# Using OpenStreetMap Data to Create Benchmarks for Description Logic Reasoners *

Thomas Eiter[1], Patrik Schneider[1], Mantas Šimkus[1], and Guohui Xiao[2]

[1] Institute of Information Systems, Vienna University of Technology, Vienna, Austria
[2] KRDB Research Centre for Knowledge and Data, Free University of Bozen-Bolzano, Italy

## 1 Introduction

*Description Logics* (DLs) are a well-established and popular family of decidable logics for knowledge representation and reasoning [6]. Several systems have been developed to reason over DL knowledge bases (KBs), which usually consist of a *TBox* and an *ABox*. A TBox describes the domain in terms of *concepts* and *roles*, while an ABox stores information about known *instances* of concepts and their participation in roles.

Naturally, classical reasoning tasks like TBox satisfiability and subsumption under a TBox have received most attention and many reasoners have been devoted to them, e.g. FaCT++ [24], HermiT [22], or ELK [14]. These and other mature systems geared towards TBox reasoning have been rigorously tested and compared (e.g., *JustBench* [7]) using several real-life ontologies like GALEN and SNOMED-CT.

A different category are reasoners for *ontology-based query answering* (*OQA*), which plays a major role in *ontology-based data access (OBDA)* [19]. They are designed to answer queries over DL KBs in the presence of large data instances (see e.g. Ontop [20], Pellet [21], Stardog [4] and OWL-BGP [16]). TBoxes in this setting are usually expressed in low complexity DLs, and are relatively small in size compared to the size of instance data. which makes OQA different from classical TBox reasoners.

Several works have considered testing OQA systems [12, 17, 18, 23, 13]. Despite this, it has been acknowledged that judging the performance of OQA reasoners is difficult due to the lack of publicly available benchmarks consisting of large amounts of *real-life instance data*. In this paper we consider publicly available geographic datasets as a source of test data for OQA systems. In similar spirit, recently some benchmarks have been proposed to test implementations of geospatial extensions of SPARQL [15, 10]. The latter benchmarks are geared towards testing spatial reasoning capabilities, and cannot be easily adapted for testing OQA systems. The main goal of this paper is to describe how benchmarks for OQA systems can be created from *OpenStreetMap* [2] (*OSM*) geospatial data by employing a rule-based data transformation framework.

The OSM project aims to collaboratively create an open map of the world. It has proven hugely successful and is constantly updated and extended. OSM data describes maps in terms of (possibly *tagged*) points, ways (geometries), and more complex aggregate objects called *relations*. We believe the following features make OSM a good source to obtain instance data for OQA reasoners:

* The authors were supported by FWF projects P24090 and P25518, the WWTF project ICT12-015, and the EU project Optique FP7-318338.

- Datasets of different sizes exist; e.g., OSM maps for all major cities, countries, and continents are directly available or can be easily generated.
- Depending on the location (e.g., urban versus rural), the density, separation, and compactness of object location varies strongly.
- Spatial objects have an inherent structure of containment, bordering, and overlapping, which can be exploited to generate topological relations (e.g., *contains*).
- Spatial objects are tagged with semantic information like the type of an object (e.g., hospitals). This information can be naturally represented as DL concepts and roles.

The paper is organized as follows. We first develop a formalization of OSM that allows to view OSM maps as relational structures. We then present a rule-based language to extract information from OSM data. Afterwards, we describe the implementation of our approach together with some generated benchmarks. Finally, we present some experimental results and conclude.

## 2 Formalization of Data Extraction

In this section, we first formally describe our model for OSM data, and then employ it to describe our rule-based language to extract instance data from OSM data.

Maps in OSM are represented using four basic constructs (a.k.a. *elements*): (1) *nodes*, which correspond to points with a geographic location; (2) *geometries* (a.k.a. *ways*), which are given as sequences of nodes; (3) *tuples* (a.k.a. *relations*), which are given as sequences of nodes, ways and tuples; (4) *tags*, which are used to describe metadata about nodes, ways and tuples. Geometries are used in OSM to express polylines and polygons, in this way describing streets, rivers, parks, etc. OSM tuples are used to relate several elements, e.g. to indicate the turn priority in an intersection of two streets.

*Formalization of OSM* We assume infinite mutually disjoint sets $\mathsf{M}_{\mathsf{nid}}, \mathsf{M}_{\mathsf{gid}}, \mathsf{M}_{\mathsf{tid}}$ and $\mathsf{M}_{\mathsf{tags}}$ of *node identifiers*, *geometry identifiers*, *tuple identifiers* and *tags*, respectively. We let $\mathsf{M}_{\mathsf{id}} = \mathsf{M}_{\mathsf{nid}} \cup \mathsf{M}_{\mathsf{gid}} \cup \mathsf{M}_{\mathsf{tid}}$ and call it the set of *identifiers*. An *(OSM) map* is a triple $\mathcal{M} = (\mathcal{D}, \mathcal{E}, \mathcal{L})$ such that:

1. $\mathcal{D} \subseteq \mathsf{M}_{\mathsf{id}}$ is finite set of identifiers called the *domain* of $\mathcal{M}$.
2. $\mathcal{E}$ is a function from $\mathcal{D}$ such that:
   (a) if $e \in \mathsf{M}_{\mathsf{nid}}$, then $\mathcal{E}(e) \in \mathbf{R} \times \mathbf{R}$;
   (b) if $e \in \mathsf{M}_{\mathsf{gid}}$, then $\mathcal{E}(e) = (e_1, \ldots, e_m)$ with $\{e_1, \ldots, e_m\} \subseteq \mathcal{D} \cap \mathsf{M}_{\mathsf{nid}}$;
   (c) if $e \in \mathsf{M}_{\mathsf{tid}}$, then $\mathcal{E}(e) = (e_1, \ldots, e_m)$ with $\{e_1, \ldots, e_m\} \subseteq \mathcal{D}$;
3. $\mathcal{L}$ is a *labeling* function $\mathcal{L} : \mathcal{D} \to 2^{\mathsf{M}_{\mathsf{tags}}}$.

Intuitively, in a map $\mathcal{M} = (\mathcal{D}, \mathcal{E}, \mathcal{L})$ the function $\mathcal{E}$ assigns to each node identifier a coordinate, to each geometry identifier a sequence of nodes, and to each tuple identifier a sequence of arbitrary identifiers.

*Enriching Maps with Computable Relations* The above formalizes the raw representation of OSM data. To make it more useful, we support incorporation of information that can be computed from a map. In particular, we allow to enrich maps with arbitrary computable relations over $\mathsf{M}_{\mathsf{id}}$. Let $\mathsf{M}_{\mathsf{rels}}$ be an infinite set of *map relation* symbols, each with an associated nonnegative integer, called the *arity*.

An *enriched map* is a tuple $\mathcal{M} = (\mathcal{D}, \mathcal{E}, \mathcal{L}, \cdot^{\mathcal{M}})$, where $(\mathcal{D}, \mathcal{E}, \mathcal{L})$ is a map and $\cdot^{\mathcal{M}}$ is a partial function that assigns to a map relation symbol $R \in \mathsf{M}_{\mathsf{rels}}$ a relation $R^{\mathcal{M}} \subseteq \mathcal{D}^n$, where $n$ is the arity of $R$. In this way, a map can be enriched with externally computed relations like the binary relations "is closer than 100m", "inside a country", "reachable from", etc. For the examples below, we assume that an enriched map $\mathcal{M}$ as above always defines the unary relation $\mathsf{Tag}_\alpha$ for every tag $\alpha \in \mathsf{M}_{\mathsf{tags}}$. In particular, we let $e \in \mathsf{Tag}_\alpha^{\mathcal{M}}$ iff $\alpha \in \mathcal{L}(e)$, where $e \in \mathcal{D}$. We will also use the binary relation $\mathsf{Inside}(x, y)$, which captures the fact the location of a point $x$ is inside a geometry $y$.

*A Rule Language for Data Extraction* We define a rule-based language that can be used to describe how an ABox is created from an enriched map. Our language is based on *Datalog with stratified negation* [5]. Let $\mathsf{D}_{\mathsf{rels}}$ be an infinite set of *Datalog relation symbols*, each with an associated *arity*. For simplicity, and with a slight abuse of notation, we assume that DL concept and role names form a subset of Datalog relations. Formally, we take an infinite set $\mathsf{D}_{\mathsf{concepts}} \subseteq \mathsf{D}_{\mathsf{rels}}$ of unary relations called *concept names* and an infinite set $\mathsf{D}_{\mathsf{roles}} \subseteq \mathsf{D}_{\mathsf{rels}}$ of binary relations called *role names*. Let $\mathsf{D}_{\mathsf{vars}}$ be a countably infinite set of *variables*. Elements of $\mathsf{M}_{\mathsf{id}} \cup \mathsf{D}_{\mathsf{vars}}$ are called *terms*.

An *atom* is an expression of the form $R(\boldsymbol{t})$ or $\neg R(\boldsymbol{t})$, where $R$ is a map or a Datalog relation symbol of arity $n$, and $\boldsymbol{t}$ is an $n$-tuple of terms. A *rule* $r$ is an expression of the form $B_1, \ldots, B_n \to R(\boldsymbol{t})$, where $B_1, \ldots, B_n$ are atoms (called *body atoms*) and $R(\boldsymbol{t})$ is an atom with $R$ a Datalog relation. We assume *Datalog safety*, i.e. each variable of a rule occurs in a non-negated body atom. A *program* $P$ is any finite set of rules. A rule or program is *ground* if it has no occurrences of variables. We only consider *stratified* programs (see [5] for a definition).

The semantics of a program $P$ is given relative to an enriched map $\mathcal{M}$. The *grounding* of a program $P$ w.r.t. $\mathcal{M}$ is the (ground) program $\mathsf{ground}(P, \mathcal{M})$ that can be obtained from $P$ by substituting in all possible ways the variables in rules of $P$ with identifiers occurring in $\mathcal{M}$ or $P$. We use a variant of the Gelfond-Lifschitz reduct [11] to get rid of map atoms in a program. The *reduct* of $P$ w.r.t. $\mathcal{M}$ is the program $P^{\mathcal{M}}$ obtained from $\mathsf{ground}(P, \mathcal{M})$ as follows:

(a) From the body of every rule $r$ delete every map atom $\neg R(\boldsymbol{t})$ with $\boldsymbol{t} \notin R^{\mathcal{M}}$.

(b) Delete every rule $r$ whose body contains a map atom $\neg R(\boldsymbol{t})$ with $\boldsymbol{t} \in R^{\mathcal{M}}$.

Observe that $P^{\mathcal{M}}$ is an ordinary stratified Datalog program with identifiers acting as constants. We let $PM(\mathcal{M}, P)$ denote the *perfect model* of the program $P^{\mathcal{M}}$. See [5] for the construction of $PM(\mathcal{M}, P)$ by fix-point computation along the stratification. We are finally ready to extract an ABox. Given a map $\mathcal{M}$ and a program $P$, we denote by $\mathsf{ABox}(\mathcal{M}, P)$ the restriction of $PM(\mathcal{M}, P)$ to the atoms over concept and role names.

We next illustrate some of the features and advantages of the presented rule language. E.g. the following rule collects in the role `hasCinema` the cinemas of a city (we use a sans-serif and a typewriter font for map and Datalog relations, respectively):

$$\mathsf{Point}(x), \mathsf{Tag}_{\mathsf{cinema}}(x), \mathsf{Geom}(y), \mathsf{Tag}_{\mathsf{city}}(y), \mathsf{Inside}(x, y) \to \texttt{hasCinema}(y, x).$$

Negation in rule bodies can be used for default conclusions. E.g. the following rule states that recreational areas include all parks that are not known to be private:

$$\mathsf{Geom}(x), \mathsf{Tag}_{\mathsf{park}}(x), \neg\mathsf{Tag}_{\mathsf{private}}(x) \to \texttt{RecreationalArea}(x).$$

## 3 Mapping Framework and Implemenation

**Mapping Framework**. The framework allows the user to define the data transformation for the instance generation in a declarative manner. As simplicity and extensibility are two of our main goals, we address these on two levels.

The first level concerns the extensibility of the mapping language and the related evaluation method by supporting: (a) An *ETL mode* which resembles the *extract*, *transform*, and *load* (ETL) process of classical data transformation tools; (b) A *Datalog mode* which offers the features of Datalog with stratified negation. The second level concerns the data sources (e.g., OSM) and external evaluation (e.g. calculating spatial relations). In certain cases we have to use external functionality for more sophisticated computations. E.g., in the *ETL mode*, we treat the scripts as custom data source or as simple transformations between input and output.

The Datalog and an ETL language are combined in the following custom mapping language. The *Data Source Declarations* are concerned with the general definitions as PostgreSQL/PostGIS connections. The *ETL Mapping Axioms* define a single ETL step, where the syntax is an extension of the Ontop mapping language. It is defined either as a pair of *source* and *target* or as a triple of *source*, *transform*, and *target*. The following sources and targets are currently available: (a) PostgreSQL/PostGIS with SQL statements as a source or target; (b) Text files with regular expressions as a source or target; (c) RDF files with SPARQL queries as a source; (d) External Python scripts and constants as a source. The *Datalog Rules* are based on the syntax of the library pyDatalog and embedded into the *source* and *target* of a mapping axiom.

**Implementation**. The main Python 2.7 script of the developed generation framework is called as follows: $\mathrm{generate.py\ -mappingFile\ file.txt\ -mode\ etl|datalog}$

The *ETL mode* is designed for bulk processing, where scalability and performance is crucial and complex calculations are not used or moved into the external scripts. We implemented the computation in a *data streaming*-based manner. The *Datalog mode* is designed for the features of stratified Datalog, where the entire result is always in-memory. In this mode, dependencies between sources and targets are considered by the declarative nature of Datalog. We use the same source and target components as in the ETL mode but follow a computation of three steps: 1. The source components are evaluated to store the n-tuples in the EDB; 2. The stratified Datalog program is evaluated; 3. The IDB predicate outputs are processed by the target components.

The *architecture* naturally results from the two modes, and the source and target components. Besides the generation script, we already provide a selection of *external scripts* for processing OSM data. E.g., $\mathrm{spatialRelationsReader.py}$ calculates the spatial relations (e.g., contains) between two RDF files.

## 4 Benchmarks and Results

All the mapping files and test data for the introduced benchmark are available online [1].

**OSM Dataset and Benchmark Ontology**. OSM offers different subset databases with different sizes and structures. Depending on the requirement, one could extract subsets fulfilling a variety of criteria as instance sizes, density, or structure (e.g., spatial topological or graph-like). For the base dataset [3], we chose the cities of *Cork*, *Riga*, *Bern*,

and *Vienna*. The cities are of increasing size and are European capitals or major cities with a high density of objects in the center and decreasing density towards the outskirts.

The chosen DL-Lite$_R$ [8] ontology for the benchmarks is taken from the MyITS project [9]. It is tailored to geospatial data sources and beyond to MyITS specific sources (e.g., a restaurant guide). The top level is built on *GeoOWL*, the second level based on the nine top-features of *GeoNames*, and the third level is built mainly from OSM categories. The ontology is for DL-Lite$_R$ systems of average difficulty, since it does contain only a few existentials quantification on the right-hand side of the inclusion axioms. However, due to its size and concept and role hierarchy depth, it posses a challenge regarding the rewritten query size.

**Spatial Relations Benchmark (BM1)**. For the cities, we transform all the amenities (e.g., restaurants), leisure areas (e.g., playgrounds), and shops to concept assertions. Then, we calculate the roles for the spatial relations *inside* between leisure areas and amenities. Further we generate several *next* relations between amenities and shops. The different *next* relations are generated according to the distances of 50m, 100m, 250m, and 500m between two objects. We define five queries that use the different "spatial" roles. In $q_1$ we evaluate over general concepts with a low selectivity of the role `contains`. The query $q_2$ determines which shops are next to amenities, which in turn are inside a park. Query $q_3$ resembles a star shaped pattern of relationships and uses the role `hasCuisine` derived from OSM tags. In query $q_4$, we have a triangular pattern of relationships and query amenities which are located in two leisure areas that overlap. With $q_5$, we illustrate that arbitrary role chains can be used by connecting `next`.

$$
\begin{aligned}
\mathtt{q}_1(y,z) &\leftarrow \mathtt{Amenity}(y), \mathtt{contains}(z,y), \mathtt{Leisure}(z)\\
\mathtt{q}_2(x,y,z) &\leftarrow \mathtt{Shop}(x), \mathtt{hasOperator}(x,w), \mathtt{SupermarketOp}(w), \mathtt{next}(x,y),\\
&\quad\ \mathtt{Amenity}(y), \mathtt{inside}(y,z), \mathtt{Leisure}(z)\\
\mathtt{q}_3(x,y,z) &\leftarrow \mathtt{Amenity}(x), \mathtt{hasQualitativeValue}(x,u), \mathtt{Cuisine}(u), \mathtt{next}(x,y),\\
&\quad\ \mathtt{Shop}(y), \mathtt{contains}(z,x), \mathtt{Leisure}(z)\\
\mathtt{q}_4(x) &\leftarrow \mathtt{Amenity}(x), \mathtt{next}(x,y), \mathtt{Leisure}(y), \mathtt{intersect}(y,z), \mathtt{Leisure}(z), \mathtt{next}(x,z)\\
\mathtt{q}_5(x,y) &\leftarrow \mathtt{Amenity}(x), \mathtt{next}(x,z), \mathtt{next}(z,y), \mathtt{Shop}(y)
\end{aligned}
$$

**Evaluation and Results**. We consider the following query answering systems: (a) Pellet [21], an OWL reasoner including an ABox query engine supporting answering conjunctive queries using optimization techniques of query simplification and query reordering (b) Stardog, the successor of Pellet, which is a commercial semantic graph database supporting SPARQL (c) OWL-BGP [16], a SPARQL wrapper for OWL-API based reasoners (e.g. HermiT used in this experiment). Finally we remark that Ontop under "classic mode" in principle can also answer queries over ontologies directly. However as currently the "classic mode" in Ontop is not well optimized, a proper evaluation is not feasible. Moreover since Ontop is designed to also take R2RML mappings into account, comparing Ontop with "pure" OQA systems is not done in this paper. The experiments were performed on a HP Proliant Linux server with 144 cores @3.47GHz, 106GB of RAM and a 1TB 15K RPM HD. The systems were evaluated on Java 7 with 8GB memory for the JVM. The timeouts are set to 10min (indicated by '-') for query answering and to 1h for ontology loading.

The evaluation results are presented in Table 1, where S stands for Stardog, P for Pellet, and OB for OWL-BGP, and show that BM1 is indeed challenging for instances as *Vienna*. We can see a gradual increase of difficulties with increasing city size and the

Table 1: Evaluation results on BM1 (time in sec.)

| | Cork | | | Riga | | | Bern | | | Vienna | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | S | P | OB | S | P | OB | S | P | OB | S | P | OB |
| $next_{250}$ | | | | | | | | | | | | |
| $q_1$ | 1.36 | 1.37 | 0.01 | 1.70 | 2.92 | 0.02 | 2.37 | 5.76 | 0.38 | 2.87 | 16.92 | 0.48 |
| $q_2$ | 1.56 | 0.04 | 0.02 | 1.55 | 0.05 | 0.05 | 1.90 | 0.08 | 0.14 | 2.21 | 15.83 | 1.04 |
| $q_3$ | 1.99 | 0.10 | 0.02 | 2.27 | 0.11 | 0.05 | 4.54 | 0.29 | 0.15 | 16.95 | 16.49 | 1.09 |
| $q_4$ | 1.90 | 0.10 | 0.29 | 1.88 | 0.51 | 0.60 | 5.07 | 1.05 | 1.75 | 21.31 | 20.99 | 16.39 |
| $q_5$ | 5.65 | 0.93 | 4.62 | 12.54 | 1.62 | 27.36 | 47.90 | 4.94 | 142.40 | - | - | - |
| load | 2.12 | 2.69 | 5.10 | 2.58 | 4.44 | 11.47 | 3.77 | 7.79 | 24.40 | 7.66 | 24.62 | 128.06 |
| $next_{500}$ | | | | | | | | | | | | |
| $q_1$ | 1.56 | 1.50 | 0.01 | 1.84 | 3.99 | 0.02 | 1.99 | 7.67 | 0.40 | 2.32 | 48.03 | - |
| $q_2$ | 1.42 | 0.04 | 0.03 | 1.63 | 0.10 | 0.12 | 1.83 | 7.86 | 0.31 | 3.84 | 52.23 | - |
| $q_3$ | 1.97 | 0.10 | 0.03 | 2.41 | 0.15 | 0.11 | 9.36 | 8.29 | 0.30 | 26.96 | 38.21 | - |
| $q_4$ | 1.45 | 0.20 | 0.23 | 2.11 | 1.84 | 23.88 | 22.42 | 10.77 | 4.13 | 8.59 | 74.84 | - |
| $q_5$ | 16.48 | 2.49 | 19.51 | 75.28 | 15.26 | 148.06 | - | - | - | - | - | - |
| load | 2.50 | 3.38 | 5.83 | 3.42 | 7.66 | 15.65 | 4.97 | 13.59 | 35.85 | 15.22 | 86.56 | - |

distance of $next$. $q_1$ and $q_2$ can be seen as the baseline results with low/high selectivity and pose no challenge to the reasoners. With the queries $q_3$, $q_4$, and $q_5$, we can see a breaking point where the performance starts to decline for all reasoners. For Stardog this is *Vienna* with $next_{250}$ (a $next$ distance of 250), for Pellet this is *Vienna* $next_{250}$, and for OWL-BGP it declines with *Vienna* $next_{250}$. Remarkably, every reasoner has the following strengths: (a) OWL-BGP has the best performance in the queries $q_1$, $q_2$, and $q_3$; (b) Stardog outperforms the other in the large instances of $q_4$, and $q_5$; (c) Pellet is best at the the medium to large instances of $q_4$, and $q_5$. Note that query $q_5$ is the most challenging one due to the join of the large $next$ relations, which are 939 988 (resp. 2 857 488) instances for $next_{250}$ (resp. $next_{500}$) in *Vienna*.

## 5 Conclusion and Outlook

In this paper, we have presented a framework for extracting instance data from OSM. Based on the framework, we have evaluated some state-of-the-art OQA systems with our benchmark and showed that the respective systems have their strengths and weaknesses depending on the particular input query, and the size and the structure of the input ABox.

Future research is directed to variants and extensions of the framework. We aim to extend the implementation to capture more input and output sources, further parameters (e.g. various degrees of graph connectedness), and services. An important functional extension would allow us to remove some assertions that are implied by the input ontology, in this way the information incompleteness could be better controlled. Another extension could be related to automatic generation of TBoxes by means of analysis of geospatial data. Further, our benchmark ontology could be extended to other important DLs as $\mathcal{EL}$ and its relatives. Finally, based on our open repository, we aim to collect a wider range of benchmarks with the related transformations and data extracts.

## References


1. City-bench. http://ghxiao.github.io/city-bench. Accessed: 2014-05-20.
2. Openstreetmap. http://www.openstreetmap.org. Accessed: 2014-05-01.
3. Openstreetmap data repository. http://download.bbbike.org/osm/bbbike/. Accessed: 2014-04-14.
4. Stardog. http://stardog.com/. Accessed: 2014-05-01.
5. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
6. Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. 2nd edition, 2007.
7. Samantha Bail, Bijan Parsia, and Ulrike Sattler. Justbench: A framework for OWL benchmarking. In *Proc. of ISWC 2010*, pages 32–47. Springer, 2010.
8. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. Autom. Reasoning*, 39(3):385–429, 2007.
9. Thomas Eiter, Thomas Krennwallner, and Patrik Schneider. Lightweight spatial conjunctive query answering using keywords. In *Proc. of ESWC 2013*, pages 243–258, 2013.
10. George Garbis, Kostis Kyzirakos, and Manolis Koubarakis. Geographica: A benchmark for geospatial rdf stores. *CoRR*, abs/1305.5653, 2013.
11. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, pages 1070–1080, 1988.
12. Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics*, 3(2-3):158 – 182, 2005.
13. Martha Imprialou, Giorgos Stoilos, and Bernardo Cuenca Grau. Benchmarking ontology-based query rewriting systems. In *Proc. of AAAI 2012*, 2012.
14. Yevgeny Kazakov, Markus Krötzsch, and František Simancík. Concurrent classification of EL ontologies. In *Proc. ISWC 2011*, pages 305–320, Berlin, Heidelberg, 2011. Springer.
15. Dave Kolas. A benchmark for spatial semantic web systems. In *4th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2008)*, October 2008.
16. Ilianna Kollia and Birte Glimm. Optimizing SPARQL query answering over OWL ontologies. *J. Artif. Intell. Res. (JAIR)*, 48:253–303, 2013.
17. Li Ma, Yang Yang, Zhaoming Qiu, Guotong Xie, Yue Pan, and Shengping Liu. Towards a complete owl ontology benchmark. In *Proc. of ESWC 2006*, pages 125–139. Springer.
18. Héctor Pérez-Urbina, Ian Horrocks, and Boris Motik. Efficient query answering for OWL 2. In *Proc. of ISWC 2009*, pages 489–504, 2009.
19. Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008.
20. Mariano Rodriguez-Muro, Roman Kontchakov, and Michael Zakharyaschev. Ontology-based data access: Ontop of databases. In *Proc. of ISWC 2013*. Springer, 2013.
21. E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *J. Web Sem.*, 5(2):51–53, 2007.
22. Giorgos Stoilos, Birte Glimm, Ian Horrocks, Boris Motik, and Rob Shearer. A novel approach to ontology classification. *Web Semantics: Science, Services and Agents on the World Wide Web*, 14(0), 2012.
23. Giorgos Stoilos, Bernardo Cuenca Grau, and Ian Horrocks. How incomplete is your semantic web reasoner? In *Proc. of AAAI 2010*. AAAI Press, 2010.
24. Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of IJCAR 2006*, pages 292–297, Berlin, Heidelberg, 2006. Springer.