule set

advance, thereby avoiding reason on irrelevant data and further shortening the reason time.

The data filtering process is given in Algorithm 3. According to the key resource entities in the resource pool and large-scale semantic data, efficient matching containing one or more key resource pool entity semantic data are preserved for subsequent forward reasoning. Unmatched data are excluded in advance, which not only reduces materialization time but also greatly optimizes memory.

**Example 3.** Suppose the following semantic data is waiting for a reason: 1) *ClericalStaff0 isMemberOf University0*, 2) *UndergraduateStudent388 isFriendOf AssociateProfessor6*, 3) *UndergraduateStudent0 enrollIn University0*, 4) *UndergraduateStudent30 hasSameHomeTownWith UndergraduateStudent349*, 5) *FullProfessor1 type Faculty*. The key resource pool is shown in Fig. 3, then data 1),3),5) are filtered out to continue the reasoning, and data 2),4) are eliminated.

**Algorithm 3.** Data filtering algorithm

**Input:** D: consists of RDF data
R: consists of resource
**Output:** AD: consists of RDF data after filter
1: **while** F = D.next **do**
2: **if** F.RS ∉ R and F.RO ∉ R and F.RP ∉ R **then**
3: Continue;
4: **end if**
5: F→AD
6: **end while**

## 5 The System and Implementation of SUMA-F

### 5.1 Architecture of SUMA-F

We implement three parts of Query Parsing Module, Rule Filter, and Data Filter in the forward reasoning system SUMA-F. The overall system architecture is shown in Fig. 5.

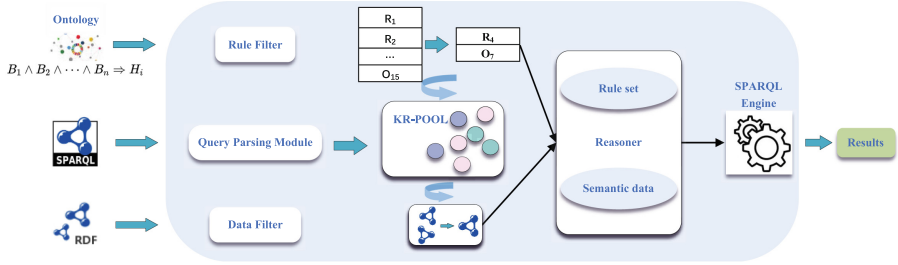



**Fig. 5.** Frame diagram of SUMA-F

The input of the system is Ontology, RDF data, SPARQL query, and OWL Horst rule, and the output is query result. A total of five modules are Query Parsing module, Rule Filter, Data Filter, Reasoner, and SPARQL Engineer.

Query Parsing Module takes queries as the input, parsing each query into multiple query mode sets, parsing the entities in each query mode in the query mode set, and placing the query mode related entities in the KR-POOL.

The Rule Filter module takes OWL Horst rule as input, matches the query pattern with the rule header according to KR-POOL and query pattern output by the Query Parsing Module, binds unknown variables with the input OWL 2 DL ontology, and continuously expands the query pattern. The matching rules are saved into the reason rule set in the expansion process, and the KR-POOL is updated with the newly added resource entities. This process continues recursively until no new rules are generated in the reason rule set, no new resource entities are added to the KR-POOL, and the recursion stops.

The Data Filter module takes large-scale RDF Data as input and combines the updated KR-POOL to match the three entities of the triple data with the resource entities in the KR-POOL. As long as more than one entity can be matched, the data are regarded as valid data. Otherwise, the data are discarded. After the Data Filter module, the large-scale data screen out some irrelevant data, and the data into the reasoning scale are greatly reduced, while reducing the reasoning scale also has a certain optimization.

The Reasoner module uses the filtered data, and the reason rule set generated by the Rule Filter module as the input for forward reason. The reason data and SPARQL query are used as the input of the SPARQL Engineer to obtain the final result.

## 6 Experiments

In this section, we implement our algorithm and compare it with other forward reason systems to demonstrate the reason performance and system scalability of SUMA-F. We simulate the dynamic database update scenario and verify that our method can effectively adapt to the frequent data update scenario. By calculating the percentage of irrelevant data reduced, the side reflects the optimization of memory.

### 6.1 Experimental Settings

SUMA-F is implemented in Java and runs on a single-node server. The node is based on the CentOS Linux release 7.9.2009 (Core) operating system. The CPU is 4 cores, the model is Intel(R) Xeon(R) CPU E5-4603 0 @ 2.00 GHz, and the memory is 128G.

**Datasets.** We use simulated and real datasets, and experiments are performed on these two datasets. Since UOBM is more expressive than LUBM [23], we chose UOBM as the simulation dataset and tested its 15 standard queries. DBpedia+ with additional tourism axioms on the DBpedia [2] dataset was selected as the real dataset for experimental evaluation of DBpedia+ axioms and 1024 queries against DBpedia+ provided by PAGOdA.

**Baselines.** We compare SUMA-F with three forward reason systems, SUMA, Pellet, and PAGOdA. SUMA has better performance in forward reasoning. Pellet is an OWL 2 DL reason engine and has reason completeness. PAGOdA shows better performance in scalability, and it is based on RDFox for reason.

**Evaluation Criteria.** We will design experiments in the following four dimensions: 1) Compare the completeness of reason with the forward reason system. Since the current backward and hybrid reasoning systems usually do not provide explicit instructions for completeness, we compare the completeness of our reasoning method with some complete forward reasoning systems on UOBM and DBpedia+. 2) Test the data expansibility of the system. We test the scalability of our system by recording reason and data loading times for each query on

datasets of different sizes. 3) We also verify the effectiveness of data filtering and rule filtering through experiments and manually simulate the reason performance of our system and other forward reason systems when the database is updated. 4) Calculating the percentage of the reduced data in the total data at different data scales reflects the optimization of memory from the side.

### 6.2 Experimental Results

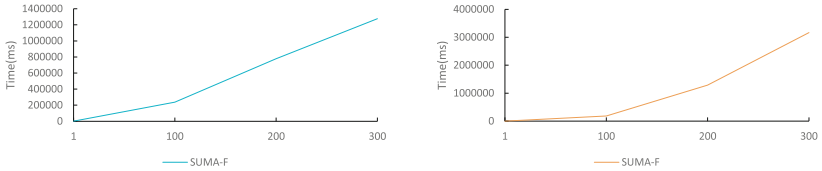
**The Soundness and Completeness Evaluation.** We compare reason completeness with SUMA, Pellet, and PAGOdA on UOBM(1), UOBM(100), and DBpedia+, respectively. We can reflect the completeness of reasoning from the side by counting the number of correct answers returned by different queries. Since Pellet cannot return in time the results of queries on both UOBM(100) and DBpedia+ datasets in a limited time, we denote the value as 0. We present the results from the completeness experiments in Table 5. One query on the UOBM dataset failed to return all answers. The number of correct answers that this query should return is 2465, and the number that our method returns is 2404. Despite missing 61 pieces of data, our reason completeness is still 97.5%. And it is speculated that the completeness of the loss is due to the existence of complex roles and relatively complex query patterns, which will continue to be optimized in the future. The number of queries on DBpedia+ that returned the correct answer is 1024, which reaches the completeness of reason.

**Table 5.** The number of queries with correct results

| Dataset | SUMA-F | SUMA | Pellet | PAGOdA |
|---|---|---|---|---|
| DBpedia+ | 1024 | 1024 | 0 | 1024 |
| UOBM(1) | 14 | 15 | 15 | 15 |
| UOBM(100) | 14 | 15 | 0 | 15 |

**The Scalability Test.** We test the scalability of UOBM on different data scales. Figure 6(a) shows the trend of data loading time as the data size increases. Figure 6(b) shows the total reason time for 15 standard queries on different data scales.

We test the materialization time according to different queries and compare it with the forward reason system. In Fig. 7, the red line represents the average materialization time, and it can be seen that the materialization time of SUMA-F is less than that of SUMA. For the Q5 query, SUMA-F materializes in 341 ms and returns all correct answers, while SUMA requires 543 ms.




(a) data loading time (b) total reason time

**Fig. 6.** Scalability for different data sizes on UOBM

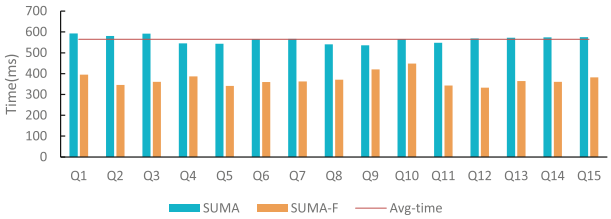
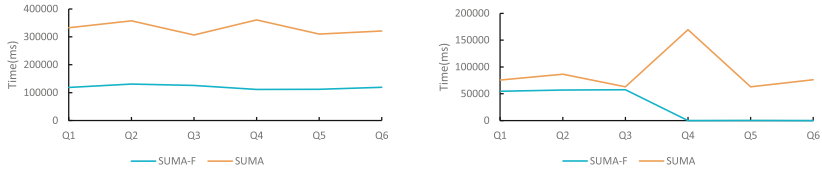



**Fig. 7.** Experimental results of materialization time

We randomly select six queries with different complexity and compare them with SUMA in terms of data loading time (Fig. 8(a)) and total reason time (Fig. 8(b)) on DBpedia+ dataset. From the results on the experimental graph, it can be seen that for all queries of different complexity, the reason time and data loading time of SUMA-F are much shorter than SUMA. When processing the three queries Q4, Q5, and Q6, due to the small number of answers, SUMA-F greatly reduces the reasoning time through the screening mechanism.




(a) data loading time (b) total reason time

**Fig. 8.** Experimental results on DBpedia+

**Effectiveness Testing of Data and Rule Filtering.** We compare the changes in reason performance of 15 queries before and after data filtering Fig. 9) and rule filtering (Fig. 10) on the UOBM dataset. It can be seen that the data and rule filtering method effectively improves the reason performance and reduces the materialization time. Thus, it is proved that our algorithm is effective for optimizing forward reason.

We manually simulate the update operation of the database, add irrelevant data to the data set to be reasoned in different proportions, and test the change of materialization time between SUMA-F and the traditional forward reason system. Irrelevant data means that for a specific query, no data related to the query answer can be inferred, that is, data that is not helpful for the enrichment of the query answer.

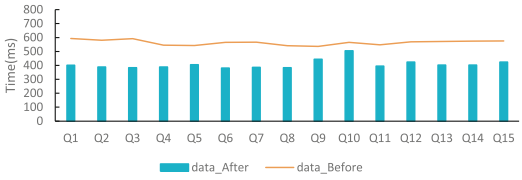



**Fig. 9.** Results of data filtering methods

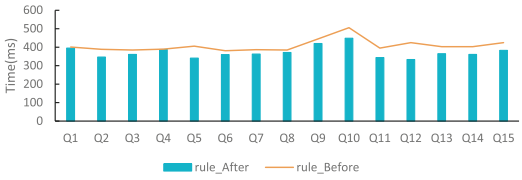



**Fig. 10.** Results of rule filtering methods

Traditional forward reasoning compute closures indiscriminately for such irrelevant data, while our system avoids reasoning on such data, effectively improving the overall performance. Figure 11 shows the change in materialization time for both systems as the proportion of irrelevant data increases. The value of the abscissa represents the ratio of the amount of added irrelevant data to the original data. It can be seen from the results that SUMA-F fluctuates relatively smoothly with the update of the database. At the same time, SUMA,

a traditional forward reason system, shows a large change in materialization time. The updated scale gradually becomes larger, and the upward trend of SUMA's reasoning time becomes more and more obvious. Through the experimental results, we show that SUMA-F reduces the cost of data update to a large extent and realizes it as a forward reasoning system desensitized to data.

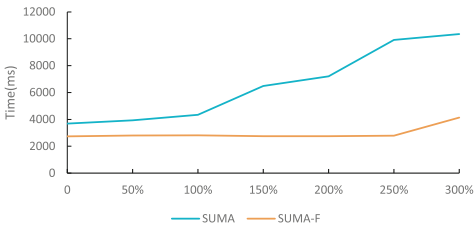



**Fig. 11.** Materialization time when the database is updated

## 7 Conclusion

We optimize forward reasoning in this paper with a query-based approach. KR-POOL is generated by parsing queries to filter reason rules and data. An originally large-scale data is refined into data that may produce all the answers required by the query. The resulting reason rule set is used to forward materialize, and the query is conducted on the materialized data set. Memory is effectively saved by downsizing rules and data. Since only valid data and rules are forward materialized, the reasoning efficiency is significantly improved at the expense of a part of reasoning completeness. In the future, we will continue to optimize the parsing order of queries and filter rules and data in a more fine-grained way. OWL 2 DL contains more complex roles, which causes us to lose a part of completeness in the design of the optimization forward reasoning method. We will further study to improve the completeness of reasoning in future work.

# Appendix

**Table 6.** RDFS and OWL Horst rulesets

| | Antecedents | Consequent |
|---|---|---|
| R1: | p rdfs:domain x, s p o | ⇒ s rdf:type x |
| R2: | p rdfs:range x, s p o | ⇒ o rdf:type x |
| R3: | p rdfs:subPropertyOf q, q rdfs:subPropertyOf r | ⇒ p rdfs:subPropertyOf r |
| R4: | s p o,p rdfs:subPropertyOf q | ⇒ s q o |
| R5: | s rdf:type x, x rdfs:subClassOf y | ⇒ s rdf:type y |
| R6: | x rdfs:subClassOf y, y rdfs:subClassof z | ⇒ x rdfs:subClassOf z |
| O1: | p rdf:type owl:FunctionalProperty, u p v , u p w | ⇒ v owl:sameAs w |
| O2: | p rdf:type owl:InverseFunctionalProperty, v p u, w p u | ⇒ v owl:sameAs w |
| O3: | p rdf:type owl:SymmetricProperty,v p u | ⇒ u p v |
| O4: | p rdf:type owl:TransitiveProperty, u p w, w p v | ⇒ u p v |
| O5: | v owl:sameAs w | ⇒ w owl:sameAs v |
| O6: | v owl:sameAs w, w owl:sameAs u | ⇒ v owl:sameAs u |
| O7a: | p owl:inverseOf q, v p w | ⇒ w q v |
| O7b: | p owl:inverseOf q, v q w | ⇒ w p v |
| O8: | v rdf:type owl:Class, v owl:sameAs w | ⇒ v rdfs:subClsaaOf w |
| O9: | p rdf:type owl:Property, p owl:sameAs q | ⇒ p rdfs:subPropertyOf q |
| O10: | u p v, u owl:sameAs x, v owl:sameAs y | ⇒ x p y |
| O11a: | v owl:equivalentClass w | ⇒ v rdfs:subClassOf w |
| O11b: | v owl:equivalentClass w | ⇒ w rdfs:subClassOf v |
| O11c: | v rdfs:subClassOf w, w rdfs:subClassOf v | ⇒ v rdfs:equivalentClass w |
| O12a: | v owl:equivalentProperty w | ⇒ v rdfs:subPropertyOf w |
| O12b: | v owl:equivalentProperty w | ⇒ w rdfs:subPropertyOf v |
| O12c: | v rdfs:subPropertyOf w, w rdfs:subPropertyOf v | ⇒ v rdfs:equivalentProperty w |
| O13a: | v owl:hasValue w, v owl:onProperty p, u p w | ⇒ u rdf:type v |
| O13b: | v owl:hasValue w, v owl:onProperty p, u rdf:type v | ⇒ u p w |
| O14: | v owl:someValuesFrom w, v owl:onProperty p,u p x, x rdf:type w | ⇒ u rdf:type v |
| O15: | v owl:allValuesFrom u, v owl:onProperty p,w rdf:type v, w p x | ⇒ x rdf:type u |

## References


1. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Sci. Am. **284**(5), 34–43 (2001)
2. Bizer, C., et al.: DBpedia-a crystallization point for the web of data. J. Web Semant. **7**(3), 154–165 (2009)
3. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: a generic architecture for storing and querying RDF and RDF schema. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 54–68. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-48005-6_7
4. Buron, M.: Efficient reasoning on large-scale heterogeneous data. Ph.D. thesis, Institut Polytechnique de Paris (2020)
5. Calvanese, D., et al.: Ontop: answering SPARQL queries over relational databases. Semant. Web **8**(3), 471–487 (2017)
6. Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: implementing the semantic web recommendations. In: Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers and Posters, pp. 74–83 (2004)

7. Chen, Y., Qian, T., Li, W., Liang, Y.: Exploiting composite relation graph convolution for attributed network embedding. J. Comput. Res. Dev. **57**(8), 1674–1682 (2020)
8. Cucala, D.T., Grau, B.C., Horrocks, I.: Sequoia: a consequence based reasoner for SROIQ. In: Proceedings of the 32nd International Workshop on Description Logics (DL 2019), pp. 1–12 (2019)
9. Eiter, T., Ortiz, M., Simkus, M., Tran, T.K., Xiao, G.: Query rewriting for Horn-Shiq plus rules. In: Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012), pp. 22–26 (2012)
10. Gómez, S.A., Fillottrani, P.R.: Materialization of OWL ontologies from relational databases: a practical approach. In: Pesado, P., Arroyo, M. (eds.) CACIC 2019. CCIS, vol. 1184, pp. 285–301. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-48325-8_19
11. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to query answering in DL-Lite. In: Proceedings of the 12th International Conference on the Principles of Knowledge Representation and Reasoning (KR 2010), vol. 10, pp. 247–257 (2010)
12. Lutz, C., Seylan, İ, Toman, D., Wolter, F.: The combined approach to OBDA: taming role hierarchies using filters. In: Alani, H., et al. (eds.) ISWC 2013. LNCS, vol. 8218, pp. 314–330. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41335-3_20
13. Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y., Liu, S.: Towards a complete OWL ontology benchmark. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 125–139. Springer, Heidelberg (2006). https://doi.org/10.1007/11762256_12
14. Motik, B., Nenov, Y., Piro, R., Horrocks, I., Olteanu, D.: Parallel materialisation of datalog programs in centralised, main-memory RDF systems. In: Proceedings of the 28th AAAI Conference on Artificial Intelligence, pp. 129–137 (2014)
15. Motik, B., Shearer, R., Horrocks, I.: Hypertableau reasoning for description logics. J. Artif. Intell. Res. **36**, 165–228 (2009)
16. Nenov, Y., Piro, R., Motik, B., Horrocks, I., Wu, Z., Banerjee, J.: RDFox: a highly-scalable RDF store. In: Arenas, M., et al. (eds.) ISWC 2015. LNCS, vol. 9367, pp. 3–20. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25010-6_1
17. Qin, X., Zhang, X., Yasin, M.Q., Wang, S., Feng, Z., Xiao, G.: SUMA: a partial materialization-based scalable query answering in OWL 2 DL. Data Sci. Eng. **6**(2), 229–245 (2021)
18. Qin, X., Zhang, X., Yasin, M.Q., Wang, S., Feng, Z., Xiao, G.: A partial materialization-based scalable query answering in OWL 2 DL. In: Proceedings of the 25th International Conference on Database Systems for Advanced Applications, pp. 171–187 (2020)
19. Shi, H., Maly, K., Zeil, S.: A scalable backward chaining-based reasoner for a semantic web. Int. J. Adv. Intell. Syst. **7**(1–2), 23–38 (2014)
20. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: a practical OWL-DL reasoner. J. Web Semant. **5**(2), 51–53 (2007)
21. Ter Horst, H.J.: Completeness, decidability and complexity of entailment for RDF schema and a semantic extension involving the OWL vocabulary. J. Web Semant. **3**(2–3), 79–115 (2005)
22. Urbani, J., van Harmelen, F., Schlobach, S., Bal, H.: QueryPIE: backward reasoning for OWL Horst over very large knowledge bases. In: Aroyo, L., et al. (eds.) ISWC 2011. LNCS, vol. 7031, pp. 730–745. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25073-6_46

23. Wang, S.-Y., Guo, Y., Qasem, A., Heflin, J.: Rapid benchmarking for semantic web knowledge base systems. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 758–772. Springer, Heidelberg (2005). https://doi.org/10.1007/11574620_54
24. Zhou, Y., Grau, B.C., Nenov, Y., Kaminski, M., Horrocks, I.: PAGOdA: pay-as-you-go ontology query answering using a datalog reasoner. J. Artif. Intell. Res. **54**, 309–367 (2015)