Künstliche Intelligenz manuscript No.
(will be inserted by the editor)

Is Your Database System a Semantic Web Reasoner?

Markus Krötzsch · **Sebastian Rudolph**

Received: date / Accepted: date

Abstract Databases and semantic technologies are an excellent match in scenarios requiring the management of heterogeneous or incomplete data. In *ontology-based query answering* (OBQA), application knowledge is expressed in ontologies and used for providing better query answers. This enhancement of database technology with logical reasoning remains challenging – performance is critical. Current implementations use time-consuming pre-processing to materialise logical consequences or, alternatively, compute a large number of large queries to be answered by a database management system (DBMS). Recent research has revealed a third option using recursive query languages to "implement" ontological reasoning in DBMS. For lightweight ontology languages, this is possible using the popular Semantic Web query language SPARQL 1.1, other cases require more powerful query languages like Datalog, which is also seeing a renaissance in DBMS today. Herein, we give an overview of these areas with a focus on recent trends and results.

Keywords Data management · Ontologies · SPARQL

1 Introduction

For several decades, database management systems (DBMS) had been considered a reliable, if somewhat boring, backbone of business IT, focusing on transactional workloads and record-like collections of uniform data. This has changed profoundly: recent years have seen exciting advancements throughout the field, including in-memory databases, NoSQL, and the ubiquitous "Big Data" paradigm.

M. Krötzsch · S. Rudolph
Fakultät Informatik
Technische Universität Dresden
01062 Dresden, Germany
E-mail: firstname.lastname@tu-dresden.de

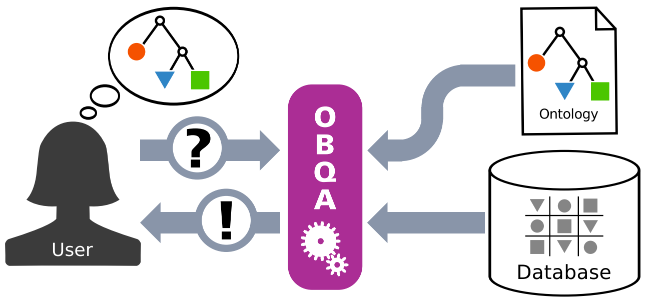

Fig. 1 Ontology-Based Query Answering

These developments are particularly interesting for semantic technologies, where DBMS are (re)discovered as a basis for knowledge representation and reasoning. There, too, we are confronted with larger and larger data collections, so that it seems natural to seek help from DBMS. On the other hand, data management can also benefit by incorporating semantic technologies. A prime example is the approach of *ontology-based query answering* (OBQA), where ontological background knowledge is used to improve the results of database queries. The goal of this technology is to create data management systems that can take the user's understanding of the application domain into account when answering queries. To do this, the mental model of the user is described formally in a so-called *ontology*, which is then used during query answering. Figure 1 illustrates this idea.

1.1 OBQA by Example

Let us consider a small example. Assume we had a database containing factual data about persons and their affiliations. We use the triple-based *Resource Description Framework* (RDF) to express this information.

`ex:markus`	`ex:worksAt`	`ex:tu-dresden`	.
`ex:sebastian`	`ex:worksAt`	`ex:tu-dresden`	.
`ex:tu-dresden`	`rdf:type`	`ex:University`	.

The first two facts connect the two individuals Markus and Sebastian with their affiliation TU Dresden. The last fact indicates that the individual TU Dresden is a university (or, more formally, that TU Dresden is an element of the *class* of all universities). According to human background knowledge, the works-at relation holding between two entities allow us to deduce more information, namely that the entity on the left is an employee and the entity on the right is an employer. We could add the respective information explicitly for Markus, Sebastian, and the TU Dresden, but we can use the expressivity of *RDF Schema* (RDFS) and add the ontological statements

`ex:worksAt`	`rdfs:domain`	`ex:Employee`	.
`ex:worksAt`	`rdfs:range`	`ex:Employer`	.

to indicate that the correspondence holds for every entity that occurs in a works-at statement. Using further ontological features of RDFS, we can describe a class hierarchy, to relate the classes mentioned so far with each other.

`ex:Employee`	`rdfs:subClassOf`	`ex:Person`	.
`ex:Person`	`rdfs:subClassOf`	`ex:LegalPerson`	.
`ex:University`	`rdfs:subClassOf`	`ex:Employer`	.
`ex:Employer`	`rdfs:subClassOf`	`ex:LegalPerson`	.

The purpose of expressing ontological information is to uniformly specify universally valid information that can be used to deduce new knowledge. This deductive capability can be captured in a formal way by means of a deduction calculus. In the case of RDFS, the corresponding calculus contains deduction rules such as the ones shown in Table 1.

Table 1 Some deduction rules for RDFS

$$\frac{x\;y\;z\;.\quad y\;\texttt{rdfs:domain}\;a\;.}{x\;\texttt{rdf:type}\;a\;.}\qquad\frac{x\;y\;z\;.\quad y\;\texttt{rdfs:range}\;a\;.}{z\;\texttt{rdf:type}\;a\;.}$$

$$\frac{x\;\texttt{rdf:type}\;a\;.\quad a\;\texttt{rdfs:subClassOf}\;b\;.}{x\;\texttt{rdf:type}\;b\;.}$$

$$\frac{a\;\texttt{rdfs:subClassOf}\;b\;.\quad b\;\texttt{rdfs:subClassOf}\;c\;.}{a\;\texttt{rdfs:subClassOf}\;c\;.}$$

Table 2 Inferences obtained in the example

`ex:markus`	`rdf:type`	`ex:Employee`	.
`ex:sebastian`	`rdf:type`	`ex:Employee`	.
`ex:Employee`	`rdfs:subClassOf`	`ex:LegalPerson`	.
`ex:University`	`rdfs:subClassOf`	`ex:LegalPerson`	.
`ex:markus`	`rdf:type`	`ex:Person`	.
`ex:sebastian`	`rdf:type`	`ex:Person`	.
`ex:markus`	`rdf:type`	`ex:LegalPerson`	.
`ex:sebastian`	`rdf:type`	`ex:LegalPerson`	.
`ex:tu-dresden`	`rdf:type`	`ex:University`	.
`ex:tu-dresden`	`rdf:type`	`ex:Employer`	.
`ex:tu-dresden`	`rdf:type`	`ex:LegalPerson`	.

Whenever the premise (upper part) of a rule holds true for some values of x, y, z, a, b, c, the conclusion (lower part) can be inferred. By iterative application of these rules, we can deduce many additional triples, shown in Table 2. The approach of OBQA simply is to take such inferred information into account when answering queries. Consider the following query:[1]

```
?x                rdf:type            ex:LegalPerson .
```

This query does not return any answers over the example database, but when taking the ontology into account, there are several matches with the inferred triples of Table 2 so that `ex:markus`, `ex:sebastian`, and `ex:tu-dresden` should be returned as answers.

1.2 From Vision to Reality

Unfortunately, the convenience of augmenting query results by inferences comes at a cost. Ontology-based query answering can be complicated in general, and in fact it is a long open problem whether OBQA is decidable at all for the W3C's widely used *OWL Web Ontology Language*. Even if it is, the computational complexity will most likely make it infeasible in practice. Expressive ontology languages generally require multi-exponential algorithms for query answering, and for the most expressive languages where OBQA is known to be decidable today, we do not have any elementary complexity bounds at all [15].

Practical problems start well before the theoretical computability limits. Lightweight ontology languages like RDFS feature low complexities even for query answering, and yet it is challenging to cope with the large datasets that are characteristic for modern data management applications. Key challenges are closely aligned with the three big *V*s that are often considered to characterise Big Data:

[1] We use `?` for variables in triple patterns like in SPARQL queries.

- *Volume.* The sheer amount of data is problematic. While polynomial algorithms are often considered tractable, an algorithm that exhibits even quadratic runtime behaviour *w.r.t. the size of the database* would be completely infeasible in practice.
- *Velocity.* Data changes at a quick rate. If it takes hours or days to solve a difficult logical entailment problem, the solution will already be meaningless when it arrives.
- *Variety.* The heterogeneity in data formats and schemas requires a significant amount of data integration. This is exactly the problem that OBQA addresses, but the practical requirements towards the expressivity and size of the ontology are significant.

These challenges are keeping both researchers and practitioners busy in the search of more adequate ontology languages, better algorithms, and more efficient implementations. Indeed, the work on OBQA approaches spans a wide spectrum from theory to practice. Applied activities, such as the EU IP *Optique* [9], are bringing recent research results to bear in industrial applications, whereas foundational works continue to explore theoretical properties of ontology and query languages (e.g., [3,16,5]).

1.3 Three Ways of OBQA

The goal of this paper is to give an overview of basic methods and recent advances in OBQA. The problem as such has a long history in computer science. From the viewpoint of traditional database technologies, OBQA corresponds to query answering over a *view* on the database that is defined by an ontology. The classical technique for solving this problem is *materialisation*, where views like the one shown in Table 2 are precomputed [11].

With the more recent interest in OBQA, an alternative approach became popular, where the user query is expanded at query time to find additional, inferred results. This method of *query rewriting* takes advantage of existing database technology for query answering, but may incur a significant communication overhead as many queries need to be answered. Some works have further combined query rewriting with materialisation to support more expressive ontology languages.

Finally, a most recent addition to the list of OBQA approaches targets Semantic Web databases that support the RDF query language SPARQL 1.1. Such queries are more powerful than traditional SELECT-PROJECT-JOIN queries in databases, and they can be used to “implement” some forms of logical reasoning entirely in queries.

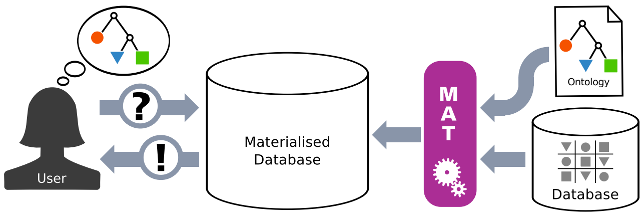

Fig. 2 Ontology-Based Query Answering with Materialisation

We now have a closer look at each of these methods, discuss their respective advantages and disadvantages, and give an outlook on future developments in this field.

2 Materialisation

The most direct approach of realising OBQA is to compute and store the required inferences, in our case the additional triples shown in Table 2. This computation can be performed by an iterated and exhaustive application of the deduction rules of an appropriate calculus like the one displayed in Table 1. Its result corresponds to a *materialised view* in databases [11]. Once the view is computed, it can be accessed like a regular database, taking advantage of the usual query answering algorithms and optimisations.

In its most basic form, materialisation can be implemented as an *ETL* (extract-transform-load) process, where the original data remains in its source, and a materialisation process creates extended data that is then stored in a dedicated database management system. This is illustrated in Fig. 2. A refined version of this approach is to integrate the original database and the materialised database into one system that can optimise the computation and view management process.

Materialisation is the basis of several DBMS that support ontological reasoning today. Indeed, the W3C’s lightweight ontology language OWL RL was specifically created to support efficient materialisation. Example systems include the RDF stores *OntoText GraphDB*, *Oracle Spatial and Graph*, and *RDFox*.

The primary challenge for materialisation approaches is the significant time required to build the view. Depending on the implementation, it might be necessary to rebuild the view whenever changes occur, or it might be possible to perform a partial recomputation. Even in the latter case, computation may not be fast enough to ensure continuous view updates in the presence of very frequent changes. On the other hand, many applications are not faced with such extreme velocity requirements, and the benefit of fast query answering

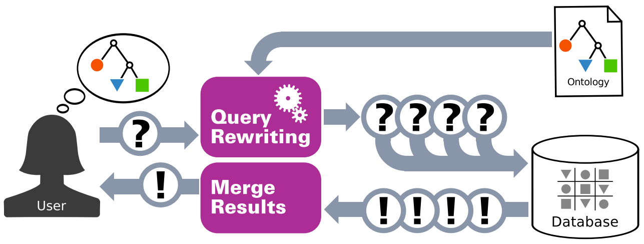

Fig. 3 Ontology-Based Query Answering with Query Rewriting

after materialisation may outweigh the disadvantages in this case.

Another general limitation of materialisation approaches is that they cannot be applied to all ontology languages. This is not obvious from our RDFS-based example, which may suggest that logical reasoning can be viewed as the result of a finite number of rule applications. Other ontology languages, however, can lead to an infinite amount of derived facts, which cannot be precomputed. This affects even lightweight ontology languages such as OWL QL and OWL EL, for which other approaches are needed.

3 Query Rewriting

Instead of computing a materialised view, one might also consider the inferences in OBQA to constitute a *virtual view* that is not to be stored. Instead, queries against that view are extended and reformulated so that they can be executed directly against the underlying data. Intuitively speaking, one transforms a query that asks for the inferred information into many queries that ask for all data that might cause such inferences to be derived.

This approach is known as *query rewriting* [12]. It is illustrated in Fig. 3, which also hints at the fact that the rewriting of a single query may lead to many queries against the database. Moreover, it can be seen that the ontology is used during rewriting, while the data is not needed at this stage: the rewritten queries work for any dataset. Query rewriting is closely related to the approach of *backward chaining* known in rule-based reasoning and especially logic programming.[2] However, the algorithms commonly used in OBQA are specific to the ontology languages used there, which ensures that rewriting always terminates.

Consider again our running example and the query for all legal persons. Considering the ontology of the example, it is clear that there are several possible cases in which an individual is classified as a legal person after performing all inferences: the individual might already be known as a legal person in the input, it might be classified as one of the subclasses of legal person, or it might be in a works-at relationship that entails it to be in such a subclass. In total, we therefore obtain the following set of query patterns that capture all possibilities for ?x to be a legal person:

```
?x    rdf:type     ex:Employee    .
?x    rdf:type     ex:Person      .
?x    rdf:type     ex:University  .
?x    rdf:type     ex:Employer    .
?x    rdf:type     ex:LegalPerson .
?x    ex:worksAt   ?y             .
?y    ex:worksAt   ?x             .
```

To answer the original query, one would now have to answer all of these queries against the input database, and then merge the results (i.e., eliminate duplicates[3]). As usual, this may require significant additional resources depending on the results' size.

Query rewriting in this sense is the most popular approach to OBQA for the lightweight ontology language OWL QL [7]. In many such applications, a traditional relational DBMS is used together with mapping rules that transform relational data to RDF. The query rewriting takes both the ontology and these mapping rules into account to produce SELECT-PROJECT-JOIN queries, which are expressed in SQL to be answered by the DBMS. This strategy is also known as *ontology-based data access* (OBDA), especially when integrating information from several legacy DBMS. Systems that support OBDA in this sense include *Ontop* and *Mastro*. Other systems integrate query rewriting with RDF stores within a single product; examples include *Complexible Stardog* and *OpenLink Virtuoso*.

The main advantage of query rewriting is that it avoids materialisation, and thus does not have to deal with the latency of view maintenance. Another possible advantage is that the approach is more easily deployed with legacy data sources since it suffices to access the database through a query interface.

The key problem of query rewriting is that the size and number of rewritten queries may be huge: exponentially many exponentially large queries are possible in the worst case [12]. Even if queries are mostly very simple, the execution of exponentially many queries over a legacy database interface is putting significant strain on the communication

[2] In this terminology, materialisation would correspond to *forward chaining*.

[3] Like always in OBQA, queries are evaluated under *set semantics*, i.e., results cannot contain duplicates. The semantics of ontological reasoning under *bag (multiset) semantics* is not usually considered, and would most likely be hard or impossible to implement in many cases.

channel. In addition, the rewritten queries may not be the type of load that a (relational) DBMS has been optimised for, and manual optimisation is difficult to inject into the process. This indicates that query rewriting might be more promising when closely integrated with the actual query engine, as done in Stardog and Virtuoso. The use of query rewriting systems on top of legacy DBMS, in contrast, requires significant optimisation.

4 Schema-Agnostic Query Rewriting

Traditional query rewriting approaches are based on conjunctive queries, which correspond to SELECT-PROJECT-JOIN queries in SQL, and their unions, as a kind of minimal interface that is supported by any DBMS. In practice, however, many DBMS feature much more powerful query languages, and a recent line of research has asked if these could be exploited for OBQA [4].

For example, the SPARQL 1.1 query language for RDF includes powerful features such as filters, property path expressions, and inline data (VALUES). Especially property paths are very interesting since they introduce a simple form of recursion into the query language. The idea is simple: instead of using an RDF property as the predicate in a triple pattern, one might specify a regular expression over RDF properties. To answer such queries, the DBMS will perform a graph traversal along paths that match the given regular expression. For example, the query

```
?x      ( ex:hasMother | ex:hasFather )+      ?y  .
```

finds all pairs of individuals `?x` and `?y` such that `?y` is an ancestor of `?x`. Here, | specifies alternative options (disjunction) and + specifies that a pattern may repeat once or more along a path.

The key for exploiting this capability for OBQA is the observation that such queries can be used to navigate not only the data, but also the ontology itself. Indeed, as shown in our examples, ontologies in RDFS and OWL can be serialised as sets of triples, which can also be queried when stored in an RDF database. A graphical representation of both ontology and data in our running example is shown in Fig. 5. For example, the following query retrieves all subclasses of legal person:

```
?a    rdfs:subClassOf*      ex:LegalPerson  .
```

This observation can be extended to another approach to OBQA, which has been called *schema-agnostic query rewriting* [4]. Indeed, if the ontology is treated as part of

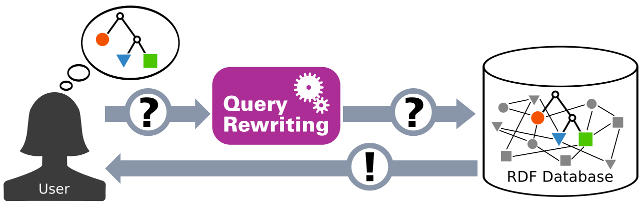

Fig. 4 Ontology-Based Query Answering with Schema-Agnostic Query Rewriting

the database, then the query rewriting does not depend on this schema-level information as it did in traditional query rewriting. The resulting workflow is illustrated in Fig. 4. A notable difference to traditional query rewriting is that the rewriting is only of polynomial size, so that it can be conveniently expressed in a single query.[4]

Returning to our running example that asks for all legal persons, we can see that a few more ontological features need to be taken into account. We can use the above scheme to find subclasses `?a` of `ex:LegalPerson`, but there are several possible reasons for why an individual `?x` might be an instance of `?a`. The following schema-agnostic rewriting uses the UNION operator of SPARQL to express disjunctions of the three relevant cases:

```
?a        rdfs:subClassOf*    ex:LegalPerson .
{
  { ?x        rdf:type            ?a . }  UNION
  { ?x        ?y                  ?z .
    ?y        rdfs:domain         ?a . }  UNION
  { ?z        ?y                  ?x .
    ?y        rdfs:range          ?a . }
}
```

Figure 6 sketches some possible matches of this query in the graph of Fig. 5. The two upper matches show that Markus and Sebastian are legal persons, while the three lower matches are three different ways of showing that TU Dresden is. It should be noted that this rewriting is specific to the selection of RDFS features that we have used in our example. Additional features, such as `rdfs:subPropertyOf`, could be taken into account, but this would lead to a larger query.

The immediate advantage of schema-agnostic query rewriting is that it avoids the exponential blow-up in queries that is inherent to traditional query rewriting. It also shares the advantage of eliminating the need for prior materialisation. An additional benefit is that it allows us to perform

[4] Using unions of queries, one could also express the rewritten queries in traditional query rewriting as a single query, but this query would be exponentially large. Issuing many small queries is less likely to overwhelm the DBMS, and is therefore preferable in this case.

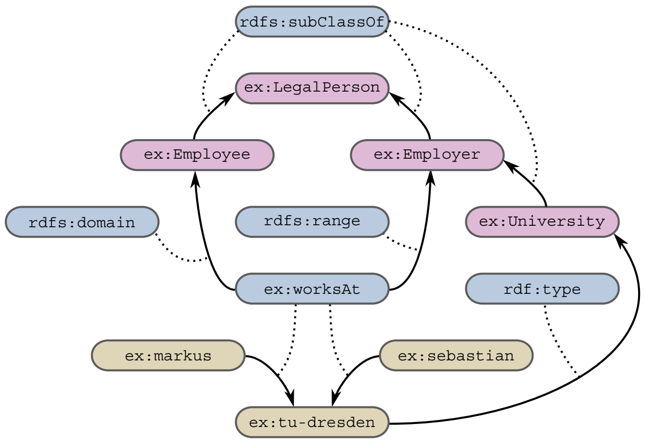

Fig. 5 RDF graph representation of running example; arrows connect triple subjects to objects; dotted lines connect predicates to triples

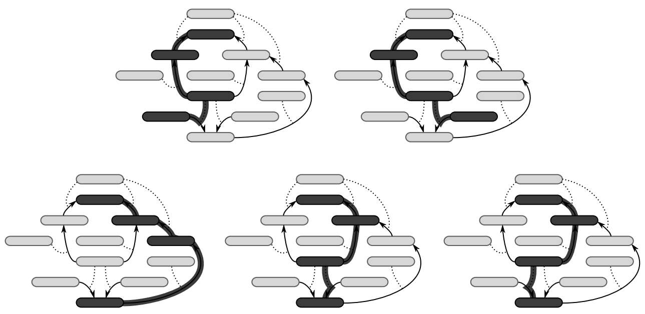

Fig. 6 Matches of the schema-agnostic rewriting of the example query in the graph of Fig. 5

OBQA against arbitrary SPARQL query services, even if these were not intended to support ontological reasoning.

In spite of these encouraging results, the practicability of schema-agnostic query rewriting largely depends on the features of the ontology and user query language. The more features an ontology language provides, the more cases have to be covered in the rewriting, and the more challenging it is for RDF stores to execute the query efficiently. Moreover, queries are becoming significantly more complex when allowing for query variables that match "generated" individuals, i.e., individuals that are not mentioned explicitly in the database but which can be inferred to exist in ontology languages like OWL QL. SPARQL cannot express such queries, but they are frequently studied in OBQA. Bischoff et al. have described schema-agnostic rewriting techniques for such queries and all of OWL QL [4]. While the rewritings remains polynomial, the queries tend to be very long (too long to show here) and they require the use of additional SPARQL 1.1 features.

It is too early to predict the impact of schema-agnostic rewriting in practical systems. However, as the implementation of SPARQL 1.1 requires efficient graph traversal to be supported in RDF stores, it seems natural to exploit this capability at least partially to implement reasoning. If integrated tightly into a system, this may not take the form of query rewriting, but might instead issue direct calls to the relevant physical graph traversal operations during query answering.

5 Beyond RDFS

RDFS is typically considered a "lightweight" ontology language with limited expressiveness. More expressive Semantic Web ontology formalisms exist, among which the *Web Ontology Language* OWL with its tractable fragments OWL QL, OWL EL and OWL RL is the most prominent example. More recently, *existential rules* ([2,6], also known as *tuple-generating dependencies* or *Datalog*$^{\pm}$) have gained interest as an alternative ontological formalism. Existential rules are based on Datalog, but extend it with the capability of *value invention* where the existence of a new individual can be inferred from a given situation. This ability of "generating" new individuals during reasoning is also a common feature in OWL and many of its sublanguages.

While each of the three practical query answering strategies that we have discussed so far is applicable to RDFS, this is no longer the case when considering more expressive fragments of OWL or existential rules. Beyond mere performance considerations, this also explains why such a variety of techniques has been developed.

Materialisation is only applicable if the ontology specifies deterministic ways of extending the given data, such that a unique database is obtained after materialisation. Logically, this property is characterised by the existence of a so-called *least model*, which can then be used for query answering. Intuitively, a materialised database corresponds to this least model. Ontology languages with this property disallow the free use of negation and disjunction, and therefore fall in the class of *Horn logics*. Existential rules and each of the previously mentioned tractable fragments of OWL satisfy this property, but OWL itself does not.

In addition, "Hornness" alone is not sufficient for being able to materialise a database, since the least model might be infinite. This behaviour may be caused by value invention in OWL and existential rules, which might lead to non-termination when attempting to exhaustively apply deduction rules in a forward-chaining way. This forward-chaining materialisation is known as the *chase* in databases and existential rules, and an important practical question therefore is whether the chase will terminate in spite of value invention. Sets of existential rules where this is the case have been termed *finite extension sets* [2]. Unfortunately, chase termination is undecidable in general, but many sufficient criteria have been proposed for detecting this situation in practice, and it could be shown that these criteria are indeed applicable to many existing OWL ontologies [8]. Finally, even if the least model is infinite, it might still be sufficiently well-behaved to admit a finite representation that can be used for query answering [18].

The second major OBQA technique, query rewriting, is also applicable only to some ontology languages. Again, termination needs to be guaranteed, this time of the backward-chaining query rewriting procedure. With the common choice of unions of SELECT-PROJECT-JOIN queries as target query formalism, only lightweight ontology languages are eligible.[5] Among the OWL dialects, only OWL QL qualifies for this method. In the existential rules world, rule sets where backward-chaining terminates are known as *finite unification sets* [2]. The most prominent representative of this class is *linear Datalog*$^{\pm}$, which significantly generalises OWL QL [6].

The applicability of schema-agnostic query rewriting relies on the expressivity of the target query formalisms. Complexity-theoretic considerations can be helpful to find out which query language is needed for which ontology language: evaluating a given query over an ontology and data cannot be more complex than evaluating the schema-agnostic rewriting of that query over a plain database. SPARQL 1.1 queries can be evaluated in NLogSpace complexity with respect to the size of the database, which agrees with the complexity of evaluating conjunctive queries in OWL QL with respect to the size of the ontology and data [1]. On the other hand, the respective complexity of OBQA is harder than NLogSpace for OWL EL and OWL RL, and certainly for OWL as a whole. OWL RL (PTime) can be covered by using *Datalog* queries as target language of the rewriting, while OWL EL (having PSpace complexity [17]) would require an even more expressive query formalism. There have been approaches similar to schema-agnostic rewriting for existential rules [10] in the sense that auxiliary information about the ontology are stored in the database and exploited for the subsequent querying. The details are different since, unlike OWL, existential rules do not come with an RDF-based encoding.

[5] More precisely, this approach can only be employed for formalisms where the data complexity of OBQA is in the complexity class AC^0, and hence strictly below logarithmic space.

6 The Future of Semantic Data Management

The current uptake of and emerging trends in OBQA indicate that it will be the focus of research and development activities for many years to come.

Current developments in databases are increasing the need for these technologies, and at the same time are helping to bring them into practice. Graph databases – not just for RDF but for graph-shaped data in general – are a major technology trend that is rapidly gaining traction in industry. Viewing datasets as graphs advocates the use of lightweight, flexible schemas that can accommodate heterogeneous datasets, which reinforces the need for technologies that can overcome the greater variety that ensues. On the other hand, a key feature of graph databases is their capacity of exploring the graph, based on some form of recursive queries. As illustrated in the initial works on schema-agnostic query rewriting, this feature can be beneficial for implementing reasoning services.

Another major trend are in-memory databases that keep most of the database in working memory in order to speed up computation. Persistence can still be provided by secondary storage, but runtime operations are largely decoupled from traditional database concerns that arise from the use of spinning disks. Such systems improve performance in areas that are most critical for systems that require logical inferencing: fast random access to data is important for recursive queries and recursive reasoning procedures alike; fast writing and indexing is useful for forward-chaining materialisation that may add large amounts of entailments. A pioneering in-memory store for rule-based reasoning on RDF is RDFox [13].

In addition to these general developments in databases, there are a number of important research trends in OBQA. Performance improvements and new algorithms bring applications into reach where one has to cope with rapidly changing data, such as data streams originating from sensor networks, social networks or other monitoring applications. By taking ontological data into account, more intelligent stream processing (also referred to as *stream reasoning*) becomes possible. The representation of time in ontologies and queries remains an important open research problem in this context.

Another promising direction of research is the exploration of new optimisation methods that perform well in practical scenarios. One interesting direction there is to take advantage of available background knowledge that is not used for inferring new entailments (like the ontology in OBQA) but that defines *constraints* that the database must satisfy. The knowledge that certain conditions are always true for the data can be used to reduce the search space of reasoning problems, e.g., by eliminating some queries in query rewriting [14]. Some such techniques are already used in the OBQA system *ontop*.

Optimisation can also be promising on the DBMS side. Query rewriting and schema-agnostic query rewriting both generate database queries algorithmically, and these queries may not be typical for traditional database applications. As OBQA applications are gaining importance, it is promising to enable DBMS to cope with this new kind of load. This requires close cooperation between researchers in semantic technologies and in databases. An example endeavour of this type is part of the work in Collaborative Research Centre *HAEC*,[6] where in-memory graph databases are developed alongside OBQA approaches.

Another area of active research that has already been hinted at in the previous section is the continued search for more appropriate ontology languages and according reasoning methods. Careful language design is essential to find the right trade-off between expressivity and complexity (or practical utility). Besides the evaluation of queries for such languages, the static analysis of queries bears a range of challenging theoretical and practical problems. For example, many results have recently been obtained regarding *query containment* in highly expressive query languages [3,16,5].

This illustrates the breadth of exciting research problems, ranging from theory to practice, that can be found in the field of ontology-based query answering. Yet this is merely a sample of the wealth of ongoing activities that are advancing the field at a rapid pace. Many other topics are currently under investigation, and ontology-based technologies will surely play their role in the future of data management.

Acknowledgements This work has been supported by the DFG in project DIAMOND (Emmy Noether Grant KR 4381/1-1) and in project HAEC (DFG SFB 912), which is part of the cfAED Cluster of Excellence. We thank the anonymous reviewers for their comments, which helped to improve the paper.

[6] DFG Sonderforschungsbereich 912: Highly Adaptive Energy-Efficient Computing

Markus Krötzsch is an Emmy Noether Research Group Leader at the Computer Science Faculty of TU Dresden. Before that, he got his PhD at the Karlsruhe Institute of Technology in 2010, and worked at the University of Oxford until 2013. His research has contributed to the fields of lightweight and rule-based ontology languages, query answering, and data management and integration. Notable projects of his include Wikipedia's knowledge base Wikidata, the content management system Semantic MediaWiki, and the ontology reasoner ELK.

Sebastian Rudolph is a full professor for computational logic at TU Dresden, Germany. Before, he worked in the knowledge management group at the Karlsruhe Institute of Technology. He obtained his PhD in mathematics at the Institute for Algebra at TU Dresden in 2006. His active research interests include logic-based knowledge representation, algebra, complexity theory, database theory, and computational linguistics. He serves on the editorial boards of the Journal of Web Semantics and the Journal on Data Semantics and is a member of the steering committee of several conferences.

References

1. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The DL-Lite family and relations. J. of Artificial Intelligence Research **36**, 1–69 (2009)
2. Baget, J.F., Leclère, M., Mugnier, M.L., Salvat, E.: On rules with existential variables: Walking the decidability line. Artificial Intelligence **175**(9–10), 1620–1654 (2011)
3. Bienvenu, M., ten Cate, B., Lutz, C., Wolter, F.: Ontology-based data access: A study through disjunctive datalog, CSP, and MMSNP. In: R. Hull, W. Fan (eds.) Proc. 32nd Symp. on Principles of Database Systems (PODS'13), pp. 213–224. ACM (2013)
4. Bischoff, S., Krötzsch, M., Polleres, A., Rudolph, S.: Schema-agnostic query rewriting for SPARQL 1.1. In: P. Mika, T. Tudorache, A. Bernstein, C. Welty, C.A. Knoblock, D. Vrandečić, P.T. Groth, N.F. Noy, K. Janowicz, C.A. Goble (eds.) Proc. 13th Int. Semantic Web Conf. (ISWC'14), *LNCS*, vol. 8796, pp. 584–600. Springer (2014)
5. Bourhis, P., Krötzsch, M., Rudolph, S.: Reasonable highly expressive query languages: Extended technical report. Available at https://ddll.inf.tu-dresden.de/web/Techreport3020 (2015)
6. Calì, A., Gottlob, G., Lukasiewicz, T.: A general datalog-based framework for tractable query answering over ontologies. J. Web Sem. **14**, 57–83 (2012)
7. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. J. of Automated Reasoning **39**(3), 385–429 (2007)

8. Cuenca Grau, B., Horrocks, I., Krötzsch, M., Kupke, C., Magka, D., Motik, B., Wang, Z.: Acyclicity notions for existential rules and their application to query answering in ontologies. J. of Art. Int. Research **47**, 741–808 (2013)
9. Giese, M., Calvanese, D., Haase, P., Horrocks, I., Ioannidis, Y., Kllapi, H., Koubarakis, M., Lenzerini, M., Möller, R., Rodriguez-Muro, M., Özcep, Ö., Rosati, R., Schlatte, R., Schmidt, M., Soylu, A., Waaler, A.: Scalable end-user access to big data. In: R. Akerkar (ed.) Big Data Computing. CRC Press (2013)
10. Gottlob, G., Kikot, S., Kontchakov, R., Podolskii, V.V., Schwentick, T., Zakharyaschev, M.: The price of query rewriting in ontology-based data access. Artif. Intell. **213**, 42–59 (2014)
11. Gupta, A., Mumick, I.S. (eds.): Materialized Views: Techniques, Implementations, and Applications. MIT Press (1999)
12. Kontchakov, R., Zakharyaschev, M.: An introduction to description logics and query rewriting. In: M. Koubarakis, G.B. Stamou, G. Stoilos, I. Horrocks, P.G. Kolaitis, G. Lausen, G. Weikum (eds.) Reasoning Web. Reasoning on the Web in the Big Data Era – 10th International Summer School, *LNCS*, vol. 8714, pp. 195–244. Springer (2014)
13. Motik, B., Nenov, Y., Piro, R., Horrocks, I., Olteanu, D.: Parallel materialisation of Datalog programs in centralised, main-memory RDF systems. In: C.E. Brodley, P. Stone (eds.) Proc. 28th AAAI Conf. on Artificial Intelligence (AAAI'14), pp. 129–137. AAAI Press (2014)
14. Rodriguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Query rewriting and optimisation with database dependencies in ontop. In: T. Eiter, B. Glimm, Y. Kazakov, M. Krötzsch (eds.) Informal Proc. 26th Int. Workshop on Description Logics (DL'13), *CEUR Workshop Proceedings*, vol. 1014, pp. 917–929. CEUR-WS.org (2013)
15. Rudolph, S., Glimm, B.: Nominals, inverses, counting, and conjunctive queries or: Why infinity is your friend! J. of Artificial Intelligence Research **39**, 429–481 (2010)
16. Rudolph, S., Krötzsch, M.: Flag & check: Data access with monadically defined queries. In: R. Hull, W. Fan (eds.) Proc. 32nd Symp. on Principles of Database Systems (PODS'13), pp. 151–162. ACM (2013)
17. Stefanoni, G., Motik, B., Krötzsch, M., Rudolph, S.: The complexity of answering conjunctive and navigational queries over OWL 2 EL knowledge bases. J. of Art. Int. Research **51**, 645–705 (2014)
18. Thomazo, M., Baget, J.F., Mugnier, M.L., Rudolph, S.: A generic querying algorithm for greedy sets of existential rules. In: G. Brewka, T. Eiter, S.A. McIlraith (eds.) Proc. 13th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'12), pp. 96–106. AAAI Press (2012)