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 logics for knowledge representation and reasoning [2]. They provide the logical foundations to OWL ontology languages [10], offer high expressivity but are decidable and often have low computational complexity. DLs are used to construct knowledge bases (KBs) that usually consist of a *TBox* and an *ABox*. A TBox describes the domain of interest in terms of *concepts* and *roles*, while an ABox stores information about known *instances* of concepts and their participation in roles.

Several systems have been developed in the last decade to reason about DL KBs. Naturally, classical reasoning tasks like TBox satisfiability and subsumption under a TBox have received most attention and many reasoners have been devoted to them (see e.g. FaCT++ [15], HermiT [14], ELK [9]). These and other mature systems geared towards TBox reasoning, have been rigorously tested and compared (e.g., *JustBench* [3]) using several real-life ontologies like GALEN and SNOMED-CT. Another important category are reasoners for *ontology-based data access* (*OBDA*) [12]. They are designed to answer queries over DL KBs in the presence of large data instances (see e.g. Ontop [13], Stardog). TBoxes in this setting are usually expressed in low complexity DLs, and are relatively small in size compared to the size of instance data. These features make reasoners for OBDA different from classical TBox reasoners.

It has been acknowledged in the DL community that judging the performance of OBDA reasoners and their underlying algorithms is limited due to the lack of publicly available benchmarks consisting of large amounts of real-life instance data. Until now, almost only the Lehigh University Benchmark (LUBM) [7] was applied to test OBDA systems, as it is tailored for query answering and provides a generator for different sizes of instance data. Another benchmarks are recently provided by Perez-Urbina et al. [11] and Imprialou et al. [8] with the focus on test query generation and the aim to compare the size and evaluation time of rewritten conjunctive queries. The mentioned benchmarks are well designed, but missing one or more of the following components. First, the generated data is often artificial and has a strong random component, which is for real-life benchmarks not realistic. Second, the generators and the ontologies are not easily modifiable and configurable because the users need programming skills. Third, a formalization and a meta-language for benchmark generation are missing (e.g., in LUBM), hence, desired properties can not be enforced (e.g., incompleteness) or repaired (e.g., inconsistency). In this paper, we consider publicly available geographic

datasets as a source of test data for OBDA systems. In particular, we describe how benchmarks for OBDA systems can be created in a simple and extensible way by employing a rule-based data transformation framework to extract instance data from *OpenStreetMap* (*OSM*) geospatial data.[3] We note that similarly to us the *Geographica* [5] suite uses OSM data for linked-data generation for testing RDF reasoners.

The OSM project aims to collaboratively create an open map of the world and has proven to be hugely successful and is constantly updated and extended. In addition, OSM data come with some (limited) amount of semantic information attached in the form of *tags*. OSM data describes maps in terms of points, ways (geometries), and more complex aggregate objects called *relations*. Each spatial object can be tagged, thus providing non-spatial information like the type of an object (e.g., hospitals), or the cuisine of a restaurant. OSM data seems to be an ideal source to obtain instance data for the following cases:

- Datasets of different sizes exist; e.g., OSM dumps for all major cities, countries, and continents are directly available.
- The data has an inherent (spatial) topology, which can be exploited to generate complex hierarchies by calculating the *spatial relations* between them (e.g., *contains*).
- Beside the topology, the (semantic) information of tags or the distance between objects can be exploited to obtain either nominal (e.g., smoking/non-smoking) or metric data (e.g., the distance to the next bus-stop).
- Depending on the location (urban versus rural), the density, separation, and compactness of the objects vary strongly. [4]

We proceed as follows. In Section 2, we provide a formal model for OSM data, and then in Section 3 describe a declarative rule-based language to extract DL ABoxes from an OSM map. We then discuss in Sections 4 and 5 the implementation of our approach together with a generated benchmark which is evaluated with Ontop.

2 Formalization of OSM

In this section we formally describe our model for OSM data, which we later employ 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*):

- *nodes*, which correspond to points with a geographic location;
- *geometries* (a.k.a. *ways*), which are given as sequences of nodes;
- *tuples* (a.k.a. *relations*), which are given as sequences of nodes, ways and tuples;
- *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.

To formalize OSM maps, which in practice are encoded in XML, 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:

[3] `http://www.openstreetmap.org`

[4] e.g., visible in `https://www.mapbox.com/osm-data-report/`

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.

For an example, assume we want to represent a bus route that, for the sake of simplicity, goes in a straight line from the point with coordinate $(0, 0)$ to the point with coordinate $(2, 0)$. In addition, the bus stops are at 3 locations with coordinates $(0, 0)$, $(1, 0)$ and $(2, 0)$. The names of the 3 stops are *S0*, *S1* and *S2*, respectively. This can be represented via the following map $\mathcal{M} = (\mathcal{D}, \mathcal{E}, \mathcal{L})$, where

- $\mathcal{D} = \{n_0, n_1, n_2, g, t\}$ with $\{n_0, n_1, n_2\} \subseteq \mathsf{M}_{\mathsf{nid}}$, $g \in \mathsf{M}_{\mathsf{gid}}$ and $t \in \mathsf{M}_{\mathsf{tid}}$,
- $\mathcal{E}(n_0) = (0, 0)$, $\mathcal{E}(n_1) = (1, 0)$, $\mathcal{E}(n_2) = (2, 0)$,
- $\mathcal{E}(g) = (n_0, n_2)$ and $\mathcal{E}(t) = (g, n_0, n_1, n_2)$,
- $\mathcal{L}(n_0) = \{S0\}$, $\mathcal{L}(n_1) = \{S1\}$ and $\mathcal{L}(n_2) = \{S2\}$.

The tuple (g, n_0, n_1, n_2) encodes the 3 stops n_0, n_1, n_2 tied to the route given by g.

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 needs not be given explicitly but can be computed from a map. In particular, we allow to enrich maps with arbitrary computable relations over M_{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 Tag_α 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 Inside, which captures the fact the location of a point x is inside a geometry y.

3 A Rule Language for Data Transformation

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* [1].

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*.

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. Elements of $\mathsf{M}_{\mathsf{id}} \cup \mathsf{D}_{\mathsf{vars}}$ are called *terms*. We call $R(\boldsymbol{t})$ and $\neg R(\boldsymbol{t})$

a *positive atom* and a *negative atom*, respectively. A *rule* r is an expression of the form $B_1, \ldots, B_n \rightarrow H$, where $B_1, \ldots, B_n$ are atoms (called *body atoms*) and H is a positive atom with a datalog relation symbol (called the *head atom*). We use $\mathsf{body}^+(r)$ and $\mathsf{body}^-(r)$ for the sets of positive and negative atoms of $\{B_1, \ldots, B_n\}$, respectively. We assume *datalog safety*, i.e. each variable of a rule occurs in some positive body atom. A rule r is *positive* if $\mathsf{body}^-(r) = \emptyset$. A *program* P is any finite set of rules. A rule or program is *ground* if it has no occurrences of variables. A program P is *positive* if all rules of P are positive. A program P is *stratified* if it can be partitioned into programs $P_1, \ldots, P_n$ such that:

(i) If $r \in P_i$ and $\neg R(\boldsymbol{t}) \in \mathsf{body}^-(r)$, then there is no $j \geq i$ such that P_j has a rule with R occurring in the head.

(ii) If $r \in P_i$ and $R(\boldsymbol{t}) \in \mathsf{body}^+(r)$, then there is no $j > i$ such that P_j has a rule with R occurring in the head.

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 variables in rules of P with identifiers occurring in $\mathcal{M}$ or P. We use a variant of the Gelfond-Lifschitz reduct [6] to get rid of maps 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 ordinary constants. We let $PM(\mathcal{M}, P)$ denote the *perfect model* of the program $P^{\mathcal{M}}$. See [1] 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 use $\mathsf{ABox}(\mathcal{M}, P)$ to denote the set of atoms obtained from $PM(\mathcal{M}, P)$ by restricting to atoms over concept and role names only.

Consider a toy program P with the following rules:

$$\mathsf{Point}(x), \mathsf{Tag}_{\mathsf{cinema}}(x), \mathsf{Geom}(y), \mathsf{Tag}_{\mathsf{city}}(y), \mathsf{Inside}(x, y) \rightarrow \mathsf{hasCinema}(y, x) \quad (1)$$

$$\mathsf{Geom}(x), \mathsf{Tag}_{\mathsf{garden}}(x), \neg\mathsf{Tag}_{\mathsf{private}}(x) \rightarrow \mathsf{RecreationalArea}(x) \quad (2)$$

The rule (1) collects in the role hasCinema the cinemas of a city. By rule (2) the concept RecreationalArea contains all parks that are not known to be private.

4 Implementation and Data Transformation Language

The abstract rule language defined in the previous section is implemented as a mapping language, which is inspired by the Ontop mapping language[5]. A mapping file contains the following two components. (1.) *Data sources declaration*: Currently, we support OWL/RDF, OSM data, relational database and even user defined functions by external programs. (2.) *Mapping axioms*: A mapping axiom is defined as a pair of source and target. The source is an arbitrary SQL query over spatial-extended RDBMSs and the target is a triple template that contains placeholders that reference column names mentioned in the source query.

[5] `http://ontop.inf.unibz.it`

Since simplicity and extensibility are main goals for the instance generation tool, the rule-based transformation language allows the user to define the data transformation in a declarative manner. In simple cases, it is not necessary to deal with implementation details, as the data is usually read from an RDBMS containing the OSM data sources and is mapped directly to the input files of the reasoner. However, in certain cases we have to use external functions for more advanced calculations, which are defined in the mapping axioms by external (Python) scripts. For example, the spatial relation *next* can be calculated in a script by measuring the distance between two objects. Further, mappings from OSM tags to the concepts/roles of an ontology can be determined by applying simple mapping scripts.

Implementation Details. The generation tool is implemented in Python 2.7 and resembles the *extract*, *transform*, and *load* (ETL) process of classical data transformation tools. [6] At the time of writing, the generation tool does not support the full expressive power of Datalog with stratified negation and processes only non-recursive Datalog. The extract component is implemented as an *iterator*, hence giving the possibility to stream the input and provide custom iterators. The load component is also extendable and supports the possibility to use data writers for different targets (e.g, text-files or RDBMSs). Currently, we provide the following extraction components:

- *Text-files*, which reads a file and converts every line into tuples depending on the field separator, e.g., comma-separated values (CSV) files can be read.
- *RDF-files*, rdf-files can be accessed using sparql queries, whereas the results are converted into tuples using the Python library *rdflib*.
- *Relational databases*, which for geospatial data is the most important input, since OSM databases are usually accessed via SQL and stored in the spatial-extended RDBMS PostGIS 2.12 (for PostgreSQL). [7]

5 Benchmarks and Evaluation

In this section we demonstrate the use of the transformation language to generate one example benchmark with instances from different cities in OSM. This should illustrate the feasibility and simplicity of our approach. All the test data are available online.[8]

Instances. Based on four cities of different size, [9] we generate four instances in Table 1. We extract concept assertions for all the amenities (e.g., restaurants), leisure areas (e.g., playgrounds), and shops. Then, we calculate the roles for the spatial relations *inside* between leisure areas/amenities and *next* between amenities/shops using a custom script (by the distance of 50m between 2 objects).

Ontology. The benchmarks are based on a custom ontology representing the geospatial domain. This ontology is used in the semantically enriched multi-Modal routing prototype *MyITS* [4]. It is a DL-Lite$_R$ ontology, which acts as a global data integration schema and is tailored to geospatial data sources and application specific sources (e.g., a

[6] `https://github.com/ghxiao/city-bench`
[7] `http://postgis.net/`
[8] `https://github.com/ghxiao/city-bench`
[9] `http://download.bbbike.org/osm/bbbike/`

Table 1. City Instances

City	objects	amenities	leisures	shops	spatial rel.
Cork	11002	674	1105	278	15217
Bern	220026	2825	12828	1539	171104
Vienna	487683	8587	25440	5259	489140
Berlin	666766	16199	29127	9791	716182

Table 2. Query Evaluation (in secs)

City	load	q_1	q_2	q_3
Cork	5.7	0.6	12.9	-
Bern	8.8	5.1	-	-
Vienna	15.0	5.5	-	-
Berlin	19.5		-	-

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.

Queries. After loading the data with the Ontop reasoner using "classic mode", we evaluate three queries. For instance, the query q_2 determines which amenities are inside a park and next to a shop; q_3 illustrates that arbitrary role chains can be used for queries simply by using the *next* relation:

$$\begin{aligned}
\mathtt{q}_1(y,z) &\leftarrow \mathtt{Amenity}(y), \mathtt{inside}(y,z), \mathtt{Leisure}(z)\\
\mathtt{q}_2(x,y,z) &\leftarrow \mathtt{Shop}(x), \mathtt{next}(x,y), \mathtt{Amenity}(y), \mathtt{inside}(y,z), \mathtt{Leisure}(z)\\
\mathtt{q}_3(x,y) &\leftarrow \mathtt{Amenity}(x), \mathtt{next}(x,z), \mathtt{next}(z,y), \mathtt{Shop}(y)
\end{aligned}$$

The results in Table 2 show that our benchmark is indeed challenging ('-' indicates a timeout of 10 mins). We also note that Ontop under "classic mode" is less optimal than that under "virtual mode" in which the (virtual) ABox instances are from the database via R2RML mappings. However, since we don't have such mappings and instances in the database, we cannot evaluate the "virtual mode" for the moment.

6 Conclusion and Outlook

In this paper, we presented an extensible framework for extracting instance data from OSM databases. We introduced a formalization of OSM, which combined with a Datalog-based rule language builds the formal underpinning. Then, we implemented an instance generation tool using a derivation of the rule language. The generation tools resembles the extract, transform, and load (ETL) proces. Finally, we demonstrated our approach by generating an example benchmark based on the OSM databases of four cities of different size. This benchmark is evaluated with the Ontop reasoner.

Future research is naturally directed to variants and extensions of the presented framework. We aim to extend the implementation to capture the full expressivity of the rule language, including negation and recursion. By adding new extraction components, we will extend our approach to generate R2RML mappings for data in the relational database. Further, we could generate TBox structures by (statistically) analyzing the OSM data, which might lead to complex ontologies. One needs also to understand the "texture" of OSM data better (e.g. graphs from the road network), so that more challenging benchmarks can be created. Based on our open repository, we aim to collect a wide range of benchmarks with the related transformations and data dumps. Finally, we will benchmark more reasoners on a wider range of generated benchmarks.

References

1. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. 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.
3. Samantha Bail, Bijan Parsia, and Ulrike Sattler. Justbench: A framework for OWL benchmarking. In Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Z. Pan, Ian Horrocks, and Birte Glimm, editors, *The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I*, volume 6496 of *Lecture Notes in Computer Science*, pages 32–47. Springer, 2010.
4. Thomas Eiter, Thomas Krennwallner, and Patrik Schneider. Lightweight spatial conjunctive query answering using keywords. In *The Semantic Web: Semantics and Big Data, 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. Proceedings*, pages 243–258, 2013.
5. George Garbis, Kostis Kyzirakos, and Manolis Koubarakis. Geographica: A benchmark for geospatial rdf stores. *CoRR*, abs/1305.5653, 2013.
6. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, pages 1070–1080, 1988.
7. Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics*, 3(2-3):158 – 182, 2005.
8. Martha Imprialou, Giorgos Stoilos, and Bernardo Cuenca Grau. Benchmarking ontology-based query rewriting systems. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*, 2012.
9. Yevgeny Kazakov, Markus Krötzsch, and František Simancík. Concurrent classification of el ontologies. In *Proceedings of the 10th International Conference on The Semantic Web - Volume Part I*, ISWC'11, pages 305–320, Berlin, Heidelberg, 2011. Springer-Verlag.
10. W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-overview/`.
11. Héctor Pérez-Urbina, Ian Horrocks, and Boris Motik. Efficient query answering for OWL 2. In *The Semantic Web - ISWC 2009, 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009. Proceedings*, pages 489–504, 2009.
12. Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Journal on data semantics x. chapter Linking Data to Ontologies, pages 133–173. Springer-Verlag, Berlin, Heidelberg, 2008.
13. Mariano Rodriguez-Muro, Roman Kontchakov, and Michael Zakharyaschev. Ontology-based data access: Ontop of databases. In *Proc. of ISWC 2013*, pages 558–573. Springer, 2013.
14. 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.
15. Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Proceedings of the Third International Joint Conference on Automated Reasoning*, IJCAR'06, pages 292–297, Berlin, Heidelberg, 2006. Springer-Verlag.