A Rule-based Framework for Creating Instance Data from *OpenStreetMap* ⋆

Thomas Eiter[1], Jeff Z. Pan[3], Patrik Schneider[1,4], Mantas Šimkus[1], and Guohui Xiao[2]

[1] Institute of Information Systems, Vienna University of Technology, Austria
[2] Faculty of Computer Science, Free University of Bozen-Bolzano, Italy
[3] University of Aberdeen, UK
[4] Vienna University of Economics and Business, Austria

Abstract. Reasoning engines for ontological and rule-based knowledge bases are becoming increasingly important in areas like the Semantic Web or information integration. It has been acknowledged however that judging the performance of such reasoners and their underlying algorithms is difficult due to the lack of publicly available datasets with large amounts of *(real-life) instance data*. In this paper we describe a framework and a toolbox for creating such datasets, which is based on extracting instances from the publicly available *OpenStreetMap* (*OSM*) geospatial database. To this end, we give a formalization of OSM and present a rule-based language to specify the rules to extract instance data from OSM data. The declarative nature of the approach in combination with external functions and parameters allows one to create several variants of the dataset via small modifications of the specification. We describe a highly flexible toolbox to extract instance data from a given OSM map and a given set of rules. We have employed our tools to create benchmarks that have already been fruitfully used in practice.

1 Introduction

Reasoning over ontological and rule-based knowledge bases (KBs) is receiving increasing attention. In particular *Description Logics* (DLs), which provide the logical foundations to OWL ontology languages, are a well-established family of decidable logics for knowledge representation and reasoning. They offer a range of expressivity well-aligned with computational complexity. Moreover, several systems have been developed in the last decade to reason over DL KBs, which usually consist of a *TBox* that describes the domain in terms of *concepts* and *roles*, an *ABox* that 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. A different category are reasoners for *ontology-based query answering* (*OQA*), which are designed to answer queries over DL KBs in the presence of large data instances (see e.g. Ontop [14], Pellet [19],and OWL-BGP [13]). TBoxes in this setting are usually expressed in low complexity DLs, and are relatively small in size compared to the instance data. These features make reasoners for OQA different from classical (TBox)

⋆ This work was supported by the Vienna Science and Technology Fund (WWTF) project ICT12-15, by the Austrian Science Fund (FWF) project P25207, and by the EU project Optique FP7-318338.

reasoners. The DL community is aware that judging the performance of OQA reasoners and their underlying algorithms is difficult due to the lack of publicly available benchmarks consisting of large amounts of *real-life instance data*. In particular, the popular Lehigh University Benchmark (LUBM) [10] only allows to generate random instance data, which provides only a limited insight into the performance of OQA systems.

In this paper, we consider publicly available geographic datasets as a source of test data for OQA systems and other types of reasoners. For the benchmark creation, we need a framework and a toolbox for extracting and enhancing instance data from *OpenStreetMap* (*OSM*) geospatial data.[5] The OSM project aims to collaboratively create an open map of the world. It has proven hugely successful and the map is constantly updated and extended. OSM data describes maps in terms of (possibly *tagged*) points, geometries, and more complex aggregate objects called *relations*. We believe the following features make OSM a good source to obtain instance data for reasoners: (a) Datasets of different sizes exist; e.g., OSM maps for all major cities, countries, and continents are directly available or can be easily generated. (b) Depending on the location (e.g., urban versus rural), the density, separation, and compactness of object location varies strongly.[6] (c) Spatial objects have an inherent structure of containment, bordering, and overlapping, which can be exploited to generate spatial relations (e.g., `contains`). (d) Spatial objects are usually tagged with semantic information like the type of an object (e.g., hospitals, smoking area), or the cuisine of a restaurant. In the DL world this information can be naturally represented in terms of concepts and roles.

Motivated by this, we present a rule-based framework and a toolbox to create benchmark instances from OSM datasets. Briefly, the main contributions are the following:

- We give a model-based formalization of OSM datasets which aims at abstracting from the currently employed but rather ad-hoc XML or object-relational representation. It allows one to view OSM maps as relational structures, possibly enriched with computable predicates like the spatial relations `contains` or `next`.
- Building on the above formalization, we present a rule-based language to extract information from OSM datasets (viewed as relational structures). In particular, a user can specify declarative rules which prescribe how to transform elements of an OSM map into ABox assertions. Different benchmark ABoxes can be created via small modifications of external functions, input parameter, and the rules of the specification.
- Our language is based on an extension of *Datalog*, which enjoys clear and well accepted semantics [1]. It has convenient features useful for benchmark generation.
- We have implemented a toolbox to create ABoxes from given input sources (e.g. an OSM database) and a given set of rules. The toolbox is highly configurable and can operate on various input/output sources, like RDF datasets, RDBMSs, and external functions. The input and result quality is measurable using descriptive statistics.
- By employing the above generation toolbox, we show on a proof-of-concept benchmark, how configurable and extensible the framework is. The toolbox has been already fruitfully been used for two benchmarks [5, 7].

Our framework and toolbox provide an attractive means to develop tailored benchmarks for evaluating query answering systems, to gain new insights about them.

[5] http://www.openstreetmap.org

[6] E.g., visible in https://www.mapbox.com/osm-data-report/

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*):[7]

- *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 a sequences of nodes, geometries, and tuples;
- *tags*, which are used to describe metadata about nodes, geometries, 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 $\mathcal{M} = (\mathcal{D}, \mathcal{E}, \mathcal{L})$ as follows.

1. $\mathcal{D} \subseteq \mathsf{M}_{\mathsf{id}}$ is a 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.

Example 1. 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 accessible to rules, we allow to enrich maps with arbitrary computable relations over M_{id}. In this way, we support incorporation of information that need not be given explicitly but can be computed from a map. 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 spatial relations like the binary relations "is closer than 100m", "inside

[7] For clarity, we rename the expressions used in OSM.

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 unary relations Point and Geom for points and geometries, and the binary relation Inside, where $\mathsf{Inside}(x, y)$ will mean that the point x is located inside the 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*. Elements of $\mathsf{M}_{\mathsf{id}} \cup \mathsf{D}_{\mathsf{vars}}$ are called *terms*.

An *atom* is an expression $R(\boldsymbol{t})$ or $not\, R(\boldsymbol{t})$, where R is a map or a Datalog relation symbol of arity n, and $\boldsymbol{t}$ is an n-tuple of terms. We call $R(\boldsymbol{t})$ and $not\, R(\boldsymbol{t})$ 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 in $\{B_1, \ldots, B_n\}$, respectively. We assume *(Datalog) safety*, i.e. each variable of r occurs in $\mathsf{body}^+(r)$. A *program* P is any finite set of rules. A rule or program is *ground* if it has no occurrences of variables. A rule r is *positive* if $\mathsf{body}^-(r) = \emptyset$. 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 $not\, R(\boldsymbol{t}) \in \mathsf{body}^-(r)$, 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 $\mathcal{M}$. Its ground program $\mathsf{ground}(P, \mathcal{M})$ can be obtained from P by replacing 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 [9] 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) Delete from the body of every rule r every map atom $not\, R(\boldsymbol{t})$ with $\boldsymbol{t} \notin R^{\mathcal{M}}$.

(b) Delete every rule r whose body contains a map atom $not\, R(\boldsymbol{t})$ 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 [1] for the construction of $PM(\mathcal{M}, P)$ by fix-point computation along the stratification. We are now 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 features of our rule language. The basic available service is to extract instances of concepts or roles by posing a standard conjunctive query over an OSM map. F.i., the following rule collects in the role `hasCinema` the cinemas of a city (we use sans-serif and typewriter font for map and Datalog relations, respectively):

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

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

$$\mathsf{Geom}(x), \mathsf{Tag}_{\mathsf{park}}(x), \mathit{not}\ \mathsf{Tag}_{\mathsf{private}}(x) \to \mathtt{RecreationalArea}(x)$$

Recursion is also useful and e.g., allows to deal with reachability, which appears naturally and in many forms in the context of geographic data. E.g. suppose we want to collect pairs b_1, b_2 of bus stops such that b_2 is reachable from b_1 using public buses. To this end, we can assume the availability of an external binary relation hasStop which relates bus routes and their stops, i.e. $\mathsf{hasStop}(x, y)$ is true in case x is a geometry identifier corresponding to a bus route and y is a point identifier corresponding to a bus stop in the route represented by x. Then the desired pairs of bus stops can be collected in the role ReachByBus using the following recursive rules:

$$\mathsf{hasStop}(x, y_1), \mathsf{hasStop}(x, y_2) \to \mathtt{ReachByBus}(y_1, y_2)$$
$$\mathtt{ReachByBus}(y_1, y_2), \mathtt{ReachByBus}(y_2, y_3) \to \mathtt{ReachByBus}(y_1, y_3).$$

Extending the Rule Language with ETL Features. We introduce a custom language for the benchmark generation, which *extends* the Datalog language of the previous paragraph with *extract*, *transform*, and *load* (ETL) features. The combined language consists of *Data Source Declarations*, *Mapping Axioms*, and *Datalog Rules*. Data Source Declarations contain general definitions like RDBMS connection strings. A mapping axiom defines 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*: Each pair/triple has a first column containing a constant, a second column referring to the data source declarations, and a third column, which is modified depending on the source, target, or transformation line, respectively.[8]

4 Benchmarking Framework

The rule language $\mathcal{L}$ of the previous section gives us the means to define the data transformations. We combine the language with an OSM database $\mathcal{S}$, an input ontology $\mathcal{O}$ (a.k.a TBox), a set $\mathcal{Q}$ of conjunctive or SPARQL queries, the generation parameters $\mathcal{P}$, and external functions $\mathcal{F}$. The benchmark framework is denoted as $\mathcal{F} = \langle \mathcal{S}, \mathcal{O}, \mathcal{Q}, \mathcal{L}, \mathcal{P}, \mathcal{F} \rangle$ and produces a set of ABox instances denoted as $\mathcal{A} = (A_1, \ldots, A_n)$. Note that $\mathcal{F}$ might modify $\mathcal{O}$ slightly.

Workflow. The workflow of creating a benchmark and evaluating the respective reasoners can be split into an *initial* and a *repeating* part. The initial part consists of the following elements: First, one has to choose the ontology $\mathcal{O}$ and decide which ontology language should be investigated. The *ontology statistics* gives a first impression on the *expressivity* of the language such as DL-Lite$_R$ [4] or $\mathcal{EL}$ [2]. Then, $\mathcal{O}$ has to be customized (e.g., remove axioms) and loaded to the system. For $\mathcal{Q}$, either handcrafted queries (related to a practical domain) have to built or synthetic queries have to be generated (e.g., Sygenia [11]). After the initial part, we are able to generate the instance data for the fixed $\mathcal{O}$ and $\mathcal{Q}$. This part of the workflow can be repeated until certain properties are reached. It has the following steps:

[8] See the detailed syntax, prerequisites, and tools on https://github.com/ghxiao/city-bench

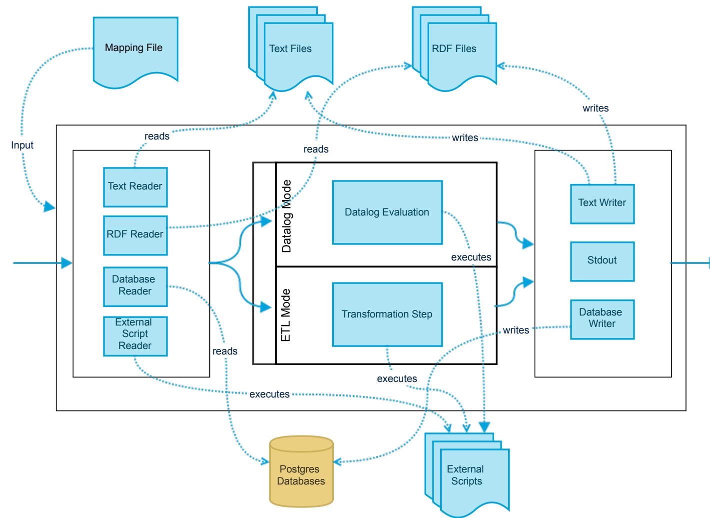

Fig. 1: System architecture, full lines are the control and dotted lines are the data flow

1. Creating an OSM database $\mathcal{S}$ with several instances, i.e., cities or countries;[8]
2. Applying *dataset statistics* to get a broad overview of the dataset, which leads to the selection of "interesting" datasets from $\mathcal{S}$;
3. Creating the rules of $\mathcal{L}$ to define the transformation for the instance generation and defining the parameters $\mathcal{P}$ and choosing the needed external functions of $\mathcal{F}$;
4. Calling the generation toolbox (see Section 5) and create the instances of $\mathcal{A}$;
5. Using *ABox statistics* to evaluate $\mathcal{A}'s$ quality, if not satisfactory, repeat from 3.;

Descriptive Statistics. For the benchmark creation, descriptive statistics serves two purposes. First, it gives a broad picture of the datasets, which is important to formulate the mapping rules. Second, we use the statistics to guide and fine tune the instance generation. I.e., for generating the `next` relation, different distances can be calculated leading to different sizes of $\mathcal{A}$. Descriptive statistics can be applied on three levels. On the *ontology level*, ontology metrics regarding $\mathcal{O}$ can be produced using *owl-toolkit*[9] to calculate the number of concepts, roles, and axioms (e.g., sub-concept). On the *dataset level*, we provide general information on the selected OSM database instance including the main elements *Points*, *Lines*, *Roads*, and *Polygons* and details about *frequent item sets* [3] of keys and tags. On the *ABox level* we provide the statistics of the generated instances in $\mathcal{A}$. For this, we count the assertion in $\mathcal{A}$ for every atomic concept or role name of $\mathcal{O}$. For future work, we aim to estimate the instances based on the *subsumption graph* for the entire concept/role hierarchy.

External Functions and Parameters. External functions bridge the gap between $\mathcal{L}$ and external computations. They allow us to develop dataset-specific customization and functionalities, where the results (atoms) are associated with predicates of $\mathcal{L}$. In Table 1, we list the currently available external functions. In addition, we provide the functions *deleteRandom* and *deleteByFilter* which drop instances randomly or filter out instances

[9] https://github.com/ghxiao/owl-toolkit

Table 1: Available External Functions

Name	Description	Predicate
transformOSM	generates from OSM tags atoms which represent concepts/roles of $\mathcal{O}$. It has to be customized to the signature of $\mathcal{O}$.	$\mathsf{Tag}_{\mathsf{Park}}(x)$
transformOSM-Random	instead of generating directly from OSM tags, it generates atoms according to a probabilities P assigned to a set O of OSM tags, e.g., $P(PublicPark)$=0.8 and $P(PrivatePark)$=0.2.	$\mathsf{Tag}_{P,O}(x)$
generateSpatial-Relation	generates the spatial relations `contains` or `next`, where a threshold parameter for the object distance can be given.	$\mathsf{next}_{10m}(x,y)$
generateStreet-Graph	generates the road/transport graph by creating instances for *edges* and *vertices* based on streets and corners between them.	$\mathsf{connected}(x,y)$ $\mathsf{Tag}_{\mathsf{corner}}(x)$

from $\mathcal{A}$. The parameters are the means to fine-tune the generation. They are often not directly observable, hence we need the statistics tool to get a better understanding of data sources. From recent literature [20, 16], we identified the following parameters for the instance generation:

- *ABox Size*: choice of the OSM instance (e.g., major cities or countries), but also by applying *deleteRandom* and *deleteByFilter*;
- *Degree of ABox Saturation:* can be indirectly manipulated by the use of Datalog rules in $\mathcal{L}$ to generate instances which otherwise would be deduced.
- *Distribution/Density of Nominal/Numeric Values:* input for *transformOSMRandom*;
- *Selectivity of Concept/Role Assertions:* input for *transformOSM* and *transformOSM-Random* and choice of OSM instances;
- *Graph Structure:* choice of OSM instance and selected graph (e.g., road vs. public transport network) for *generateStreetGraph*.

5 Implementation

We have developed for the framework a generation toolbox in Python 2.7. The main script generate.py is called as follows: generate.py –mappingFile mapping.txt

Modes. We provide two different modes with different evaluation strategies. The *Direct mode* is designed for simple bulk processing, where scalability and performance is crucial and complex calculations are moved to custom external scripts. We implemented the computation in a *data streaming*-based manner. The target components could be extended to custom triple stores like *Jena TDB*. The mapping axioms are evaluated in sequential order, hence dependencies between sources and targets are not considered. The *Datalog mode* extends the Direct mode and is designed for Datalog programs using the DLV system for evaluation.[10] The Datalog results are calculated in-memory and we follow a three-layered computation: 1. Previous ETL steps are evaluated to create the fact files for the EDB; 2. The defined Datalog programs (maintained in external files) are evaluated on the EDB files with the DLV module; 3. The (filtered) results (i.e., perfect models) are parsed and converted to tuples which then can be used by any target component. For now, we only handle a single model due to stratified Datalog.

[10] http://www.dlvsystem.com/dlv/

Table 2: Cities Dataset

City	$\#_{Points}$	$\#_{Lines}$	$\#_{Poly}$
Cork	6 068	14 378	4 934
Riga	19 172	43 042	67 708
Bern	68 831	83 351	151 195
Vienna	245 107	151 863	242 576
Berlin	236 114	218 664	430 652

Table 3: Road Network Instances

City	$\#_{Road}$	$\#_{Node}$	$\#_{connect}$	$\#_{Shop}$	$\#_{Bank}$	$\#_{opr}$	$\#_{next_{50}}$
Cork	6 476	45 459	46 013	278	36	36	750
Riga	6 620	35 107	37 007	827	102	102	1 408
Bern	17 995	130 849	134 670	1 539	120	120	10 285
Vienna	40 915	191 220	207 429	5 259	506	506	23 151
Berlin	46 320	204 342	226 554	9 791	588	588	81 911

Architecture. In Figure 1, we show the architecture of the framework. It naturally results from the two modes and the source and target components. The following source and target components are implemented. For *Text files*, we use the standard functions of Python for reading, writing, and evaluating regular expressions. For *RDF files*, which are accessed by SPARQL queries, we leverage the functions of the *rdflib* library. At present for *RDBMSs*, we only include access to the spatial-extended RDBMS PostGIS 2.12 (for PostgreSQL), which is the most common system for OSM.

External Functions and Statistics. Besides the main script, we implemented Python scripts for the external functions from Section 4 for processing the (OSM) data. The list of implemented functions (e.g., GenerateStreetGraph.py) is available online.[8] StatsOSM.py and StatsABox.py are the statistical scripts for estimating the structure of the ABox and the main OSM elements. It calculates the values for the most used field/tag combination. Additionally, we find the most frequent item sets using the *FP-Growth* algorithm.[11] For the ABox, we use the *rdflib* library to count and report the basic concept and role assertions.

6 Example Benchmark

In this section, we demonstrate how the framework can be applied to generate a proof-of-concept benchmark for OQA systems. All the mapping files, test datasets, and statistics are available online.[12]

OSM Dataset and Benchmark Ontology. There are different subsets of different sizes and structures available for OSM. For this example, we chose the cities of *Cork*, *Riga*, *Bern*, *Vienna*, and *Berlin*.[13] Using the dataset statistics module for *Vienna*, we observe for the field *Shop* 816 supermarkets, 453 hairdressers, 380 bakeries. For the field *Highway*, we have 29 392 residential, 4 087 secondary, and 3 973 primary streets.

The DL-Lite$_R$ [4] benchmark ontology is taken from the MyITS project [6]. It is tailored to geospatial and project specific data sources (e.g., a restaurant guide). The ontology is for OQA systems of average difficulty having only a few existential quantification on the right-hand side of the inclusion axioms. Due to its size and concept and role hierarchy depth, it poses a challenge regarding the rewritten query size.

[11] https://github.com/enaeseth/python-fp-growth
[12] https://github.com/ghxiao/city-bench/tree/master/benchmarks/rr2015
[13] Downloaded on the 1.10.14 from http://download.bbbike.org/osm/bbbike/

Creation of the Street Network. Besides creating the concept assertions for banks and shops, we extract the road network of the cities using the external function *generateStreetGraph* and encode the different roads into a single *road graph*. The road graph is represented by *nodes* which are asserted to the concept `Point` and by *edges* which are asserted to the role `connected`. By increasing the distances (e.g., from 50m to 100m) we saturate the `next` relation and generate more instances. Further, we use Datalog rules to calculate all paths (i.e. the transitive closure) of the street graph. The ABox statistics is shown in Table 2, the cities are of increasing size, starting with *Cork* (25 000 objects) and ending with *Berlin* (885 000 objects).

7 Related Work

In addition to the "de facto" standard benchmark LUBM [10] and extended LUBM [16] with randomly generated instance data with a fixed ontology, several other works deal with testing OQA systems. They can be divided along conceptional reasoning, query generation, mere datasets, synthetic and real-life instance generation. The benchmarks provided by [18] consist of a set of ontologies and handcrafted queries, tailored for testing query rewriting techniques. These benchmarks are a popular choice for comparing the sizes of generated queries. In [20], the authors have provided tools to generate ABoxes for estimating the incompleteness of a given OQA system. In a similar spirit is [11], which provides tools to automatically generate conjunctive queries for testing correctness of OQA systems. The same authors also provide a collection of benchmarks for evaluating query rewriting systems [17]. They did not offer any novel generation tool. The work of [15] for OBDA is designed based on real data from the Norwegian Petroleum Directorate FactPages. However, it is focused solely on a fixed DL-Lite$_R$ ontology and queries. None of the above benchmarks provide large amounts of real-life instance data and an extended framework including various parameters and external functions. Furthermore, most of the mentioned approaches do not consider an iterative generation process using statistics to guide the generation. In the area of Spatial Semantic Web systems, a couple of benchmarks have been proposed to test geospatial extensions of SPARQL including the spatial extension of LUBM in [12] and the *Geographica* benchmark [8]. They are pre-computed and Queries geared towards testing spatial reasoning capabilities of systems, but not designed with OQA in mind.

8 Conclusion and Outlook

We have presented a flexible framework for generating instance data from a geospatial database for OQA systems. In particular, we have introduced a formalization of OSM and a Datalog-based mapping language as the formal underpinning of the framework. Datalog offers powerful features such as recursion and negation for benchmark generation. We have implemented an instance generation tool supporting the main *Datalog* mode and a simple *Direct* (extract-transform-load) mode for several types of input sources. Finally, we have demonstrated our approach on a proof-of-concept benchmark.

Future research is naturally directed to variants and extensions of the presented 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. Furthermore, a tighter integration of the Datalog solver/engine and the source/target

components using dlvhex[14] is desired, which leads to a more efficient evaluation and more advanced capabilities (e.g., creating different ABoxes using all calculated answer sets). Then, we aim to apply our framework to generate benchmarks for an extensive study of different OQA reasoners with different underlying technologies. Finally, the instance assertion statistics could be extended to the full subsumption graph.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. F. Baader, S. Brand, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proc. of IJCAI 2005*, pages 364–369. Morgan-Kaufmann Publishers, 2005.
3. C. Borgelt. Frequent item set mining. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, 2(6):437–456, 2012.
4. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *dl-lite* family. *J. Autom. Reasoning*, 39(3):385–429, 2007.
5. T. Eiter, M. Fink, and D. Stepanova. Computing repairs for inconsistent dl-programs over $\mathcal{EL}$ ontologies. In *Proc. of JELIA 2014*, pages 426–441, 2014.
6. T. Eiter, T. Krennwallner, and P. Schneider. Lightweight spatial conjunctive query answering using keywords. In *Proc. of ESWC 2013*, pages 243–258, 2013.
7. T. Eiter, P. Schneider, M. Simkus, and G. Xiao. Using openstreetmap data to create benchmarks for description logic reasoners. In *Workshop proc. of ORE 2014*, July 2014.
8. George Garbis, Kostis Kyzirakos, and Manolis Koubarakis. Geographica: A benchmark for geospatial rdf stores (long version). In *Proc. of ISWC 2013*, pages 343–359. Springer, 2013.
9. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of ICLP/SLP 1988*, volume 88, pages 1070–1080, 1988.
10. Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics*, 3(2-3):158 – 182, 2005.
11. M. Imprialou, G. Stoilos, and B. Cuenca Grau. Benchmarking ontology-based query rewriting systems. In *Proc. of AAAI 2012*, 2012.
12. D. Kolas. A benchmark for spatial semantic web systems. In *4th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2008)*, October 2008.
13. I. Kollia and B. Glimm. Optimizing SPARQL query answering over OWL ontologies. *J. Artif. Intell. Res. (JAIR)*, 48:253–303, 2013.
14. R. Kontchakov, M. Rezk, M. Rodriguez-Muro, G. Xiao, and M. Zakharyaschev. Answering SPARQL queries over databases under OWL 2 QL entailment regime. In *Proc. of ISWC 2014*. Springer, 2014.
15. D. Lanti, M. Rezk, G. Xiao, and D. Calvanese. The NPD benchmark: Reality check for OBDA systems. In *Proc. of EDBT 2015*. ACM Press, 2015.
16. L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu. Towards a complete OWL ontology benchmark. In *Proc. of ESWC 2006*, pages 125–139. Springer.
17. J. Mora and O. Corcho. Towards a systematic benchmarking of ontology-based query rewriting systems. In *Proc. of ISWC 2013*, pages 376–391. Springer, 2013.
18. H. Pérez-Urbina, I. Horrocks, and B. Motik. Efficient query answering for OWL 2. In *Proc. of ISWC 2009*, pages 489–504, 2009.
19. 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.
20. G. Stoilos, B. Cuenca Grau, and I. Horrocks. How incomplete is your semantic web reasoner? In *Proc. of AAAI 2010*. AAAI Press, 2010.

[14] http://www.kr.tuwien.ac.at/research/systems/dlvhex/