Datalog revisited for reasoning in Linked Data [⋆]

Marie-Christine Rousset[1,2], Manuel Atencia[1], Jerome David[1], Fabrice Jouanot[1],
Olivier Palombi[3,4], and Federico Ulliana[5]

[1] Université Grenoble Alpes, Grenoble INP, CNRS, Inria, LIG, F-38000, Grenoble, France
[2] Institut universitaire de France, 75005, Paris, France
[3] Université Grenoble Alpes, Grenoble INP, CNRS, Inria, LJK, F-38000, Grenoble, France
[4] Université Grenoble Alpes, LADAF, CHU Grenoble , F-38000 Grenoble, France
[5] Université de Montpellier, CNRS, Inria, LIRMM, F-34000 Montpellier

Abstract. Linked Data provides access to huge, continuously growing amounts of open data and ontologies in RDF format that describe entities, links and properties on those entities. Equipping Linked Data with inference paves the way to make the Semantic Web a reality. In this survey, we describe a unifying framework for RDF ontologies and databases that we call deductive RDF triplestores. It consists in equipping RDF triplestores with Datalog inference rules. This rule language allows to capture in a uniform manner OWL constraints that are useful in practice, such as property transtivity or symmetry, but also domain-specific rules with practical relevance for users in many domains of interest. The expressivity and the genericity of this framework is illustrated for modeling Linked Data applications and for developing inference algorithms. In particular, we show how it allows to model the problem of data linkage in Linked Data as a reasoning problem on possibly decentralized data. We also explain how it makes possible to efficiently extract expressive modules from Semantic Web ontologies and databases with formal guarantees, whilst effectively controlling their succinctness. Experiments conducted on real-world datasets have demonstrated the feasibility of this approach and its usefulness in practice for data integration and information extraction.

1 Introduction

Thanks to the RDF data model, the Semantic Web has become a reality with the rapid development of Linked Data. Linked Data provides access to huge, continuously growing amounts of open data in RDF format that describe properties and links on entities referenced by so-called Uniform Resource Identifiers (URIs).

RDFS and OWL languages [5] allow to express a lot of useful logical constraints on top of RDF datasets, and existing Semantic Web tools implement inference algorithms to exploit them. In particular, the Jena environment [6] includes a rule-based reasoner that implements the RETE algorithm [21]. When the inference mode is launched, the *saturated* dataset is computed, which is the set of RDF facts that can be logically inferred

[⋆] This work has been partially supported by the ANR projects Pagoda (12-JS02-007-01) and Qualinca (12-CORD-012), the joint NSFC-ANR Lindicle project (12-IS01-0002), and LabEx PERSYVAL-Lab (11-LABX-0025-01).

[6] https://jena.apache.org/documentation/inference/

from the input RDF dataset and a given set of rules. The saturation process is guaranteed to terminate if the rules are safe, i.e., if the variables appearing in the conclusion of each rule also appear in its condition part.

Safe rules (also called Datalog rules) on top of RDF facts capture in a uniform way most of the OWL constraints useful in practice, as well as mappings across different datasets, and also domain knowledge provided by experts, while guaranteeing a polynomial data complexity of reasoning and query answering [2].

In the setting of a unifying framework that we have called *deductive RDF triple-stores*, we have followed a rule-based approach to address several problems raised by exploiting semantic web knowledge bases. For this, we have extended and adapted forward-chaining and backward-chaining algorithms intially developed for Datalog deductive databases.

This survey is structured as follows. In Section 2, we first recall the ingredients of Linked Data and we define what we call a deductive RDF dataset to capture several ontological constraints expressing data semantics. In Section 3, we survey the rule-based data linkage approach that we have developed in the context of Linked Data based on reasoning for inferring differentFrom and sameAs facts. In Section 4, we summarize our approach for extracting *bounded-level* modules from RDF knowledge bases. Finally, in Section 5, we illustrate our methodology for rule-based integration of heterogeneous data and ontologies through several applications related to Medicine. Finally, we conclude in Section 6.

2 Datalog rules on top of RDF datasets

We first recall the ingredients of Linked Data and then we define what we call a deductive RDF dataset to capture several ontological constraints expressing data semantics.

2.1 RDF datasets in Linked Data

An RDF dataset in Linked Data is defined by a URL u and a set F of RDF facts that are accessible as URL through a query endpoint. We will denote by $ds(u)$ the set F of RDF facts that can be queried at the URL u.

An RDF fact is a triple $t = (s, p, o)$ where the subject s is either a URI or a blank node, the predicate p is a URI, and the object o may be either a URI, a blank node or a literal. We will denote the vocabulary used in $ds(u)$ by $voc(u)$, i.e., the names of predicates used to declare triples in the dataset accessible at the URL u.

2.2 Queries over RDF datasets in Linked Data

Queries over Linked Data are SPARQL **conjunctive queries** entered through a given query endpoint accessible at a given URL. In this paper, we use a simplified notation for SPARQL queries, and, without loss of generality, we consider that all variables are distinguished.

A query $q(u)$ asked to an RDF dataset identified by (and accessible at) the URL u is a conjunction of **triple patterns** denoted by $TP_1(v_1), \ldots, TP_k(v_k)$ where each

triple pattern $TP_i(v_i)$ is a triple (s^v, p^v, o^v) in which the subject s^v, the predicate p^v, or the object o^v can be variables: v_i is the set of variables appearing in the triple pattern. Variables are denoted by strings starting by '?'. $TP_i(v_i)$ is a **ground** triple pattern if its set of variables v_i is empty (denoted by $TP_i()$). A ground triple pattern corresponds to a RDF fact. A **boolean query** is a conjunction of ground triple patterns.
The evaluation of a query $q(u) : TP_1(v_1), \ldots, TP_k(v_k)$ over the dataset $ds(u)$ consists in finding substitutions θ assigning the variables in $\bigcup_{i \in [1..k]} v_i$ to constants (i.e., identifiers or literals) such that $TP_1(\theta.v_1), \ldots, TP_k(\theta.v_k)$ are RDF facts in the dataset.

The corresponding answer is equally defined as the tuple of constants assigned by θ to the variables or as the set of corresponding RDF facts $TP_1(\theta.v_1), \ldots, TP_k(\theta.v_k)$ that will be denoted by $\theta.q(u)$. In the remainder of the paper, we will adopt the latter definition. The answer set of the query $q(u)$ against the dataset $ds(u) = F$ is thus defined as:

$$Answer(q(u), F) = \bigcup_{\{\theta | \theta.q(u) \subseteq F\}} \{\theta.q(u)\}$$

For a boolean query $q(u)$, either the answer set is not empty and we will say that the query is evaluated to **true**, or it is empty and we will say that it evaluated to **false**.

For a query $q(u)$ to have a chance to get an answer when evaluated over the dataset $ds(u)$, it must be **compatible with** the vocabulary used in this dataset, i.e., (a) the predicates appearing in the triple patterns of $q(u)$ must belong to the set $voc(u)$ of predicates known to occur in $ds(u)$, (b) the URIs appearing as constants in the triple patterns of $q(u)$ must have u as prefix.

In accordance with SPARQL queries allowing different FROM operators, a conjunctive query can in fact specify several entry points $u_1, \ldots, u_n$ of datasets over which the query has to be evaluated. We will denote such a query $q(u_1, \ldots, u_n)$. The above definitions of answers and compatibility can be generalized appropriately by replacing the dataset $ds(u)$ by the union $\bigcup_{i \in [1..n]} ds(u_i)$ of the specified datasets.

2.3 Deductive RDF datasets

In order to capture in a uniform way semantic constraints that can be declared on top of a given RDF dataset, but also possibly mappings between local predicates and external predicates within the vocabulary of other datasets, and domain knowledge provided by domain experts, we consider that RDF datasets can be enriched with Datalog rules. The Datalog rules that we consider are of the form: $Cond_r \rightarrow Conc_r$, in which the condition $Cond_r$ is a conjunction of triple patterns (i.e., a conjunctive query) and the conclusion $Conc_r$ is a triple pattern. We consider **safe** rules, i.e., rules such that all the variables in the conclusion are also in the condition. Datalog rules on top of RDFS facts capture most of the OWL constraints used in practice, while guaranteeing a polynomial data complexity for reasoning and query answering.

A deductive RDF dataset $dds(u)$ accessible at the URL u is thus a local knowledge base (F, R) made of a set of RDF facts F and a set R of rules. The application of rules allows to infer new facts that are logically entailed from $F \cup R$. A rule r can be applied to F if there exists a substitution θ such that $\theta.Cond_r \subseteq F$ and the result of the rule application is $F \cup \{\theta.Conc_r\}$. These new facts can in turn trigger rules and infer additional facts. This is formalized in the following definition of the standard semantics of a knowledge base $F \cup R$ composed of a finite set of facts F and a finite set of rules R, based on the least fixed point of immediate consequence operator T_R.

Definition 1 (Datalog semantics).
- $(F, R) \vdash_1 f$ *iff there exists a rule* $TP_1(v_1) \wedge \ldots \wedge TP_k(v_k) \rightarrow TP(v)$ *is in* R *and there exists a mapping* θ *from its variables to constants such that* $f = \theta.TP(v)$ *and* $\theta.TP_i(v_i) \in F$ *for every* $i \in [1..k]$.
- $(F, R) \vdash f$ *iff there exists* i *such that* $f \in T_R(F_i)$ *where* $F_0 = F$*, and for every* $i \geq 0$*,* $F_{i+1} = T_R(F_i) = F_i \cup \{f | F_i, R \vdash_1 f\}$

For a finite set of facts F and a finite set of safe rules R, there exists a unique least fixed point F_n (denoted by $SAT(F, R)$) such that for every $k \geq n$ $F_k = T_R(F_n)$, i.e., there exists a step in the iterative application of the immediate consequence operator for which no new fact is inferred. Several forward-chaining algorithms exist to compute $SAT(F, R)$, in particular the semi-naive bottom-up evaluation in Datalog [2], and the RETE algorithm [21] that is implemented in many rule-based reasoners, including in Semantic Web tools such as Jena [7].

Query evaluation over a deductive dataset
The evaluation of a query $q(u) : TP_1(v_1), \ldots, TP_k(v_k)$ over a deductive dataset $dds(u)$ consists in finding substitutions θ such that the facts $TP_1(\theta.v_1), \ldots, TP_k(\theta.v_k)$ can be **inferred** from the deductive dataset, or equivalently belong to the result $SAT(F, R)$ of the facts that can be inferred from F and R:

$$Answer(q(u), (F, R)) = Answer(q(u), SAT(F, R))$$

Thus, a boolean query $q(u)$ is evaluated to **true** if and only if $q(u) \in SAT(F, R)$, i.e., if and only if $(F, R) \vdash q(u)$, where $\vdash$ is the standard notation for logical inference.

Within the vocabulary of a deductive dataset, we distinguish the extensional predicates (EDB predicates for short) that appear in the triplets of the dataset F, from the intentional predicates (IDB predicates) that appear in conclusion of some rules in R. Like in deductive databases, and without loss of generality (i.e., by possibly renaming predicates and adding rules), we suppose that these two sets are disjoint. We will denote ODB predicates the *external* predicates (i.e., defined in a different namespace than the considered deductive dataset) that possibly appear in the dataset or in the rules. These predicates are the core of Linked Data in which a good practice is to re-use existing reference vocabularies. We suppose (again, without loss of generality) that the set of ODB predicates is disjoint from the set of IDB predicates (but not necessarily from the set of EDB predicates).

[7] https://jena.apache.org/documentation/inference/

3 Rule-based data linkage

Data linkage consists in deciding whether two URIs refer to the same real-world entity. This is a crucial task in Linked Data. In particular, it is very important to correctly decide whether two URIs refer to the same real-world entity for developing innovative applications on top of Linked Data, that exploit the cross-referencing of data [26, 20]. This task is often referred to as data interlinking, and is also known as record linkage and entity resolution, and it has been widely studied for the case of relational data [16]. As regards to Linked Data, data linkage is especially challenging since (1) tools need to scale well with large amounts of data, (2) data is frequently described using heterogeneous vocabularies (ontologies), and (3) tools need to deal with data which is inherently incomplete, and very often noisy.

In the context of Linked Data and RDF data, different approaches to data linkage have been proposed. Most of them are based on numerical methods that use linkage rules to compare property values of resources, using similarity measures to handle noisy data. They conclude weighted sameAs links, from which the links with higher weights are expected (but never guaranteed) to be correct [48, 34]. These approaches suffer from two weaknesses. First, rules cannot be chained, as they are thought to be applied only once; and second, weights are combined in a non-formal manner, since there is no formal semantics that captures the combination of weights.

In contrast, like a few other works [40, 31], we promote a rule-based approach equipped with full reasoning.

First, we have investigated a logical approach that exploits uniqueness constraints (such as inverse functional properties and keys) and other schema constraints, domain knowledge and alignments between different vocabularies which can be modelled as logical rules. This enables to infer all *certain* sameAs and differentFrom facts that are logically entailed from a given set of domain constraints and input facts. Our main contribution is a novel algorithm, called Import-by-Query, that enables the scalable deployment of such an approach in the decentralized setting of Linked Data. The main challenge is to identify the data, possibly distributed over several datasets, useful for inferring owl:sameAs and owl:differentFrom facts of interest. Compared to the approach reported in [31], relying on a global import obtained by a breadth-first crawl of the Linked Data cloud, we perform a selective import while guaranteeing completeness for the inference of the targeted owl:sameAs and owl:differentFrom facts. For doing so, the Import-by-Query algorithm that we have designed alternates steps of sub-query rewriting and of tailored querying of the Linked Data cloud to import data as specific as possible to infer owl:sameAs and owl:differentFrom facts. It is an extension of the well-known query-subquery algorithm for answering Datalog queries over deductive databases. Experiments conducted on a real-world dataset have demonstrated the feasibility of this approach and its usefulness in practice for data linkage and disambiguation.

We summarize this logical approach in Section 3.1.

Logical approaches applying only certain rules over clean and complete data guarantee to provide sound results, i.e., a 100% precision. However, the recall may be low because in Linked Data, data is inherently incomplete and possibly noisy. Input facts may be missing to trigger rules, either because some values for properties involved in rules conditions are absent for some URIs, or because some of these values are noisy with some misspelling that prevents some conditions to be satisfied. In addition, rules may be missing to infer sameAs facts with certainty, although some strong evidence could be obtained from the combination of soft constraints. In order to handle this, we have modeled the general data linkage problem as a reasoning problem with uncertainty. We have introduced a probabilistic framework for modelling and reasoning over uncertain RDF facts and rules that is based on the semantics of probabilistic Datalog, and we have designed an algorithm, ProbFR, based on this framework. This approach is summarized in Section 3.2

3.1 Logical approach for data linkage [4]

Illustrative Scenario

We describe here a simplified scenario inspired by the task of disambiguation of named entities in a large real-world RDF documentary catalog produced by the French National Audiovisual Institute (INA), and that we have used in our experiments.

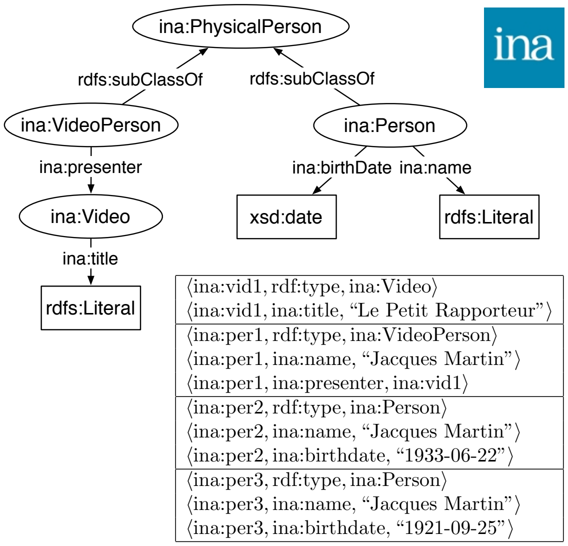
Figure 1 shows an extract of the INA vocabulary and a sample of RDF triples from the INA dataset.[8] Any person entity is an instance of the class ina:PhysicalPerson, which has two subclasses: ina:Person and ina:VideoPerson. The class ina:Person is used for representing French personalities while ina:VideoPerson is used for identifying person entities that play a role in a video. INA experts want to disambiguate individuals within ina:Person, and link these individuals to the ones of ina:VideoPerson.

Three homonymous persons are described in Figure 1, all named “Jacques Martin”: ina:per1, ina:per2 and ina:per3. It is unknown if these entities represent the same or different persons, but some additional information is given: ina:per1 is known to be the presenter of a program recorded in the video ina:vid1 whose title is “Le Petit Rapporteur”, whereas ina:per2 and ina:per3 have dates of birth “1933-06-22” and “1921-09-25”, respectively.

Our approach to disambiguating the person entities ina:per1, ina:per2 and ina:per3 consists in exploiting domain knowledge and constraints, as well as general properties of owl:sameAs and owl:differentFrom, all this knowledge being expressed in a uniform way by rules. Table 1 shows rules which, for the purpose of this simplified scenario, we can assume they have been validated by INA experts. R1-R3 are domain-specific rules. R1 expresses that ina:birthdate is functional. This rule can be used to infer that ina:per2 and ina:per3 are different because they have different dates of birth. R2 expresses that ina:name and ina:birthdate form a key (within the INA dataset), and R3 the fact that two persons who have the same name and presented programs recorded in videos with the same title must be the same. R2 and R3 indeed could be useful for deciding if ina:per1 refers to the same person as ina:per2 or ina:per3, but some information is missing: the date of birth of ina:per1 is not known, or whether ina:per2 or ina:per3 are presenters and of which programs.

The above missing information can be completed thanks to external data coming from DBpedia. In Figure 2, we show DBpedia facts describing the DBpedia person entity db:per1, and an extract of the DBpedia vocabulary. Rules R4 and R5 in Table 1

[8] We have slightly modified the INA vocabulary (e.g. translating French terms into English terms) for the sake of readability.



⟨ina:vid1, rdf:type, ina:Video⟩
⟨ina:vid1, ina:title, "Le Petit Rapporteur"⟩

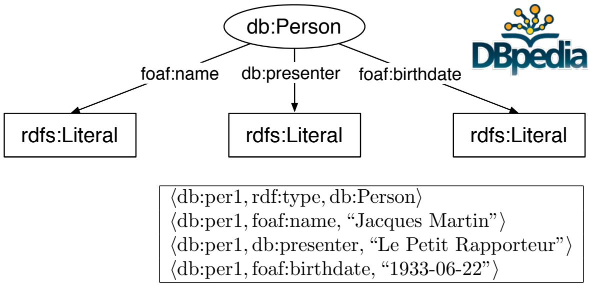
⟨ina:per1, rdf:type, ina:VideoPerson⟩
⟨ina:per1, ina:name, "Jacques Martin"⟩
⟨ina:per1, ina:presenter, ina:vid1⟩

⟨ina:per2, rdf:type, ina:Person⟩
⟨ina:per2, ina:name, "Jacques Martin"⟩
⟨ina:per2, ina:birthdate, "1933-06-22"⟩

⟨ina:per3, rdf:type, ina:Person⟩
⟨ina:per3, ina:name, "Jacques Martin"⟩
⟨ina:per3, ina:birthdate, "1921-09-25"⟩

Fig. 1. An extract of INA vocabulary and RDF facts.

translate mappings from the INA and DBpedia vocabularies. Specifically, these mappings state that ina:name and ina:birthdate are equivalent to foaf:name and foaf:birthdate, respectively, and that the composition of ina:presenter and ina:title is equivalent to db:presenter. Let us assume that rules R4 and R5 have been validated by INA experts too. With these rules it can be inferred that db:per1 is the same as ina:per1 because they have the same name and they have presented a program with the same title; and that db:per1 is the same as ina:per2 since they have the same name and birthdate. Therefore, by transitivity of same-as (rule R6 in Table 1), it can be inferred that ina:per1 is the same as ina:per2, and, since ina:per2 is different from ina:per3 then (due to R7) ina:per1 is different from ina:per3 too.



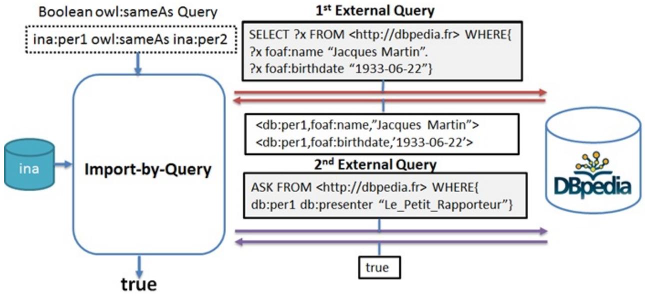
⟨db:per1, rdf:type, db:Person⟩
⟨db:per1, foaf:name, "Jacques Martin"⟩
⟨db:per1, db:presenter, "Le Petit Rapporteur"⟩
⟨db:per1, foaf:birthdate, "1933-06-22"⟩

Fig. 2. An extract of DBpedia vocabulary and RDF facts.

R1 : $(?x1, \text{ina:birthdate}, ?b1), (?x2, \text{ina:birthdate}, ?b2), (?b1, \text{notEqualTo}, ?b2) \rightarrow (?x1, \text{owl:differentFrom}, ?x2)$
R2 : $(?x1, \text{ina:name}, ?n), (?x2, \text{ina:name}, ?n), (?x2, \text{ina:birthdate}, ?b), (?x1, \text{ina:birthdate}, ?b)$
$\rightarrow (?x1, \text{owl:sameAs}, ?x2)$
R3 : $(?x1, \text{ina:name}, ?n), (?x2, \text{ina:name}, ?n), (?x1, \text{ina:presenter}, ?v1), (?x2, \text{ina:presenter}, ?v2), (?v1, \text{ina:title}, ?t),$
$(?v2, \text{ina:title}, ?t) \rightarrow (?x1, \text{owl:sameAs}, ?x2)$
R4 : $(?x1, \text{ina:name}, ?n), (?x2, \text{foaf:name}, ?n), (?x1, \text{ina:presenter}, ?v), (?v, \text{ina:title}, ?t), (?x2, \text{db:presenter}, ?t)$
$\rightarrow (?x1, \text{owl:sameAs}, ?x2)$
R5 : $(?x1, \text{ina:name}, ?n), (?x2, \text{foaf:name}, ?n), (?x1, \text{ina:birthdate}, ?b), (?x2, \text{foaf:birthdate}, ?b)$
$\rightarrow (?x1, \text{owl:sameAs}, ?x2)$
R6 : $(?x1, \text{owl:sameAs}, ?x2), (?x2, \text{owl:sameAs}, ?x3) \rightarrow (?x1, \text{owl:sameAs}, ?x3)$
R7 : $(?x1, \text{owl:sameAs}, ?x2), (?x2, \text{owl:differentFrom}, ?x3) \rightarrow (?x1, \text{owl:differentFrom}, ?x3)$
R8 : $(?x1, \text{ina:name}, ?n1), (?x2, \text{foaf:name}, ?n2), (?n1, \text{built-in:name-similar}, ?n2), (?x1, \text{ina:birthdate}, ?b),$
$(?x2, \text{foaf:birthdate}, ?b) \rightarrow (?x1, \text{owl:sameAs}, ?x2)$

Table 1. Rules in the INA illustrative scenario.

To avoid downloading the complete DBpedia, and, more generally, the whole Linked Open Data (something that is not practical), our import-by-query approach generates, for each targeted owl:sameAs fact, a sequence of external sub-queries as specific as possible to obtain just the missing facts. The external sub-queries generated by our algorithm for the particular query $(\text{ina:per1}, \text{owl:sameAs}, \text{ina:per2})$ in our example are shown in Figure 3.



```
SELECT ?x FROM <http://dbpedia.fr> WHERE{
?x foaf:name "Jacques Martin".
?x foaf:birthdate "1933-06-22"}
```

```
<db:per1,foaf:name,"Jacques Martin">
<db:per1,foaf:birthdate,'1933-06-22'>
```

```
ASK FROM <http://dbpedia.fr> WHERE{
db:per1 db:presenter "Le_Petit_Rapporteur"}
```

Fig. 3. The resultant external sub-queries submitted to DBpedia and their returned answers.

Problem statement

Given a deductive dataset $dds(u) = (F, R)$, and a boolean query $q(u)$ the local evaluation of which gives an empty answer set (i.e., $(F, R) \not\vdash q(u)$), we aim to construct a set of external queries $q_1(u_1), \ldots, q_k(u_k)$ for which we can guarantee that the subsets of external facts resulting from their evaluation over the (possibly huge) external datasets are sufficient to answer the initial query. More formally:

$$(F \cup \bigcup_{i \in [1..k]} Answer(q_i(u_i), ds(u_i)), R) \ \vdash \ q(u)$$

$$\text{iff } (F \cup \bigcup_{i \in [1..k]} ds(u_i), R) \ \vdash \ q(u)$$

The more specific the external queries are, the less external facts have to be added and stored to the local dataset and therefore the more interesting a proposed approach is to solve this problem.

The iterative Import-by-Query Algorithm

We now describe the algorithm that we have designed and implemented for solving the problem stated above.
Given an input boolean same-as query q, a deductive dataset (F, R), and a set $\bar{u}$ of query entry points to external datasets, Import-by-Query iteratively alternates steps of sub-query rewriting based on backward chaining and of external query evaluation.

Each sub-query rewriting step is realized by an adaptation of the *Query-Subquery* algorithm [47, 2] that is a set-oriented memoing backward chaining method [29] used in deductive databases for evaluating Datalog programs. This results in the *Query-External-Subquery* (*QESQ* for short) algorithm. For space limitation, here we just explain its main principles, compared to *Query-Subquery*, when applied to a list SG of subgoals. QESQ handles the subgoals built on EDB or IDB predicates exactly like *Query-Subquery*, i.e., iteratively removes subgoals built on EDB predicates if they can be matched with local facts, propagates the corresponding substitutions to the remaining subgoals, replaces a subgoal g built on an IDB predicate by the list of partially instantiated conditions of a rule whose conclusion can be matched to g. As for the subgoals on ODB predicates, they are handled by QESQ before the subgoals on IDB predicates, and once all the subgoals built on EDB predicates have been removed, and after the corresponding substitutions are applied to the remaining subgoals in the list. These ODB subgoals are conjuncted to obtain an external query q_{ext}, the compatibility of which must be checked w.r.t. $\bar{u}$ to be considered further. QESQ then treats the remaining list SG_{idb} of subgoals on IDB predicates just as *Query-External-Subquery*, i.e., triggers the recursive call $QESQ(SG_{idb})$. It will return as output either **true** or **false** (if it has enough local information to infer a result to the input boolean query), or a set of external queries that, if compatible with the vocabulary of the given external datasets, are then conjuncted with q_{ext} to constitute the output returned by QESQ(SG). As a result QESQ $(\{q\})$ succeeds in handling locally the goal q using F and R just like *Query-Subquery* and then the process is stopped and the result returned by Import-by-Query is **true** or **false** accordingly, or it produces a set $\{q_1(\bar{u}_1), \ldots, q_k(\bar{u}_k)\}$ of external queries the evaluation of which is likely to bring missing facts to F for proving the goal q using R. If this set is empty, the process is stopped and the result returned by Import-by-Query is **false**.

Each evaluation step simply consists in choosing one of the external query $q_i(\bar{u}_i)$ produced by the sub-query rewriting step and to submit it to Linked Data through the specified query entry points. The result is either an empty set (negative result) or a set of external facts (positive result) that can be added to the current local dataset. In both cases, the result is memorized in an associated answer table for the sub-query $q_i(\bar{u}_i)$ that will be thus marked as an already processed subgoal for which the (positive or negative) result is known and can be directly exploited later on. If the result is positive, a new iteration of Import-by-Query is started on the same input except for the set of facts F that is enriched with the facts obtained as the result of the evaluation of the external query $q_i(\bar{u}_i)$. If the result is negative, another external query $q_j(\bar{u}_j)$ in the set produced by the current call to QESQ is evaluated. If the evaluation of all the external queries in the set returns 'false', then the process is stopped and the result returned by Import-by-Query on q is **false**.

The termination of the Import-by-Query algorithm relies on the termination of QESQ, which is guaranteed by the same memoing technique as *Query-Subquery* (i.e., by handling goal and answer tables for each ODB and IDB predicate). The soundness and completeness of the Import-by-Query algorithm results from the soundness and completeness of *Query-Subquery* [47] and from the observation that the result produced by *Query-Subquery*, if applied to the same input in which the ODB predicates are just considered as additional EDB predicates, would be the same as the one produced by Import-by-Query. The reason is that the only difference of Import-by-Query is to replace successive matching of atomic goals against the facts by matching *all at once* the atomic goals composing the external queries produced by QESQ. This does not impact the global boolean result of the sequence of goal matching.

Combining forward and backward chaining

Like any backward chaining method, Import-by-Query (and its main component QESQ) re-starts from scratch for each new goal it tries to solve, even if the facts and the rules remain unchanged. The intermediate subgoals generated and handled by QESQ can be simplified if the input rules are replaced by their (partial) instantiations obtained by the propagation of the facts into (the conditions of) the rules.

Fact propagation is a forward chaining method used in inference engines such as RETE [21] for rule-based systems. It avoids redundant evaluation of same conditions appearing in several rules by memorizing, for each fact f, which condition it satisfies in which rule (possibly already partially instantiated by facts previously propagated), and the corresponding variable substitution that is then applied to all the remaining conditions of the rules.

In our setting, we perform fact propagation as a pre-processing step of the Import-by-Query algorithm, by computing at the same time the set $SAT(F, R)$ of facts that can be inferred locally, and the set $PI(F, R)$ of partial instantiations of the rules in R. This forward reasoning step can be summarized as follows, where $SAT(F, R)$ is initialized as F and $PI(F, R)$ is initialized as R:

```
– FOR each $f$ in $SAT(F, R)$
      FOR each rule $Cond_r \rightarrow Conc_r$ in $PI(F, R)$ having a condition $c$ that can be
      matched with $f$, i.e., there exists $\theta$ such that $\theta.c = f$
        ∗ IF $c$ is the only condition in $Cond_r$ THEN add $\theta.Conc_r$ to $SAT(F, R)$
        ∗ ELSE add to $PI(F, R)$ the rule obtained from $\theta.Cond_r \rightarrow \theta.Conc_r$ by
          removing the condition $\theta.c$ (that is satisfied by the fact $f$).
– Remove from $PI(F, R)$ those rules whose condition contains EDB predicates that
  are not ODB predicates (and thus cannot be satisfied by local facts).
– RETURN $(SAT(F, R), PI(F, R))$
```

Each partially instantiated rule r_i returned in $PI(F, R)$ is issued from an input rule r in which some conditions have been matched to facts $f_1, ..., f_k$ that have been inferred before (and added to $SAT(F, R)$), and thus allows us to infer the same conclusion as the input rule r on any set of facts including $f_1, ..., f_k$. The result $SAT(F, R) \cup PI(F, R)$ is then logically equivalent to the input deductive dataset $F \cup R$ for inferring facts on IDB predicates from the union of F and a set OF of external facts (with ODB predicates), i.e. for every fact f an external set of facts OF:

$$(F \cup OF, R) \vdash f \text{ iff } (SAT(F, R) \cup OF, PI(F, R)) \vdash f$$

Therefore, it can be equivalently used for proving goals by checking whether they belong to $SAT(F, R)$, or for rewriting goals by applying $QESQ$ to the $PI(F, R)$ (instead of the original R).

Experiments

We have conducted experiments on a real deductive dataset composed of 35 rules and about 6 million RDF facts from INA dataset. Most of the 35 rules capture local knowledge in the domain (functional properties and keys declared as schema constraints, and rules provided by INA experts), mappings between INA and DBpedia vocabularies, and general properties of owl:sameAs and owl:differentFrom. Some of the rules of our experiments involve a built-in predicate (called built-in:name-similar) to allow slight differences when comparing literal values corresponding to person names (e.g. R8 in Table 1). This predicate depends on a built-in function which checks if the similarity of the two name strings is above a given threshold. In all our experiments we used edit distance and 0.99 as a threshold. Other built-in predicates involved in the rules are not-equal, less-or-equal, sum, etc. It is worth noting that the 35 rules can be extended or modified without the need of changing the algorithmic machinery of our approach.

Experimental Goals and Set-Up

The goal of our experiments was threefold: (1) to show that external information available in Linked Open Data is useful to infer owl:sameAs and owl:differentFrom facts within INA referenced persons, and, thus, to disambiguate local homonyms; (2) to assess the gain in reduced imported facts of our Import-by-Query approach compared to approaches based on forward reasoning only; and (3) to evaluate the runtime of our Import-by-Query algorithm and the possible amortized gain if fact propagation is performed beforehand.

The external datasets from Linked Open Data with which the INA vocabulary shares terms are DBpedia.org and DBpedia.fr. The baseline for evaluating our two first goals is a set of 0.5 million external facts obtained by downloading from DBpedia.org and DBpedia.fr (using their SPARQL endpoints) all the facts about entities having the same name as one of the homonyms in the INA dataset. We applied a preprocessing step on the original INA dataset to keep only the facts on predicates appearing in the rules conditions. The resulting dataset contains almost 1.15 million of RDF facts and will be the INA dataset referred to henceforth.

Our algorithms have been implemented in SWI-Prolog. All the evaluations were done on a machine with an Intel i7 Quad-core processor and 6 GB of memory.

Experimental Results

For evaluating our first goal, we applied (using our forward reasoner) the set of 35 rules to (a) the INA dataset only and (b) the union of the INA dataset with the baseline external facts, and then we compared the number of owl:sameAs and owl:differentFrom facts on INA homonyms we obtained. The rules applied to the INA dataset only allowed to infer 2 owl:sameAs facts and 108 owl:differentFrom facts, compared to the 4,884 owl:sameAs and 9,764 owl:differentFrom facts inferred when the external facts were added to the process. This clearly demonstrates the benefit of using external information from Linked Open Data for local disambiguation. These resulting 14,648 facts are guaranteed to be correct under the assumption that both rules and data are correct. However, since this is not ensured for DBpedia data, we asked INA experts to evaluate a random sample of 500 of such facts, and all of them were assessed to be true.

The rule expressing sameAs transitivity is crucial for inferring all the owl:sameAs facts that cannot be inferred locally. More generally, full reasoning is very important to discover owl:sameAs and owl:differentFrom facts. In order to show this, we applied Silk to the same two datasets (the INA dataset only, and the union of the INA dataset with the baseline external facts). For doing so, we first had to translate our rules into the Silk specification language. It is not possible, however, to translate into Silk our rules concluding on owl:differentFrom atoms. Thus, we focused on the rules leading to owl:sameAs inference. Among the 4,884 owl:sameAs facts discovered by our full forward reasoner, Silk (which does not perform full reasoning) only discovered 88, i.e. less than 2% of the total. This shows that inference is important for data linkage.

For evaluating our second experimental goal, we took as reference boolean queries the above sample of 500 owl:sameAs and owl:differentFrom facts, and we applied our Import-by-Query algorithm to each of these boolean queries. The number of external facts imported by our algorithm for all boolean queries was 6,417, which makes, on average, 13 imported facts per boolean query. In contrast, the total number of baseline external facts needed to conclude the boolean queries with the forward reasoner was much higher ($\sim$500,000). This shows that our Import-by-Query algorithm reduces drastically the number of imported facts needed for disambiguating local data.

Concerning the runtime evaluation, the import-by-query algorithm requires 3 iterations on average — it successively outputs and evaluates 3 external sub-queries (each of them being produced by calling QESQ) — before termination. It takes on average 186 s per boolean query when applied to the initial set of rules and the local dataset. This drops to 7 s when it is applied to the partially instantiated rules obtained by fact propagation beforehand, which means a gain in time of 179 s ($\sim$96%). With respect to the fact propagation, we propagated all facts involving properties of class ina:Person. This took 191 s but it is done only once for all queries, and its cost is amortized very fast, as shown by the above numbers.

Discussion

We have proposed a novel approach for data linkage based on reasoning and adapted to the decentralized nature of the Linked Data cloud. This approach builds on the formal and algorithmic background of answering Datalog queries over deductive databases, that we have extended to handle external rewriting when local answers cannot be obtained. In contrast with existing rule-based approaches for data linkage [40, 31] based on forward reasoning to infer same-as facts, Import-by-Query is a backward chaining algorithm that imports *on demand* only external facts useful to infer target same-as facts handled as boolean queries. Our experiments have shown that this approach is feasible and reduces the number of facts needed to be imported. Compared to the depth-first approach sketched in [1] for distributed Query-Subquery, our QESQ algorithm generates external rewriting in a breadth-first way.

Performing fact propagation beforehand in order to apply Import-by-Query to a set of more specific rules than the original ones is an optimization close to the ones proposed in QueryPIE [46] for efficient backward reasoning on very large deductive datasets. One important difference, though, is that in the QueryPIE setting, the problem of handling recursive rules can be fully delegated to forward reasoning because all the facts are given and the recursive rules concern a well identified subset of them (so called terminological facts). Another major difference is that Import-by-Query performs query rewriting if no local answer is obtained from the input deductive dataset.

The Import-by-Query approach in [25] is limited to ABox satisfiability queries used as oracles in Tableau-based reasoning. Compared to the many recent works on ontology-based data access initiated by [14], in which query rewriting is done independently of the data, we have designed a *hybrid* approach that alternates (external) query rewriting and (local) query answering. We plan to look into this hybrid approach further, in particular to deal with ontological constraints expressible in Datalog$^{\pm}$ [13].

The interest of our rule-based approach is that it is generic and declarative: new rules can be added without changing the algorithmic machinery. At the moment the rules that we consider are certain. As a result, the same-as facts that they allow to infer are guaranteed to be correct (under the assumption that the input data does not contain erroneous facts). This is crucial to get automatically same-as facts that are certain, in particular when the goal of discovering same-as links is data fusion, i.e. replacement of two URIs by a single one in all relevant facts. Another added-value to get certain same-as and different-from facts is to find noisy data thanks to contradictions. However, in many cases, domain knowledge is not 100% sure such as pseudo-keys [11] and probabilistic mappings [45]. Data itself may be uncertain due to trust and reputation judgements towards data sources [9]. Handling uncertain domain knowledge should enable to discover more same-as facts that may be true even if inferred with some uncertainty. This is addressed in the next section.

3.2 Reasoning over uncertain RDF facts and rules [3]

We have designed a probabilistic framework to model and reason on uncertain RDF facts and rules, based on the semantics of probabilistic Datalog [23]. Probabilistic Datalog extends (deterministic) Datalog [2] by associating each ground fact and each instantiated rule with a basic probabilistic *event* that the corresponding fact or rule is true. Each derived fact is then inferred with its *provenance* in the form of an event expression made of a boolean combination of the basic events of the ground facts and rules involved in its derivation. It can be put in disjunctive normal form, in which a conjunction of events represents a derivation branch, and disjunctions represent the different derivation branches. Some simplifications can be performed before the computation of the resulting probabilities: a conjunction containing disjoint events can be suppressed ; basic events known to be certain can be removed from the conjunctions where they are involved thus leading to conjunctions with only uncertain events. An extreme case is when a conjunction is made of certain events only, which represent a way to derive a fact with certainty. In this case the whole event expression can be simplified to $\top$ which denotes certain events. The logical semantics of the (simplified) event expression is then the basis for computing the probability of the corresponding derived fact in function of the probabilities assigned to the events identifying the input facts and rules participating to its derivation. In the general case, computing the probability of the disjunction of conjunctions of events requires to know the probabilities of all the combinations of events in the expression. In practice, in particular in applications dealing with large amounts of data, only the probabilities of single events will be known. We will then make the same default assumptions of independence or disjointness of single events, as usually done in most Information Retrieval models [22]. To fit with such assumptions, we have to impose some constraints on the rules, that will be explained below.

Probabilistic RDF facts extends the standard data model of Linked Data used to state properties on entities referenced by so-called Uniform Resource Identifiers (URIs). Properties are themselves identified by URIs. So-called data properties relate entities with literals (e.g., numbers, strings or dates), while object properties relate two entities.

A **probabilistic RDF fact** is an RDF triple $t = (s, p, o)$ (in which the subject s is a URI, the predicate p is a URI, and the object o may be either a URI or a literal) associated with an event key e denoting the probabilistic event that t is true. A **probabilistic RDF rule** is a safe rule with variables, associated with an event key denoting the probability that any of its instantations is true.

Each probabilistic RDF fact and rule are assigned a distinct event key, except the certain facts and rules that are assigned the special event key $\top$ denoting events that are certain. For a probabilistic fact f (respectively rule r), we will denote $e(f)$ (respectively $e(r)$) the probabilistic event e associated with the fact f (respectively the rule r).

In the rules, we also allow conditions $B(\bar{x}, \bar{a})$ where B is a built-in predicate (i.e., a function call), $\bar{x}$ a vector of variables appearing in the triple conditions of the same rule, and $\bar{a}$ may be a non empty set of values of parameters for calling B. The following rule is an example of a rule with a built-in predicate:$Similar(?s_1, ?s_2, levenshtein, 0.2)$:

$$r_0 : (?x\, hasName\, ?s_1) \wedge (?y\, hasName\, ?s_2) \wedge\, Similar(?s_1,\, ?s_2,\, levenshtein,\, 0.2)$$
$$\rightarrow (?x\, sameName\, ?y)$$

For each pair of strings (s_1, s_2) for which the two triple conditions are satisfied by facts ($i_1\, hasName\, s_1$) and ($i_2\, hasName\, s_2$), $Similar(s_1,\, s_2,\, levenshtein,\, 0.2)$ applies the normalized Levenshstein distance $levenshtein(s_1, s_2)$ on the two strings s_1 and s_2, and if this distance is less than 0.2 returns the corresponding probabistic fact $Similar(s_1,\, s_2,\, levenshtein,\, 0.2)$ with $1 - levenshtein(s_1, s_2)$ as probabilty.

The semantics of inferred probabilistic facts is defined by extending the definition of $SAT(F, R)$ (see Definition 1) with their *provenance* defined as boolean combinations of all the events associated with the input facts and rules involved in their inference.

Definition 2 (Provenance-based semantics of probabilistic inferred facts). *For every fact f in $SAT(F,R)$, its provenance (denoted $Prov_{R,F}(f)$) is defined as follows:*

- *if $f \in F$: $Prov_{R,F}(f)$ = $e(f)$*
- *else: $Prov_{R,F}(f)$ = $\bigvee_{(r,\theta)\in R(f)} e(r) \wedge \bigwedge_{i\in[1..k]} Prov_{R,F}(\theta.TP_i(v_i))$ where $R(f)$ is the set of instantiated rules (r,θ) having f as conclusion (i.e., rules r of the form $TP_1(v_1) \wedge \ldots \wedge TP_k(v_k) \rightarrow TP(v)$ for which θ is a mapping such that $\theta.TP(v) = f$ and $\theta.TP_(v_i) \in SAT(F,R)$ for every $i \in [1..k]$).*

For every fact f in $SAT(F,R)$, its probability (denoted $P(f)$) is defined as the probability of its provenance: $P(f) = P(Prov_{R,F}(f))$

Illustrative example
Let us consider the following probabilistic RDF facts and rules (for which we omit to display the event keys) composed of 5 input facts and of 4 rules expressing different ways to infer sameAs facts between individuals (to have the same name, to have the same name and the same birthdate, to be married to the same individual, or by transitivity of the sameAs relation):
f_1: $(i_1\, sameName\, i_2)$
f_2: $(i_1\, sameBirthDate\, i_2)$
f_3: $(i_1\, marriedTo\, i_3)$
f_4: $(i_2\, marriedTo\, i_3)$
f_5: $(i_2\, sameName\, i_4)$
r_1: $(?x\, sameName\, ?y) \rightarrow (?x\, sameAs\, ?y)$
r_2: $(?x\, sameName\, ?y), (?x\, sameBirthDate\, ?y) \rightarrow (?x\, sameAs\, ?y)$
r_3: $(?x\, marriedTo\, ?z), (?y\, marriedTo\, ?z) \rightarrow (?x\, sameAs\, ?y)$
r_4: $(?x\, sameAs\, ?z), (?z\, sameAs\, ?y) \rightarrow (?x\, sameAs\, ?y)$
Three derived facts are obtained with their provenance:

$$Prov_{R,F}((i_1\, sameAs\, i_2)) =$$
$$(e(r_1) \wedge e(f_1)) \vee (e(r_2) \wedge e(f_1) \wedge e(f_2)) \vee (e(r_3) \wedge e(f_3) \wedge e(f_4))$$
$$Prov_{R,F}((i_2\, sameAs\, i_4)) = (e(r_1) \wedge e(f_5))$$
$$Prov_{R,F}((i_1\, sameAs\, i_4)) =$$
$$e(r_4) \wedge Prov_{R,F}((i_1\, sameAs\, i_2)) \wedge Prov_{R,F}((i_2\, sameAs\, i_4))$$

The first one captures that the fact $(i_1\, sameAs\, i_2)$ can be inferred as a result of 3 different derivation branches (one using the rule r_1 and the input fact f_1, another one using the rule r_2 and the input facts f_1 and f_2, and the third one using the rule r_3 and the input facts f_3 and f_4). The second one captures that $(i_2\, sameAs\, i_4)$ results from a single derivation branch, using the rule r_1 and the fact f_5. The last one illustrates how the provenances can be built iteratively during the saturation process: the last derivation step leading to the inference of $(i_1\, sameAs\, i_4)$ involves the rule r_4 and two facts inferred at a previous iteration (namely, $(i_1\, sameAs\, i_2)$ and $(i_2\, sameAs\, i_4)$) for which the event expressions computed beforehand as their provenance can be combined with the event key of r_4.

These event expressions can be simplified by exploiting facts and rules that are certain. For instance, if we know that the two facts f_2 and f_3 are certain as well as the rule r_4, we can suppress $e(f_2)$, $e(f_3)$ and $e(r_4)$ in the conjuncts of the above expressions because they are all equal to the event $\top$ always true. We now obtain for $Prov_{R,F}((i_1\, sameAs\, i_2))$: $(e(r_1) \wedge e(f_1)) \vee (e(r_2) \wedge e(f_1)) \vee (e(r_3) \wedge e(f_4))$

When many facts and several rules are certain, such simplifications lead to a drastic reduction of the size of event expressions, which is important for the feasibility and the scalability of the approach in practice.

This example illustrates how the construction and the simplification of the provenance can be incorporated into the saturation process and thus how a given forward-reasoning algorithm can be easily extended to compute the provenance during the inference of the corresponding facts.

The ProbFR algorithm.

Algorithm 1 describes the ProbFR algorithm that we have implemented and used in our experiments.

Algorithm 1: The *ProbFR* algorithm

Input: A set F of input (probabilistic) facts and a set R of (probabilistic) rules

Output: The set F_{sat} of inferred (probabilistic) facts with for each inferred fact f its event expression $x(f)$

```
(1)   for each $f \in F$: $x(f) \leftarrow e(f)$
(2)   $F_{sat} \leftarrow F$
(3)   $\Delta \leftarrow F$
(4)   repeat
(5)     $\Delta_1 \leftarrow \emptyset$
(6)       foreach rule $r$: $c_1 \wedge \ldots \wedge c_k \rightarrow c$ for which there exists a
          substitution $\theta$ and facts $f_1, \ldots, f_k \in F_{sat}$ (among which atleast
          one of them belongs to $\Delta$) such that $f_i = \theta.c_i$ for every $i \in [1..k]$:
(7)     let $f = \theta.c$:
(8)     if $f \notin F_{sat}$
(9)      add $f$ to $\Delta_1$
(10)      $x(f) \leftarrow \mathcal{N}_\vee(e(r) \wedge \bigwedge_{i\in[1..k]} x(f_i))$
(11)     else $x(f) \leftarrow x(f) \vee$
(12)               $\mathcal{N}_\vee(e(r) \wedge \bigwedge_{i\in[1..k]} x(f_i))$
(13)      $F_{sat} \leftarrow F_{sat} \cup \Delta_1$
(14)      $\Delta \leftarrow \Delta_1$
(15)  until $\Delta_1 = \emptyset$
(16)  return $F_{sat}$
```

It starts with the set of initial facts and rules and repeats inference steps until saturation. Each inference step (Line (4) to (15)) triggers all the rules whose conditions can be matched with known facts (i.e., input facts or facts inferred at previous steps). At each iteration, the set Δ contains the facts that have been inferred at the previous iteration. The constraint (expressed in Line (6)) that rules are only triggered if atleast one of their conditions can be matched with facts in Δ guarantees that instantiated rules are not triggered twice during the inference process. The algorithm stops as soon as no new fact has been inferred during a given iteration (i.e., Δ_1 remains empty over this iteration). The algorithm returns the set F_{sat} of inferred facts, and computes for each of them an event expression $x(f)$ (Lines (10) and (11)). The function $\mathcal{N}_{\vee}$ denotes the transformation of a conjunction into its disjunctive normal form. It consists in applying ieratively the distributivity of the conjunction connector ($\wedge$) over the disjunction connector ($\vee$), and in simplifying when possible the (intermediate) results as follows: (1) remove the duplicate events and the certain events $\top$ from each conjunction of events, (2) if a conjunction within a disjunction becomes empty (i.e., if all its events are certain), replace the whole disjunction by $\top$. Each event expression $x(f)$ is thus either $\top$ or of the form $Conj_1 \vee ... \vee Conj_l$ where each $Conj_i$ is a conjunction of event keys tracing the uncertain input facts and rules involved into one of the l branches of uncertain derivation of f.

The termination of the ProbFR algorithm is guaranteed by the fact that the rules are safe.The only facts that can be inferred from safe rules and a set F of ground atoms are instantiations of conclusion atoms by constants appearing in F. Their number is finite. More precisely, since the input facts and conclusion atoms are built on are binary predicates, the number of constants appearing in the input facts is less than $2 \times |F|$ (at most two distinct constants per input fact), and the number of inferred facts is then less than $4 \times |R| \times |F|^2$ (atmost as many predicates in conclusion as rules, and for each of them, atmost as many instantiations as pairs of constants).

The following theorem states the soundness and completeness of the algorithm.

Theorem 1.
Let F_{sat} be the result returned by $ProbFR(F, R)$:
$F_{sat} = SAT(F, R)$.
For each $f \in F_{sat}$, let $x(f)$ be the event expression $x(f)$ computed by $ProbFR(F, R)$:
$x(f) \equiv Prov_{F,R}(f)$

For the first point, we prove by induction on i that each iteration $i \geq 1$ of the algorithm computes the set of facts $F_i = T_R(F_{i-1})$ (as defined in Definition 1), and thus $SAT(F, R)$ at the last iteration where the least fixed point reached. For the second point, for a derived fact f, we prove, by induction on the number n of iterations of $ProbFR$ after which no new instantiation of rules can infer f, that $x(f)$ is a disjunctive normal form of $Prov_{F,R}(f)$, and therefore is logically equivalent to it.

As a result of Definition 2 and Theorem 1, it worths to emphasize that the probabilities values of inferred facts is independent of the order in which the rules are triggered to derive them.

Data complexity analysis.

We are interested in estimating how the worst-case time complexity of the algorithm depends on the size $|F|$ of the input data, which is the most critical parameter in the setting of Linked Data. The number of iterations of ProbFR is atmost $|F_{sat}|$, which is less than $4 \times |R| \times |F|^2$ as shown just above. At each iteration, in the worst case, the condition part of each rule must be evaluated against the facts, and the event expressions for the provenance of the inferred facts must be computed. Let c the maximum number of conditions per rule. The evaluation of each condition part of each rule can be performed in polynomial time (in fact, in at most $|R| \times |F_{sat}|^c$ elementary steps).

For the computation of the event expressions, the most costly operation is the transformation $\mathcal{N}_\vee$ into disjunctive normal form of conjunctions $e(r) \wedge \bigwedge_{i \in [1..k]} x(f_i)$. The number k of conjunctions is less than the bound c of conditions per rule, and each $x(f_i)$ is a disjunction of at most l conjunctions of event keys, where l is the maximum number of uncertain derivation branches for inferred facts. This parameter l is bounded by b^d where d is the maximal depth of reasoning to infer a fact from F and R, and b is the maximal branching factor of $ground(F, R)$ (which denotes the set of rules triggered during the execution of $ProbFR(F, R)$). Therefore, each call of $\mathcal{N}_\vee$ performs at most $b^{d \times c}$ distributivity operations on conjunctions of at most $|F| + |R|$ event keys. Since the maximal depth of reasoning is the number of iterations of $ProbFR(F, R)$, d can be equal to $|F_{sat}|$. Then, the data complexity of the provenance computation may be exponential in the worst-case. This meets known results on query evaluation in probabilistic databases [43]. Different solutions are possible to circumvent this worst-case complexity, like restricting the form of rules/queries like in [17] or imposing some constraints on the input facts (such as a bounded treewidth in [6]). In practice, in particular if most of the input facts are certain, the size of the event expressions remains small. If all the input facts are certain, the only event keys that can be involved in the event expressions are the ones attached to the uncertain rules. The complexity of the algorithm can be controlled by imposing a practical bound in the number l of conjunctions produced in Line (11). This solution is justified in our setting since the computed probabilities are used to keep only the most probable inferred facts, i.e., the facts that are inferred with a probability greater than a given high threshold. For our experiments, we have limited this number l to be 8.

Effective computation of probabilities of inferred facts from their provenance. For each inferred fact, given its provenance as an event expression in disjunctive normal form, the following formula is the basic theoretical tool to compute its probability:

(1) $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$.

The recursive application of the above formula for computing the probability of a disjunction of l conjunctions of events $E_1 \vee \ldots \vee E_l$ leads to alternate the subtractions and additions of the probabilities of all the possible conjunctions $E_{j_1} \wedge \ldots \wedge E_{j_i}$.This raises two major issues: first, their number is exponential in l; second the exact values of all these probabilities is usually not available.

An usual way to circumvent the latter is to make the assumption of independence between events, as it is done in probabilistic databases [43] or in most Information Retrieval models [22]. In our case however, two rules such that the condition part of one rule is contained in the condition part of the second (like the rules r_1 and r_2 of the example) are obviously not independent. For such rules, we enforce pairwise disjointness by imposing that the more general rule applies only if the more specific rules do not apply. In this way, we are sure that the corresponding dependent events do not appear in any event expression computed during the saturation process. To be consistent with the probabilistic setting, we also impose that the probability assigned to the event corresponding to the more specific rule (r_2 in our example) is higher than the one assigned to the event of more general rule (r_1 in our example).

For each pair r, r' with same conclusion (up to variables names), let us denote $r \preceq r'$ if $Cond_r$ is contained into $Cond_{r'}$. Checking whether $r \preceq r'$ can be done by using any conjunctive query containment algorithm [15] with a complexity independent of the data.

To summarize, we make the assumptions of :

- pairwise *disjointness* between events associated with pairs of rules r, r' such that $r \preceq r'$

- *independence* of the events that are not disjoint.

For the effective computation of the probability of an inferred fact f,

- first, the provenance expressions $x(f) = E_1 \vee \ldots \vee E_l$ computed by the $ProbFR$ algorithm are simplified by removing each conjunction of events E_i in which an event $e(r)$ appears if there is a conjunction of events E_j $(j \neq i)$ such that $e(r')$ appears in E_j and $r \preceq r'$.

- second, the probability of f is computed by iteratively applying the formula (1) on the resulting event expression.

In our example, the rules r_2 and r_1 are such that $r_1 \preceq r_2$. We can thus remove the conjuncts containing $e(r_1)$ and we obtain for $x((i_1\, sameAs\, i_2))$:

$(e(r_2) \wedge e(f_1)) \vee (e(r_3) \wedge e(f_4))$.

Now, considering the remaining events as independent, we can compute the effective probability $P((i_1\, sameAs\, i_2))$ as follows:

$$P((i_1\, sameAs\, i_2)) =$$
$$(P(e(r_2)) \times P(e(f_1))) + (P(e(r_3)) \times P(e(f_4)))$$
$$-(P(e(r_2)) \times P(e(f_1)) \times P(e(r_3)) \times P(e(f_4)))$$

Note that the above simplification can be incorporated into the $ProbFR$ algorithm at each update of event expression (Line (11)) and that determining the possible pairs of rules r, r'such that $r \preceq r'$ can be done in advance before launching $ProbFR$ as it is independent of the set of facts F.

This simplification has an impact on the practical complexity of the effective computation of the probabilities, even if, in theory and in the worst-case, it remains exponential in the number l of conjunctions within provenance expressions. As we have explained it before, this number l can be bounded in practice.

The assumption of disjointness between events associated with rules r, r' such that $r \preceq r'$ is important for the feasability of the approach but it also fits well with the open-world assumption that holds in Linked Data. In fact, it captures a restricted form of negation since, under this disjointness assumption, the event $e(r)$ models worlds where the condition of r is satisfied and the additional conditions of r' are not satisfied.

Setting up of the input probabilities.

The above approach for probabilistic inference is agnostic with respect to the way the input probabilities are obtained, either given by experts, returned by built-in predicates or tools, or learned by supervised methods. This said, it is important to note that training sets (required by supervised machine learning techniques) that would be big enough to scale to the setting of Linked Data do not exist and are almost impossible to build manually. On the other hand, it is quite easy for domain experts to decide whether a given rule is uncertain, but setting up its probability is tricky. The two-steps computation of a provenance-based approach as ours has the big advantage to possibly re-compute the numerical values of probabilities for the inferred facts from the provenance expressions computed once for all. This enables to start with a rough setting of rules probabilities chosen from a small set of values just for distinguishing rules on a simple scale of uncertainty (for instance set at 0.9 the rules a priori considered as almost always certain, 0.8 the rules judged as highly probable but less than the previous ones, and so on), and to adjust these values a posteriori based on a feedback on a sample of results. The provenance of wrong sameAs links inferred with a high probability provides explicitly the rules involved in the different reasoning branches leading to their derivation. It is a useful information for a domain expert to choose the rules to penalize by decreasing their numerical probabilities.

3.3 Rule-based data linkage with uncertainty

When used for data interlinking, rules typically translate varied knowledge that combines schema constraints, alignments between different ontologies and general properties on OWL relations such as owl:sameAs. This knowledge may be certain, but, very often, it has some degree of uncertainty. It is the case when a correspondence in an ontology alignment is attached a confidence value lower than 1, or when domain experts provide knowledge they are not 100% sure about, or the case of pseudo-keys that are automatically computed by pseudo-key discovery tools [11, 44]. This uncertain knowledge can be translated by means of probabilistic rules.

Tables 2 and 3 show rules translating, respectively, certain and uncertain knowledge for the task of interlinking person entities in DBpedia and MusicBrainz datasets. These rules are actually part of the rules that we used in our experiments (reported in Section 3.4). Rule musicalArtist in Table 2, for example, is a certain rule that translates the DBpedia knowledge that the class dbo:musicalArtist is subsumed by dbo:Artist. Rule enrich_dboBand1 translates a certain correspondence in an alignment between Schema.org vocabulary and DBpedia ontology stating that the class schema:Person is subsumed by dbo:Person. The rule sameAsVIAF is a certain rule that translates the assertion that the VIAF id is a key for persons and, therefore, allows to infer sameAs links between person entities from DBpedia and MusicBrainz. Notice that this rule actually involves the two equivalent properties dbp:viaf and mb:ViafID of DBpedia and MusicBrainz vocabularies. This means that the condition ($?x$ dbp:viaf $?id$) in the rule will be instantiated by a DBpedia entity, and ($?y$ mb:ViafID $?id$) by a MusicBrainz entity. This kind of "key across different datasets" is called a link key in the literature [10]. Note also that, instead of using owl:sameAs, we use our own customized sameAs predicates (:sameAsPerson) which allowed us to easily identify the type of the inferred sameAs links in our experiments. Rule sameAsIsPerson1 is a certain rule that translates transitivity of sameAs.

ID	Conditions	Conclusion
musicalArtist	($?w$ dbo:musicalArtist $?x$)	($?w$ dbo:artist $?x$)
enrich_dboBand1	($?x$ rdf:type schema:MusicGroup)	($?x$ rdf:type dbo:Band)
sameAsVIAF	($?x$ dbp:viaf $?id$) , ($?y$ mb:ViafID $?id$)	($?x$:sameAsPerson $?y$)
sameAsIsPerson1	($?x$:sameAsPerson $?y$), ($?z$ mb:is_person $?y$)	($?x$:sameAsPerson $?z$)
similarNamesPerson	($?x$ rdf:type dbo:Person), ($?x$ rdfs:label $?l$), MBsolrsimilar($?l$,0.8,$?z$,'persons_mb')	($?x$:solrPSimilarName $?z$)

Table 2. Certain rules for interlinking person entities in DBpedia and MusicBrainz.

ID	Conditions	Conclusion	Weight
sameAsBirthDate	($?x$:solrPSimilarName $?l$), ($?y$ skos:myLabel $?l$), ($?x$ dbo:birthDate $?date$), ($?y$ mb:beginDateC $?date$ (	($?x$:sameAsPerson $?y$)	w_1
sameAsPersonArtistWr	($?w1$ dbo:artist $?x$), ($?w1$:solrWrSimilarName $?lw$), ($?y$ mb:writer $?w2$), ($?w2$ skos:myLabel $?lw$) , ($?x$:solrPSimilarName $?lp$), ($?y$ skos:myLabel $?lp$)	($?x$:sameAsPerson $?y$)	w_2
sameAsMemberOfBand	($?x$:solrPSimilarName $?l$), ($?y$ skos:myLabel $?l$), ($?y$ mb:member_of_band $?gr2$), ($?gr2$ skos:myLabel $?lg$), ($?gr1$ dbp:members $?x$), ($?gr1$:solrGrSimilarName $?lg$)	($?x$:sameAsPerson $?y$)	w_3

Table 3. Uncertain rules for interlinking person entities in DBpedia and MusicBrainz.

Rule similarNamesPerson deserves special attention because it contains a built-in predicate (namely MBsolrsimilar) that encapsulates the call to a full-text search tool (namely Solr [9]) to extract strings from MusicBrainz similar to labels of person entities in DBpedia. More precisely, for each string instantiation s of the variable $?l$, obtained by mapping with DBpedia facts the two first conditions ($?x$ rdf:type dbo:Person) and ($?x$ rdfs:label $?l$) of the rule, MBsolrsimilar(s, 0.8, $?z$, 'person_mb') is a procedure call returning as many probabilistic facts MBsolrsimilar(s, 0.8, s', 'person_mb') as labels s' of person entities in MusicBrainz detected by Solr as similar to s with a similarity greater than 0.8. The probability attached to each probabilistic fact MBsolrsimilar(s, 0.8, s', 'person_mb') is the calculated string similarity. Thus similarNamesPerson is a certain rule that will infer uncertain facts of the form ($?x$:solrPSimilarName $?z$) due to condition MBsolrsimilar($?l$,0.8,$?z$,‘persons_mb’), which will be instantiated with built-in uncertain facts. Built-in predicates such as MBsolrsimilar enable to embed standard similarity functions into our rule-based approach to overcome the problem of mis-

[9] http://lucene.apache.org/solr/

spelling errors in names of persons, groups and songs that may occur in DBpedia and MusicBrainz datasets.

Table 3 shows three additional rules allowing to infer sameAs links between person entities from DBpedia and MusicBrainz datasets, but, in contrast with the sameAsVIAF rule explained above, they are not 100% certain. Rule sameAsBirthDate, for example, says that if two persons have similar names and the same birthdate then they are $likely$ to be the same person. This rule must be considered uncertain for two reasons. First, it relaxes the strict condition of having exactly the same name by the soft constraint of having similar names as it is specified by ($?x$:solrPSimilarName $?l$). Second, strictly speaking the properties ”name” and ”birthdate” do not constitute a key, even if it is likely that two named entities representing persons that are well-known enough to be described in datasets like DBpedia and MusicBrainz will refer to the same person if they share the same name and birthdate. In fact, sameAsBirthDate translate a *soft* link key, as it combines the equivalent properties dbo:birthDate and mb:beginDateC that are used in DBpedia and MusicBrainz vocabularies to relate a person with her date of birth. The rules sameAsPersonArtistWr and sameAsMemberOfBand are uncertain too. The first one says that, if two persons have similar names and they are artists of songs with similar names, they are the same person, and the second rule says that if two persons have similar names and are members of musical bands with similar names, they are the same person. Again, this may not be always true, but in most cases. The weights in Table 3 correspond to the probabilistic events associated with each of these uncertain rules.

An important point to emphasize is that the (certain or uncertain) rules allowed in our rule-based modeling express pieces of knowledge that can be assembled and combined through several reasoning steps. For instance, the condition ($?w_1$ dbo:artist $?x$) of the sameAsPersonArtistWr rule may be triggered by facts inferred by the musicalArtist rule. The chaining between rules is not known in advance and is determined by the input datasets which they apply to. In addition, due to recursive rules (such as sameAsIsPerson rule), even if the termination of the saturation process is guaranteed, the number of reasoning steps cannot be known in advance and also depends on the input datasets. It is worthwhile to note that recursive rules add an expressive power that is required for data linkage in particular to express sameAs transitivity.

The translation into rules can be semi-automatic, for instance for translating into certain rules schema constraints that have been declared in OWL such as the functionality or transitivity of some relations, or for translating into (certain or uncertain) rules alignments discovered by ontology mapping tools [19]. A certain number of uncertain rules useful for data interlinking must however be provided by domain experts to express fine-grained knowledge that may be specific to the datasets concerned by the linkage task. While it is quite easy for domain experts to decide whether a given rule is uncertain, setting up its probability is tricky. The two-steps computation has the big advantage to possibly re-compute the numerical values of probabilities for the inferred facts, starting from the event expressions built once for all in the first step that is a symbolic computation independent of the numerical values of rules probabilities. This enables to start with a rough setting of rules probabilities chosen from a small set of values just for distinguishing rules on a simple scale of uncertainty (for instance set at 0.9 the rules a priori considered as almost always certain, 0.8 the rules judged as highly probable but less than the previous ones, and so on), and to adjust these values a posteriori based on a feedback on a sample of results. The event expressions of wrong sameAs links inferred with a high probability provide explicitly the rules involved in the different reasoning branches leading to their derivation. It is a useful information for a domain expert to choose the rules to penalize by decreasing their numerical probabilities.

In our experiments, such an incremental adjustment for the probabilities of the three uncertain rules of Table 3 resulted into: $w_1 = 0.9$, $w_2 = 0.4$ and $w_3 = 0.6$.

It is worth emphasizing that rules with quite low probabilities (such as 0.4 for the sameAsPersonArtistWr rule) can yet significantly contribute to the final probability of a fact inferred by different reasoning branches.

3.4 Experimental evaluation

We have conducted experiments to evaluate the performance of our method on real datasets. Our main goal was to measure the effectiveness of our method to discover links at large scale, and to assess the expected gain in terms of recall and the loss in precision when using uncertain rules instead of certain rules only. We also wanted to show how the probabilistic weights attached to the links allow to filter out incorrect links. Finally, we aimed at comparing our tool to a state-of-the-art interlinking tool, namely Silk [48].

Experimental Setting We used three datasets in our experiments: DBpedia, INA and MusicBrainz. The objective was to find sameAs links between named entities of person, musical band, song and album included in the datasets. Our choice of these datasets was based upon the fact that these are all large datasets (tens of millions of triples), and of a very different nature: DBpedia was built from Wikipedia infoboxes, INA from catalog records mainly containing plain text, and MusicBrainz from more structured data coming from a relational database.

The DBpedia version we used was DBpedia 2015-04,[10] the latest version at the time the experiments were conducted. From all available (sub) datasets, we only used the ones including RDF triples with properties appearing in the rules that we used in the experiments (below we give more details about the rules), which make together one single dataset of around 73 million RDF triples. The INA dataset contains around 33 million RDF triples, while the MusicBrainz dataset around 112 million RDF triples. The INA dataset was built from all the records (plain text) in a catalog of French TV musical programs using an specialised RDF extractor. Some RDF facts in the INA dataset have numerical weights between 0 and 1 since their accuracy could not be 100% assessed during the extraction process. The MusicBrainz dataset was built from the original postgreSQL table dumps available at the MusicBrainz web site using an RDF converter. This version is richer than the one of the LinkedBrainz project.[11]

[10] http://wiki.dbpedia.org/Downloads2015-04

[11] http://linkedbrainz.org/

Table 4 shows the number of person, musical band, song and album entities in each of the considered datasets, where Person, e.g. symbolises the class union of all the classes that represent persons in each dataset. No bands or albums are declared in INA, written NA (not applicable) in Table 4.

Class	DBpedia	MusicBrainz	INA
Person	1,445,773	385,662	186,704
Band	75,661	197,744	NA
Song	52,565	448,835	67,943
Album	123,374	1,230,731	NA

Table 4. Number of person, musical band, song and album entities in DBpedia, MusicBrainz and INA.

We have designed two sets of rules that we used as inputs for our algorithm to interlink DBpedia and MusicBrainz first and then MusicBrainz and INA. We came up with 86 rules for interlinking DBpedia and MusicBrainz, from which 50 of them are certain and 36 are uncertain, and 147 rules for interlinking MusicBrainz and INA, 97 of them certain and 50 uncertain. By a way of example, Table 2 and Table 3 of Section 3.3 include some of the certain and uncertain rules that we used for interlinking DBpedia and MusicBrainz.

ProbFR has been implemented on top of Jena RETE and uses SWI-Prolog v6 to compute the disjunctive normal forms for the event expressions during RETE inference. Prolog is also used to implement the second step of ProbFR, i.e. to compute effective probabilities given event expressions. In order to avoid potential combinatorial explosion, the current parameter of ProbFR is tuned to a maximum of 8 derivation branches for each event expression. All ProbFR experiments were run on a Bi-processor intel Xeon 32 x 2.1GHz, 256 GB of RAM, with Linux CentOS 6 as operating system.

Experimental Results We ran our algorithm to interlink DBpedia and MusicBrainz first, and then MusicBrainz and INA, using in each case the corresponding rules. Our algorithm discovered 144,467 sameAs links between entities of DBpedia and MusicBrainz and 28,910 sameAs links between entities of MusicBrainz and INA. Additionally, our algorithm found 132,166 sameAs links internal to the INA dataset.

In order to evaluate the quality of the found links, and since no gold standard was available, we estimated precision, recall and F-measure by sampling and manual checking. In order to compute precision, for each of the classes considered we took a sample of 50 links from the links found by our algorithm (i.e. 200 links in total for DBpedia and MusicBrainz, and 100 links for MusicBrainz and INA), and we manually checked whether these links were correct. For computing recall, we randomly selected 50 instances of each of the classes, and we found links manually. Then, we calculated recall based on this make-do gold standard. F-measure was based on the estimations of precision and recall.

In order to assess the gain of using uncertain rules, we also ran our algorithm only with certain rules, and then we compared the results obtained using only certain rules with the ones obtained using all rules (both certain and uncertain rules). This concerned the experiments between DBpedia and MusicBrainz only, as no other certain rule than sameAs transitivity was used for MusicBrainz and INA.

	DBpedia and MusicBrainz						**MusicBrainz and INA**					
	Only certain rules			All rules			Only certain rules			All rules		
	P	R	F	P	R	F	P	R	F	P	R	F
Person	1.00	0.08	0.15	1.00	0.80	0.89	NA	NA	NA	1.00	0.34	0.51
Band	1.00	0.12	0.21	0.94	0.84	0.89	NA	NA	NA	NA	NA	NA
Song	NA	NA	NA	0.96	0.74	0.84	NA	NA	NA	1.00	0.40	0.57
Album	NA	NA	NA	1.00	0.53	0.69	NA	NA	NA	NA	NA	NA

Table 5. Precision (P), recall (R) and F-measure (F) for the task of interlinking DBpedia and MusicBrainz datasets, and MusicBrainz and INA datasets, using certain rules only, and certain and uncertain rules together.

Table 5 shows all the results. Let us focus on the results concerning DBpedia and MusicBrainz. As expected, when certain rules were used only, precision was 100%. This only concerns Person and Band classes because the initial set of rules did not include any certain rule concluding links for Song and Album (written NA in Table 5). However, recall was very low: 0.08 for Person and 0.12 for Band. When both certain and uncertain rules were used, a 100% precision was achieved for Person and Album classes only, since for Band and Song, precision was 0.94 and 0.96, respectively. However, recall increased significantly for Person and Band: 0.80 and 0.84. This shows the gain of using uncertain rules for data linkage. Now, when looking at the samples of Band and Song classes, we realised that all wrong links had a probability value lower than 0.9 and 0.6, respectively. This means that, when limited to those links having a probability value higher or equal to 0.9 and 0.6, the estimated precision for the classes Band and Song was 100% (Table 6). The estimated recall was 0.80 and 0.54. This shows the gain of using weights for interlinking.

	P	R	F
Band$_{\geqslant 0.90}$	1.00	0.80	0.89
Song$_{\geqslant 0.60}$	1.00	0.54	0.72

Table 6. Gain of using weights for interlinking DBpedia and MusicBrainz.

Table 7 shows the number of links that are discovered when n sameAs rules[12] are implied in the derivation. For instance, 28,614 links are discovered using two sameAs rules, and among these links 27,692 are new links, i.e. they were not discovered using only one rule. With tools like Silk and LIMES, using the same set of rules, we can expect to find around 115,609 links only.

[12] We only consider rules that conclude to sameAs statements because other rules can be handled with preprocessing by tools like Silk or LIMES.

# rules	# links	# new links
1	115,609	115,609
2	28,614	27,692
3	1,790	1,152
4	59	14

Table 7. Number of links discovered when n rules are implied in the derivation. Results given for interlinking DBpedia and MusicBrainz.

Comparison with Silk Since Silk cannot handle rule chaining, we divided the rules used by ProbFR into sameAs rules (i.e. rules with sameAs in the conclusion), and intermediate rules that are used to trigger antecedents of other rules (including the sameAs rules). We manually translated these intermediate rules into SPARQL Update queries and these updates were performed before the Silk execution. Some sameAs rules could not be translated into Silk because they are recursive (sameAs appears in their antecedent and conclusion). To be able to compare methods on the same basis, we employed the levenshtein normalised distance with a threshold of 0.2, which corresponds to the similarity parameter set up to 0.8 in Solr. The aggregation of different comparisons within a rule was performed using maximum distance to be compliant with the conjunction used in rules. We executed Silk for interlinking DBpedia and MusicBrainz. Silk found 101,778 sameAs links, from which 100,544 were common to the ones found by ProbFR. ProbFR found 43,923 links that were not discovered by Silk and Silk found 1,234 links not discovered by ProbFR. In theory all the links discovered by Silk should have been discovered by ProbFR and Silk should have found up to 115,609 links. These differences can be explained by the way levensthein distance are implemented in each tools and by a normalisation of URL that is performed by ProbFR and not available in Silk. As a conclusion, ProbFR outperformed Silk because of rule chaining (more links are discovered). Dealing with uncertainty allows to enhance precision without losing much recall.

In terms of time performance, Silk took more than 53 hours (with 16 threads, blocking activated, on a Bi-processor Intel Xeon, 24 x 1.9GHz) while ProbFR achieved the task in 18 hours (on a Bi-processor Intel Xeon, 32 x 2.1GHz). Even if the difference could be partially explained by the difference in hardware, the main reason comes from implementation design. Silk mainly relies on disk indexing and uses few RAM (around 1-2 GB) while ProbFR runs into main memory and uses around 250 GB of RAM for this experiment.

3.5 Discussion

Dedupalog [7] is a Datalog-like language that has been specially designed for handling constraints useful for record linkage. It handles both hard and soft rules that define respectively valid clusterings and their costs. The associated algorithm computes a valid clustering with a minimal cost. Whereas the general problem is NP-complete, they provide a practical algorithm that scales to the ACM database that contains 436,000 records. Even if the algorithmic techniques are very different from ours, the scalability is obtained by similar restrictions on the rule language. However, the goal is to compute a valid clustering and not to compute probabilities of inferred facts.

Probabilistic logical frameworks such as Markov logic [41] and Probabilistic Soft Logic (PSL) [12] have been used for entity resolution. Markov Logic allows for full probabilistic reasoning. The weights attached to formulas are learned either from data or from probabilities arbitrarily given. This learning phase is made under closed-world assumption. Once a Markov Logic Network is learned, the weighted satisfiability of any candidate link has to be computed. This is not scalable in practice. Then, candidate pairs are filtered using a cheap similarity such as TF.IDF: non matching pairs are added as false atoms. Experiments have been conducted on Cora dataset (1295 instances) and a sample of Bibserv ($10,000$ instances). PSL allows probabilistic inference based on similarities functions. As Markov Logic, formulas' weights are learned making closed world assumption. Furthermore, it allows to assign weights to facts using the similarity of sets of property values (which assumes that sets are fully known). Like Datalog, it is restricted to conjunctive rules. Experiments have been performed on the task of Wikipedia article classification and ontology matching.

Contrary to aforementioned approaches, in ProbFR, probability computation and inference are separated. All rules are iteratively applied to compute the saturation and the provenances of every deduced facts. Probabilities are then computed from the provenances. This allows to change the probabilities assigned to rules and reevaluated quickly the probabilities of inferred facts without recomputing the saturation. Another difference is that probabilities attached to formulas can be given or learned from data. No further learning is required.

Decoupling the symbolic computation of provenances from the numerical computation of probabilities makes probabilistic reasoning more modular and more transparent for users. This provides explanations on probabilistic inference for end-users, and useful traces for experts to set up the input probabilistic weights.

Currently, the threshold for filtering the probabilistic sameAs facts that will be retained as being true must be set up and adjusted manually. As future work, we plan to design a method to set up this threshold automatically by, besides inferring sameAs facts, inferring differentFrom facts too, and then exploiting the sameAs and differentFrom facts (and their probabilities) that are inferred for the same pairs of entities. We also plan to design a backward-reasoning algorithm able to deal with probabilistic rules, that could be combined with the ProbFR probabilistic forward-reasoner for importing on demand useful data from external sources.

4 Extraction of modules from RDF knowledge bases [39]

The Semantic Web consolidated a legacy of ontologies and databases today seen as *reference systems* for building new Semantic Web applications. To illustrate, consider a medical application for anatomy, whose goal is to showcase the structure of the human body, the most common pathologies and diseases, and the scientists that contributed to their study. A structural description of human anatomy can be drawn from FMA[13] or My Corporis Fabrica (MyCF).[14] A taxonomy of clinical terms about diseases can be extracted from SNOMED, [15] while biographical informations about scientists implied in studies can be taken from DBPedia.[16] These reference system contain knowledge that can be *reused* to minimize the introduction of errors in the application. However, it is inconvenient to integrate in the application the whole datasets, as they contain complementary data and ontology axioms that are logically redundant. It is thus preferable to extract lightweight fragments of these reference systems - the *modules* - that are relevant for the application, and then to build on top of them.

While extracting modules from ontologies has been largely investigated for Description Logics (DL) [24, 32], module extraction from RDF triplestores has received little attention. Yet, more and more huge RDF datasets are flourishing in the Linked Data and some of them, like DBPedia or YAGO [42], are increasingly reused in other more specialized datasets. RDF is a graph data model based on triples accepted as the W3C standard for Semantic Web data, with a simple ontology language, RDF Schema (RDFS). The W3C proposed OWL for writing expressive ontologies based on DL constructors. Whereas OWL is often seen as an extension of RDFS, this is not exactly the case. Both RDFS and the RDF query language (SPARQL) feature the possibility of accessing *at the same time* the ontology data and schema, by making variables ranging over classes or properties. This *domain meta-modeling* goes beyond the first-order setting typically considered in DL [18]. As a consequence, DL modularization frameworks are not applicable to popular RDF datasets like DBpedia or YAGO. Also, the clear separation between the ABox and the TBox made in DL to define the semantics of modules is not appropriate for RDF where facts and schema statements can be combined within a single RDF triplestore to accommodate heterogeneous knowledge from the Web. Another limit of the current approaches is that the existing semantics do not allow to limit the *size* of the extracted modules. As discussed in [24], the risk in practice is to output large portions of the initial ontologies, thus jeopardizing the gains of modularization.

The RDF knowledge bases that we consider are deductive RDF datasets as defined in Section 2.3: an RDF knowledge base is a pair $\langle D, R\rangle$ where D is an RDF dataset and R is a finite set of (possibly recursive) rules.

Figure 4 presents an RDF dataset, together with its graph version. The example is inspired by the MyCF ontology [36], which classifies digital representation of human body parts, acquired by IRMs or tomographies, according to anatomical knowledge.

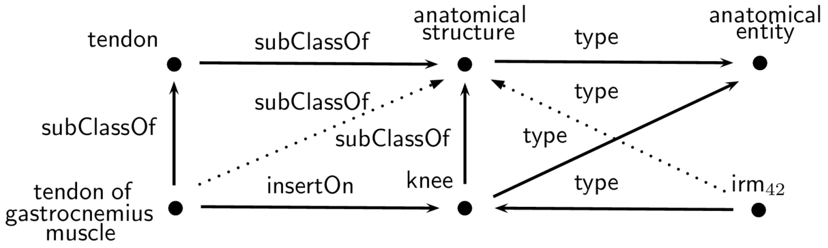
[13] fma.biostr.washington.edu

[14] www.mycorporisfabrica.org

[15] www.ihtsdo.org/snomed-ct

[16] www.dbpedia.org

For instance, the type edge connecting irm_{42} with knee, corresponds to the triplestore atom $(\mathsf{irm}_{42}, \mathsf{type}, \mathsf{knee})$, which is the standard RDF syntax for class membership.



(tendon, subClassOf, anatomical_structure)
(anatomical_structure, type, anatomical_entity)
(tendon_gastr._muscle, insertOn, knee)
(tendon_gastr._muscle, subClassOf, tendon)
(knee, subClassOf, anatomical_structure)
(knee, type, anatomical_entity)
(irm$_{42}$, type, knee)

Fig. 4. Triplestore D_1

A path $p_{(u_0,u_n)} = (u_0, v_1, u_1), (u_1, v_2, u_2), \dots, (u_{n-1}, v_n, u_n)$ is a sequence of atoms where each u_i, v_i are terms. The *length* of a path is the number of its atoms, here $|p_{(u_0,u_n)}| = n$.

We denote a rule by r and a set of rules by R. To illustrate, the rules for class subsumption
$r_1 : (x, \mathsf{type}, y), (y, \mathsf{subClassOf}, z) \to (x, \mathsf{type}, z)$
$r_2 : (x, \mathsf{subClassOf}, y), (y, \mathsf{subClassOf}, z) \to (x, \mathsf{subClassOf}, z)$
on D_1 entail that irm_{42} has type anatomical_structure, and that a subclass of this last one is tendon_gastr._muscle.

Datalog supports recursion by design. A rule r is said to be recursive if its conclusion unifies with one of its premises. In this work, we consider sets of rules where recursion is limited to recursive rules, like

$$r_1 : (x, \mathsf{hasPart}, y) \to (y, \mathsf{partOf}, x)$$
$$r_2 : (x, \mathsf{insertOn}, y), (y, \mathsf{partOf}, z) \to (x, \mathsf{insertOn}, z)$$
$$r_3 : (x, \mathsf{partOf}, y), (y, \mathsf{partOf}, z) \to (x, \mathsf{partOf}, z)$$

and, we exclude the presence of *indirect* recursion, in all cases where this involves *non-recursive rules*, like

$$r_4 : (x, \mathsf{contains}, y) \to (x, \mathsf{partOf}, y)$$
$$r_5 : (x, \mathsf{partOf}, y), (y, \mathsf{partOf}, z) \to (z, \mathsf{contains}, x)$$

This mild restriction on recursion is of practical relevance, as it is enjoyed by the most relevant RDFS rules, like the mutually recursive ones for domain and range.

$r_{\mathsf{dom}} : (x, \mathsf{domain}, z), (y, x, y') \to (y, \mathsf{type}, z)$
$r_{\mathsf{ran}} : (x, \mathsf{range}, z'), (y, x, y') \to (y', \mathsf{type}, z')$

Following Definition 1, the saturated RDF dataset obtained from D and the set of rules R, is defined as $\textsc{Sat}(D, R) = \{\mathsf{t} \in D' \mid D, R \vdash D'\}$.

We write $D, R \vdash p_{(u_0,u_n)}$ for the entailment of a path that holds if all path atoms are in $\textsc{Sat}(D, R)$.

Rule entailment, also referred as the immediate consequence operator for rules defines, by means of *semantic conditions*, when a Datalog rule r is entailed by a set R.

Definition 3 (Rule Entailment). *A rule r is entailed by a set R, denoted by $R \vdash r$, if for all triplestore D it holds that $\textsc{Sat}(D, r) \subseteq \textsc{Sat}(D, R)$. A set R' is entailed from R, denoted by $R \vdash R'$ when $R \vdash r$ for all $r \in R'$.*

Finally, knowledge base entailment, denoted by $\langle D, R\rangle \vdash \langle D', R'\rangle$, holds when $D, R \vdash D'$ and $R \vdash R'$.

4.1 Bounded-level Modules

We propose a novel semantics for bounded-level modules allowing to effectively control their size. We employ a notion of *level of detail* for modules in such a *deductive* setting. For example, a signature $(\,\mathsf{subClassOf}, \mathsf{partOf}\,)^3[\,\mathsf{eye}\,]$ limits the module-data extracted from a triplestore, by allowing to retrieve a description of all subclasses and subparts of the eye up to three levels.

A module is declared by means of a signature Σ of the form $\Sigma = (\,\mathsf{p}_1, \ldots, \mathsf{p}_n\,)^k[\,\mathsf{a}\,]$ where the constants $\mathsf{p}_1, \ldots, \mathsf{p}_n$ represent the *properties of interest* of the module, the constant a represents an *object of interest* of the module, and k is a positive integer denoting the *level of detail* of the module. An example of module signature is $(\,\mathsf{partOf}\,)^3[\,\mathsf{eye}\,]$. Intuitively, a module M induced by a signature Σ on a reference system $\langle D, R\rangle$ is a deductive triplestore $M{=}\langle D_M, R_M\rangle$ which is logically entailed by $\langle D, R\rangle$ and conforming to Σ, in the sense that all data and rule atoms employ the properties $\mathsf{p}_1, \ldots, \mathsf{p}_n$ *only*. Furthermore, to control the module size, the facts in M are restricted to the *paths* rooted at the object of interest a, of length bounded by k.

We say that an atom conforms to Σ, denoted by $(v_1, u, v_2) \mathrel{\overset{\circ}{\circ}} \Sigma$, if u is a property of Σ or $u \in \textsc{Vars}$. A set of atoms Δ conforms to Σ if all of its atoms do. Then, $\langle D, R\rangle$ conforms to Σ if so do D and R.

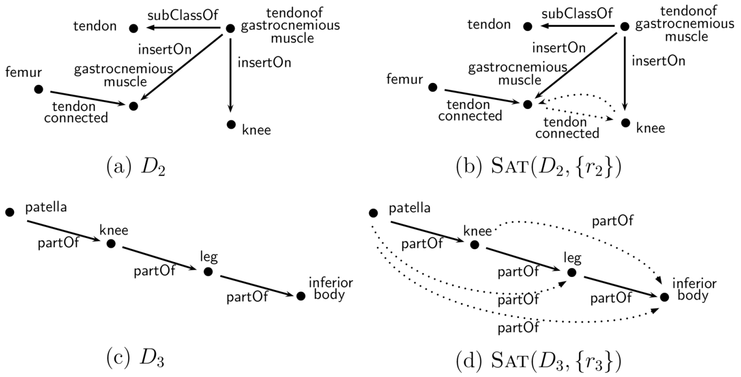
In Figure 5(c) it holds that $D_3 \mathrel{\overset{\circ}{\circ}} (\,\mathsf{partOf}, \mathsf{subClassOf}\,)^2[\,\mathsf{knee}\,]$. However, it does not hold that $D_3 \mathrel{\overset{\circ}{\circ}} (\,\mathsf{subClassOf}\,)^1[\,\mathsf{knee}\,]$.

Restricting the module paths is a way to effectively control the module size. Nevertheless, for the completeness of the module data, it is essential to guarantee that the module entails *all* of such bounded paths entailed by $\langle D, R\rangle$. In a deductive setting, adding new paths in the graph, defining properly D_M becomes challenging.

First, we observe that to avoid incomplete modules, the paths of D_M have to be drawn from $\textsc{Sat}(D, R)$. To see this, consider D_2 in Figure 5(a) and a rule inferring pairs of organs (y, z) physically connected by a tendon

$r_2 : (x, \mathsf{insertOn}, y), (x, \mathsf{insertOn}, z), (x, \mathsf{subClassOf}, \mathsf{tendon}) {\Rightarrow} (y, \mathsf{tendonConnected}, z)$

A user interested in the organs directly and indirectly connected to the femur of this



(a) D_2

(b) $\textsc{Sat}(D_2, \{r_2\})$

(c) D_3

(d) $\textsc{Sat}(D_3, \{r_3\})$

Fig. 5. Triplestore examples

triplestore can declare the module signature $\Sigma_2 = (\,\mathsf{tendonConnected}\,)^2[\,\mathsf{femur}\,]$. By restricting the module data D_M to the paths in D_2 of length bounded by 2 that are rooted at femur and that use the property tendonConnected only, we get:

$D_M = \{(\mathsf{femur}, \mathsf{tendonConnected}, \mathsf{gastroc.Muscle})\}$.

This dataset has however to be considered incomplete. As shown in Figure 5(b), the rule r_2 entails on D_2 also the fact

$(\mathsf{gastroc.Muscle}, \mathsf{tendonConnected}, \mathsf{knee})$.

This forms a path of length two together with the original triple

$(\mathsf{femur}, \mathsf{tendonConnected}, \mathsf{gastroc.Muscle})$,

that should be included in D_M. The example illustrates clearly that D_M depends from the rules in R.

However, taking into account *all* paths in $\textsc{Sat}(D, R)$ is not desirable for defining modules of bounded size. In some cases, the triples entailed by *recursive* rules may produce new edges in the data graph that behave like shortcuts between resources, thereby wasting the module parametricity. Consider D_3 in Figure 5(c) and the recursive rule r_3 defining the transitivity of partOf

$r_3 : (x, \mathsf{partOf}, y), (y, \mathsf{partOf}, z) \rightarrow (x, \mathsf{partOf}, z)$

The saturated triplestore $\textsc{Sat}(D_3, r_3)$ is depicted in Figure 5(d).

It contains $(\mathsf{patella}, \mathsf{partOf}, \mathsf{knee})$ but also

$(\mathsf{patella}, \mathsf{partOf}, \mathsf{leg})$

and $(\mathsf{patella}, \mathsf{partOf}, \mathsf{inferiorBody})$.

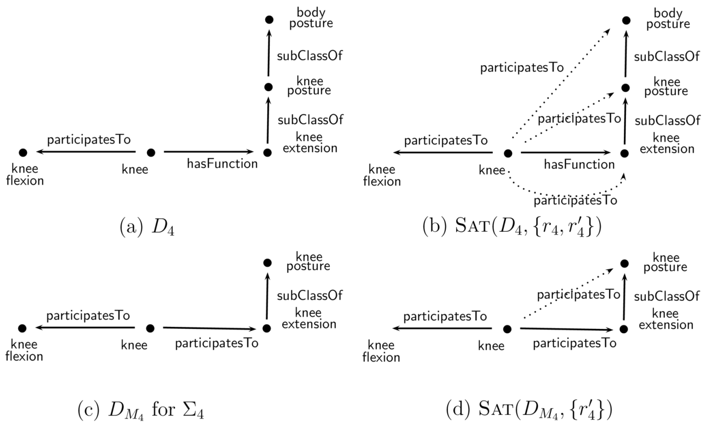
More generally, it contains all triples of the form $\mathrm{t}_{\mathsf{b}} = (\mathsf{patella}, \mathsf{partOf}, \mathsf{b})$ entailed by the transitivity of partOf. This means that if we take into account the recursive rule r_3 for defining the module paths, then all triples t_{b} are likely to be part of the module induced by signature $(\,\mathsf{partOf}\,)^1[\,\mathsf{knee}\,]$. This undermines the module parametricity because it retrieves all resources connected with knee regardless of the level of detail k.

Our solution to both keep into account implicit triples and make parametricity effective, is to define the module data as a subgraph of a *partially-saturated* triplestore obtained by applying non-recursive rules only, while fully *delegating* the recursive rules to the module rules. This leads to the following novel definition of module.

Definition 4 (Module).
Let $\langle D, R\rangle$ be a deductive triplestore and $\Sigma = (\,\mathsf{p}_1, \ldots, \mathsf{p}_n\,)^k[\,\mathsf{a}\,]$ a signature. Then, $M{=}\langle D_M, R_M\rangle$ is a module for Σ on $\langle D, R\rangle$ if

1. $\langle D_M, R_M\rangle \mathrel{\overset{\circ}{\circ}} \Sigma$
2. $\langle D, R\rangle \vdash \langle D_M, R_M\rangle$
3. *if* $p_{(\mathsf{a},\mathsf{b})} \mathrel{\overset{\circ}{\circ}} \Sigma$ *and* $|p_{(\mathsf{a},\mathsf{b})}| \leq k$ *then*
   (a) $D, R^{\mathsf{NonRec}} \vdash p_{(\mathsf{a},\mathsf{b})}$ *implies* $D_M, R_M \vdash p_{(\mathsf{a},\mathsf{b})}$
   (b) $D_M, R \vdash p_{(\mathsf{a},\mathsf{b})}$ *implies* $D_M, R_M \vdash p_{(\mathsf{a},\mathsf{b})}$

Point 1 and 2 of the definition state the well-formedness and the logical entailment of the modules, respectively. Point 3 is the crux of the definition. Property $3(a)$ says that every path rooted at a of k-bounded length and conforming to Σ, that is entailed by the *non-recursive* rules of the reference system R^{NonRec}, must also be inferable by M. Property $3(b)$ enforces that the module rules R_M infer the same paths conforming to Σ as the *whole set of rules* R, but only when applied to the module data D_M. In contrast with the spirit of previous approaches (e.g., [24]), our definition does not enforce that *every* fact in the signature entailed by the reference triplestore also belongs to the module. Relaxing the module conditions in this way allows to control the module size, and cope with recursive rules.



(a) D_4 (b) $\textsc{Sat}(D_4, \{r_4, r_4'\})$

(c) D_{M_4} for Σ_4 (d) $\textsc{Sat}(D_{M_4}, \{r_4'\})$

Fig. 6. Triplestore and module examples

To illustrate the definition, consider the triplestore D_4 of Figure 6(a) equipped with the rules below.

$r_4 : (x, \mathsf{hasFunction}, y) \rightarrow (x, \mathsf{participatesTo}, y)$
$r_4' : (x, \mathsf{participatesTo}, y), (y, \mathsf{subClassOf}, z) \rightarrow (x, \mathsf{participatesTo}, z)$

Figure 6(b) depicts $\textsc{Sat}(D_4, \{r_4, r'_4\})$. Consider now:

$\Sigma_4{=}(\,\mathsf{participatesTo}, \mathsf{subClassOf}\,)^2[\,\mathsf{knee}\,]$.

A module M_4 for Σ_4 contains all paths rooted at knee of length at most 2, employing participatesTo and subClassOf only. Note that if the recursive rule r'_4 is considered, then the triple $\mathrm{t}_1{=}(\mathsf{knee}, \mathsf{participatesTo}, \mathsf{bodyPosture})$ is included in the module dataset, which is not desirable. In contrast, $\mathrm{t}_2 = (\mathsf{knee}, \mathsf{participatesTo}, \mathsf{kneePosture})$ is expected to be in a module for the signature Σ_4. A structure satisfying Definition 4 is $M_4 = \langle D_{M_4}, R_{M_4}\rangle$ with D_{M_4} depicted in Figure 6(c) and $R_{M_4} = \{r'_4\}$. Note that t_2 is not explicitly in the module dataset D_{M_4} but can be inferred by r'_4 as shown in Figure 6(d).

Next, we present two algorithms for extracting module data and rules compliant with this novel semantics.

4.2 Extracting Module Data

The extraction of the module dataset can be done by leveraging on the evaluation of Datalog queries and implemented on top of existing engines. Given a module signature $\Sigma = (\,\mathsf{p}_1, \ldots, \mathsf{p}_n\,)^k[\,\mathsf{a}\,]$, the Datalog program Π_Σ below computes all paths rooted at a, of length bounded by k, and built on the properties of interest of Σ. It does so, in the extension of the relation m, starting from a triplestore modeled with a single relation t.

$$\Pi_\Sigma{=}\begin{cases} \quad t(\mathsf{a}, \mathsf{p}_i, x) & \rightarrow m^1(\mathsf{a}, \mathsf{p}_i, x) \\ m^j(x_1, y_1, x)\,, t(x, \mathsf{p}_i, y) & \rightarrow m^{j+1}(x, \mathsf{p}_i, y) \\ \quad m^j(x, y, z) & \rightarrow m(x, y, z) \end{cases}$$

An instance of the rules is included for each $i = 1..n$ and $j = 1..k$. Π_Σ is a non-recursive set of rules of size $O(nk)$ that can always be evaluated in at most k steps. Then, to infer all paths of bounded length entailed by non-recursive rules of a reference system, the set Π_Σ is evaluated together with R^{NonRec}. As a result, the union $\Pi_\Sigma \cup R^{\mathsf{NonRec}}$ gives a non-recursive set of rules that can be evaluated in LOGSPACE data-complexity. The completeness of module data extraction follows from the completeness of Datalog query evaluation. Below, we write $Q_m(D, \Pi_\Sigma{\cup}R^{\mathsf{NonRec}})$ for the answer set of the evaluation of the Datalog program $\Pi_\Sigma{\cup}R^{\mathsf{NonRec}}$ defining the relation m, on top of the dataset D. This constitutes the module data D_M.

Theorem 1 (Module Data Extraction) *For all path $p_{(\mathsf{a},\mathsf{b})} \overset{\circ}{\circ} \Sigma$ with $|p_{(\mathsf{a},\mathsf{b})}| \leq k$ we have $D, R^{\mathsf{NonRec}} \vdash p_{(\mathsf{a},\mathsf{b})}$ if and only if $p_{(\mathsf{a},\mathsf{b})} \in Q_m(D, \Pi_\Sigma{\cup}R^{\mathsf{NonRec}})$.*

4.3 Extracting Module Rules

We now present an algorithm for module rule extraction that, together with the dataset extracted in the previous section, yields a module compliant with our semantics.

By Definition 4, a module is constituted of rules *entailed* by that of the reference system, and built on the properties of interest *only*. As the properties of interest of a module may restrict those employed by a reference system, the module rules cannot be just a subset of the original ones. Rule extraction is thus performed by an *unfolding* algorithm, that proceeds by replacing the premises of a rule with that of another one, until obtaining a set conforming to the signature. To illustrate, consider $\Sigma = (\,\mathsf{p},\mathsf{q}\,)^k[\,\mathsf{a}\,]$ and the rules below.

$r_1 : (x, \mathsf{q}, y), (y, \mathsf{partOf}, x) \rightarrow (x, \mathsf{q}, y)$
$r_2 : (x, \mathsf{p}, y) \rightarrow (x, \mathsf{partOf}, y)$

Although the rule r_1 does not conform to Σ, it can be unfolded with r_2 so as to obtain a module rule. As the atom (y, partOf, x) in the body of r_1 unifies with the conclusion of r_2, it can be replaced by (y, p, x), so as to get the rule $\bar{r} = (x, \mathsf{q}, y), (y, \mathsf{p}, x) \rightarrow (x, \mathsf{q}, y)$. Rule $\bar{r}$ is called an *unfolding* of r_1 with r_2.

In the above example, one unfolding step is enough to have a rule $\bar{r}$ that is conform to the module signature and that, by construction, is entailed by $\{r_1, r_2\}$. It is easy to see that this can be generalized, and that rules belonging to unfoldings of a set of rules R are entailed by R. However, in presence of recursive rules the set of unfoldings of a rule may be infinite, as illustrated below.

Example 2 *Consider $\Sigma = (\,\mathsf{p},\mathsf{q}\,)^3[\,\mathsf{a}_1\,]$ and R with*

$r_1 : (x, \mathsf{partOf}, y) \rightarrow (x, \mathsf{q}, y)$
$r_2 : (x, \mathsf{partOf}, y), (y, \mathsf{partOf}, z) \rightarrow (x, \mathsf{partOf}, z)$
$r_3 : (x, \mathsf{p}, y) \rightarrow (x, \mathsf{partOf}, y)$

Here, r_1 can be unfolded with r_2 and r_3, thus obtaining

$\bar{r} : (x_1, \mathsf{p}, x_2), (x_2, \mathsf{p}, x_3) \rightarrow (x_1, \mathsf{q}, x_3)$

However, there exist infinitely many unfoldings of rule r_2 with itself that yield expressions of the form $(x_1, \mathsf{p}, x_2), (x_2, \mathsf{p}, x_3), (x_3, \mathsf{p}, x_4) \rightarrow (x_1, \mathsf{q}, x_4)$ that use any finite sequence of variables $x_1, \ldots, x_n$. This set of unfoldings cannot be strictly speaking a set of triplestore or module rules, because it is *infinite*.

```
Algorithm 2: MRE($N_{ToUnfold}, R_{ToApply}, \Sigma$)
for all $r_1 \in N_{ToUnfold}$
    if $r_1 \overset{\circ}{\circ} \Sigma$ then:
        $R_M \leftarrow r_1$
        remove $r_1$ from $R_{ToApply}$
    else:
        for all $r_2 \in R_{ToApply}$ s.t. $r_1 \neq r_2$
            for all $r \in$ RuleUnfolding$(r_1, r_2)$
                if $r \overset{\circ}{\circ} \Sigma$ then: $R_M \leftarrow r$
                $R_M \leftarrow$ MRE$(\{r\}, R_{ToApply} \setminus \{r, r_2\}, \Sigma)$
return $R_M$
```

To avoid ending up with infinite sets of module rules, we devised an unfolding algorithm based on a *breadth-first* strategy. Algorithm MRE (Algorithm 2) performs Module Rules Extraction. It takes as input a set of rules to be unfolded $N_{ToUnfold}$, a set of rules to be used for the unfolding R_{ToApply}, and a signature Σ. Given a deductive triplestore $\langle D, R\rangle$ the first call to the algorithm is MRE($N_{\mathit{ToUnfold}}, R, \Sigma$). The set $N_{\mathit{ToUnfold}} \subseteq R$ is constituted of all rules $r \in R$ that conclude on a property of interest, that is *head*$(r) \overset{\circ}{\circ} \Sigma$. Any rule belonging to N_{ToUnfold} (whose premises use properties that are not in Σ) is unfolded in a breadth-first fashion until no rule in R_{ToApply} can be applied. All rules in R are considered for unfolding ($R_{\mathit{ToApply}} = R$). Procedure *RuleUnfolding*(r_1, r_2) progressively unfolds each *subset* of atoms in the body of r_1 that unify with the conclusion of r_2. For example, the three breadth-first unfoldings of $r_1 : (x, \mathsf{p}, y), (x, \mathsf{p}, z) \rightarrow (x, \mathsf{p}, y)$ with $r_2 : (x, \mathsf{partOf}, y) \rightarrow (x, \mathsf{p}, y)$ are

$\bar{r}_3 : (x, \mathsf{p}, y), (x, \mathsf{partOf}, z) \rightarrow (x, \mathsf{p}, y)$
$\bar{r}_4 : (x, \mathsf{partOf}, y), (x, \mathsf{p}, z) \rightarrow (x, \mathsf{p}, y)$
$\bar{r}_5 : (x, \mathsf{partOf}, y), (x, \mathsf{partOf}, z) \rightarrow (x, \mathsf{p}, y)$

Note that a rule is never unfolded with itself by the algorithm (thus avoiding a depth-first fashion). The fact that r_2 used for the unfolding is discarded from R_{ToApply} (line 10) ensures the termination of the extraction procedure, even in the presence of recursive rules.

Theorem 3 (Rule Extraction Algorithm) *Let R be a set of rules and Σ a module signature. Algorithm MRE always terminates in $O(2^{|R|\times|r|})$ and produces a set of rules R_M conforming to Σ such that for all r $\overset{\circ}{\circ}$ Σ it holds*

$$R_M \vdash r \quad \textit{implies} \quad R \vdash r \quad (\textsc{Soundness})$$

Furthermore, when $R^{\mathsf{Rec}} \overset{\circ}{\circ} \Sigma$ we also have

$$R \vdash r \quad \textit{implies} \quad R_M \vdash r \quad (\textsc{Completeness})$$

Algorithm MRE is sound, in the sense that it computes a set of rules entailed by R. Furthermore, for the case where all recursive rules in R conform to Σ, the algorithm is also complete, in the sense that it produces a set of rules R_M that entails all rules R can entail on the properties of Σ. As a consequence, any dataset D_M (computed as for Theorem 4) paired with R_M constitutes a module meeting Definition 4, and in particular the point 3(b). If this condition does not hold, module extraction may be incomplete. To see this, consider again $\langle D, R\rangle$ of Example 2 with $D = \{(\mathsf{a}_1, \mathsf{p}, \mathsf{a}_2),\ (\mathsf{a}_2, \mathsf{p}, \mathsf{a}_3), (\mathsf{a}_3, \mathsf{p}, \mathsf{a}_4)\}$. Recall that $\Sigma = (\,\mathsf{p}, \mathsf{q}\,)^3[\,\mathsf{a}_1\,]$, and then notice that the recursive rule $r_2 \not\overset{\circ}{\circ} \Sigma$. Here, module data extraction yields $D_M = D$. Observe now that the atom $(\mathsf{a}_1, \mathsf{q}, \mathsf{a}_4)$ belongs to $\textsc{Sat}(D_M, R)$. As MRE outputs the set $R_M = \{(x, \mathsf{p}, y), (y, \mathsf{p}, z) \rightarrow (x, \mathsf{q}, z)\}$, the triple $(\mathsf{a}_1, \mathsf{q}, \mathsf{a}_4)$ does not belong to $\textsc{Sat}(D_M, R_M)$, while it should. Hence, $\langle D_M, R_M\rangle$ does not satisfy Definition 4.

Surprisingly enough, this case of incompleteness *is independent of algorithm MRE*. In fact, when R includes recursive rules that do not conform to Σ, it does not exist an algorithm that outputs a finite set of rules R_M such that $R \vdash r$ implies $R_M \vdash r$, for all $r \overset{\circ}{\circ} \Sigma$. As Example 2 illustrates, the extracted R_M must mimic an *infinite* set of rules of the form $(x_1, \mathsf{p}, x_2), (x_2, \mathsf{p}, x_3)\ldots(x_{n-1}, \mathsf{p}, x_n) \to (x_1, \mathsf{q}, x_n)$. One may think of capturing this infinite set by adding a recursive rule $r_{\mathsf{p}} : (x, \mathsf{p}, y), (y, \mathsf{p}, z) \to (x, \mathsf{p}, z)$ together with $\bar{r} : (x_1, \mathsf{p}, x_2), (x_2, \mathsf{p}, x_3) \to (x_1, \mathsf{q}, x_3)$. However, adding this recursive rule makes infer triples using p that are not entailed by the reference system, thereby violating point 2 of Definition 4. We can also ask whether this infinite set of rules can be reduced to a finite set that directly depends on k. Unfortunately, the answer is negative. Furthermore, it is unpractical for real systems to consider a specific module data D_M and bound by $O(|D_M|)$ the number of self-unfolding of a recursive rule during extraction, as this can output an unmanageable set of rules, that are (still) not robust to updates. Therefore, understanding when algorithm MRE is complete is key for module extraction.

This kind of unfolding issues have also been recognized and studied by earlier works on the optimization of recursive Datalog [28].

Finally, note that Theorem 3 is actually stronger than what required by Definition 4, because (i) it is based on *semantic* conditions and therefore it holds for any rule r entailed by R (unfoldings are just a particular case) and (ii) it is independent from the module data, and thus suitable for other module semantics.

A characterization of the whole module extraction task follows as a corollary of Theorems 1 and 3.

4.4 Experiments

We implemented bounded-level module extraction on top of Jena 2.11.2 TDB, and compared it against two related approaches to show its benefits in terms of flexibility and succinctness of the extracted modules. We considered the following three Semantic Web datasets.

MyCF	0.5M triples	11 domain-specific rules
GO	1M triples	15 domain-specific rules
Yago2*	14M triples	6 RDFS rules

Yago2* is the union of Yago2Taxonomy, Yago2Types and Yago2Facts datasets. We sampled classes and properties from these ontologies, and combined them to obtain a set of signatures used to run module extraction. We considered 2500 MyCF ontology classes combined with 20 subsets of its properties, of size 1-4. For the GO ontology (`www.geneontology.org`), we sampled 350 classes and 12 property sets (size 1-4). Since Yago knowledge is more diverse than a domain-specific ontology, to avoid empty modules we first selected three groups of properties that are frequently used together, and then subset them (size 2, 4, 6). We tested 100 Yago resources for each group. Finally, we made k ranging over $\{1, 2, 3, 5, 10\}$.

Closest competitor approaches Relevant methods to our work are Traversal Views [35] and Locality-based modules [24]. Traversal Views (TV) compute a bounded-level view of an RDF database, in the same spirit as our approach. This method does not support inference rules, and it does not give any guarantee about extracted modules. In practice, in the presence of rules, a traversal view may miss relevant triples. Locality-Based (LB) module extraction computes a conservative extension of an ontology by checking logical conditions on its schema. In contrast with our method, it cannot modularize untyped RDF data and, because it enforces strong logical guarantees on a module, it cannot control a priori its size.

Results of module data extraction Figures 7 and 8 report on the size of bounded-level modules, compared with those of TV and LB.The graphs show the average number of triples, for modules grouped by the same number of properties and k value, in logarithmic scale. In Figure 9 we report the test on Yago2 with our approach, since LB does not support this RDF dataset.

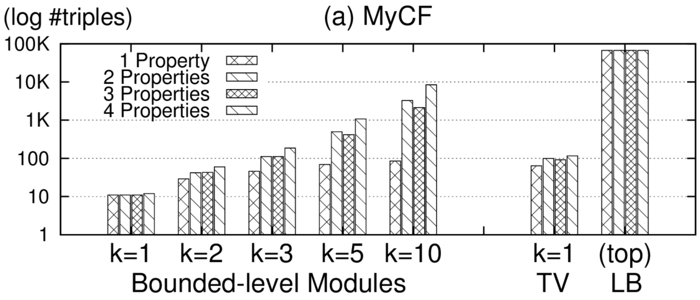


Fig. 7. Size of Extracted Modules from MyCF

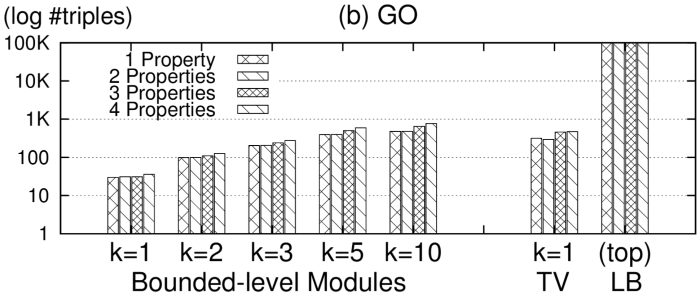


Fig. 8. Size of Extracted Modules from GO

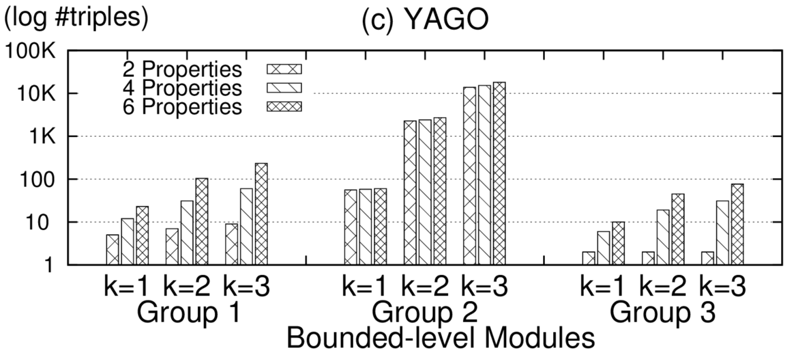


Fig. 9. Size of Extracted Modules from Yago2

As expected, the succinctness of bounded-level modules depends on k. The transitivity of the properties declared in the signature also has an impact. This is evident with Yago2 in Figure 9. Group 2 has properties inherently transitive (isLocatedIn, isConnectedWith) dominating for example (created, owns) in group 1 and (hasGender, isAffiliatedTo) in group 3. Hence, bounded-level modules can be very helpful to control the data succinctness with transitive properties.

Being TV unaware of rules, it may miss relevant data when implicit triples are not considered. We tested this claim, over the *non-saturated* MyCF ontology. Indeed, 42% (15072/35740) of the (non-empty) modules extracted by TV were missing relevant triples wrt our approach, as some subproperty rules were not evaluated. To overcome this limitation, we tested TV over the *saturated* MyCF. For concision, in Figure 7 we report only the minimal level of detail ($k = 1$). This already outlines a lower bound for the module size. As we can see, $k = 1$ already produces fairly larger modules than our approach. This is because of the MyCF rules for transitivity and property-chains. Increasing k gives modules of size in the order of the saturated triplestore. The same discussion holds for GO in Figure 8. LB extraction for *top-locality* modules has been tested thanks to the available prototype [17]. For MyCF and GO, it outputs almost the whole ontology (Figures 7 and 8). This is due to ontology axioms that cannot be ignored for the logical completeness of the method.

[17] www.cs.ox.ac.uk/ isg/tools/ModuleExtractor/

5 Rule-based integration of heterogeneous data and models [36, 37]

Computer modeling and simulation of the human body is becoming a critical and central tool in medicine but also in many other disciplines, including engineering, education, entertainment. Multiple models have been developed, for applications ranging from medical simulation to video games, through biomechanics, ergonomics, robotics and CAD, to name only a few. However, currently available anatomical models are either limited to very specific areas or too simplistic for most of the applications.

For anatomy, the reference domain ontology is the Foundational Model of Anatomy (FMA [38]) which is a comprehensive description of the structural organization of the body. Its main component is a taxononomy with more then 83000 classes of anatomical structures from the macromolecular to the macroscopic levels. The FMA symbolically represents the structural organization of the human body. One important limitation of the state-of-the-art available ontologies is the lack of explicit relation between anatomical structures and their functions. Yet, human body modeling relies on morphological components on the one hand and functional and process descriptions on the other hand. The need for a formal description of anatomical functions has been outlined in [30], with some guidelines for getting a separate ontology of anatomical functions based on an ontological analysis of functions in general formal ontologies such as GFO [27] or Dolce [33]. Complex 3D graphic models are present in more and more application software but they are not explicitly related to the (anatomical) entities that they represent making difficult the interactive management of these complex objects.

Our approach for supporting efficient navigation and selection of objects in 3D scenes of human body anatomy is to make explicit the anatomic and functional semantics of 3D objects composing a complex 3D scene through a symbolic and formal representation that can be queried on demand. It has been implemented in *My Corporis Fabrica* (MyCF), which realizes a rule-based integration of three types of models of anatomy: structural, functional model and 3D models. The added-value of such a declarative approach for interactive simulation and visualization as well as for teaching applications is to provide new visualization/selection capabilities to manage and browse 3D anatomical entities based on the querying capabilities incorporated in MyCF.

The core of MyCF is a comprehensive anatomical ontology, the novelty of which is to make explicit the links between anatomical entities, human body functions, and 3D graphic models of patient-specific body parts. It is equipped with inference-based query answering capabilities that are particularly interesting for different purposes such as:

- automatic verification of the anatomical validity of 3D models. Indeed, it is important to select the correct set of anatomical entities that participates to a simulation, e.g. a simulation of movements where the correct bones, muscles, ligaments, . . . , are required to set up all the 3D and mechanical simulation parameters. These requirements are very close to the selection requirements described in the 'Background' section. They can be regarded as equivalent to a selection operator;
- automatic selection and display of anatomical entities within a 3D scene. Anatomical entities can vary largely in size, can be very close to each other or even hidden by other anatomical entities. The use of geometric means to select useful sets of en-

tities is not suited whereas inference-based queries using human body functions can provide much more suited means. Such selection capabilities are particular relevant for diagnosis for instance;

- training students on anatomical entities participating to a certain body function. Here again, this purpose is close to that of selection functions where the connection between function and anatomical entities provides new means to browse and highlight features of anatomical structures accessible in 3D.

The current version of the ontology contains almost 74000 classes and relations as well as 11 rules stored in a deductive RDF triple store using a Sesame server, and that can be queried with a remote-access facility via a web server[18]. The ontology can be easily updated, just by entering or deleting triples and/or by modifying the set of rules, without having to change the reasoning algorithmic machinery used for answering queries. It is the strength of a declarative approach that allows a fine-grained domain-specific modeling and the exploitation of the result by a generic (domain-independent) reasoning algorithm.

MyCF features three distinct taxonomies linked by relations and rules:

- Anatomical entities, such as *knee*, *shoulder*, and *hand*, denote parts of the human body, and give a formal description of canonical anatomy;
- Functional entities, such as *gait*, *breath*, and *stability*, denote the functions of the human body, and are the fundamental knowledge to explain the role of each anatomical entity;
- Finally, 3D scenes with entities such as *3D-object*, *3D-scene* define the content required to get 3D views of patient-specific anatomical entities described by 3D graphical models related to anatomical entities.

Figure 10 shows an extract of this integrated ontology, in which the green classes refer to the 3D models, the pink classes to the structural model and blue classes to the functional entities.

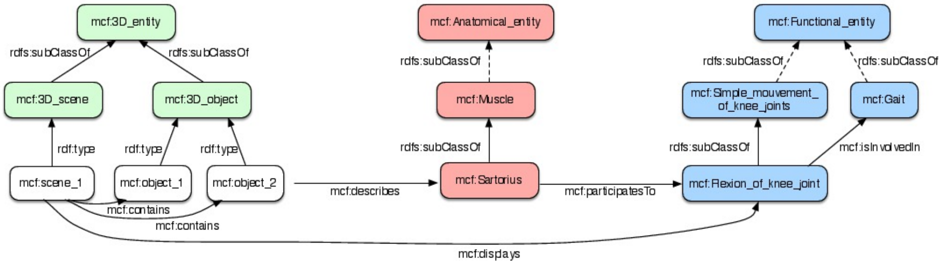


Fig. 10. The general structure of MyCF integrated ontology (extract)

The inference rules of MyCF express complex connections between relations, within or across the three taxonomies. For instance, the following rules express connections

[18] `http://mycorporisfabrica.org/mycf/`

that hold in anatomy between the relations rdfs:subClassOf and mcf:InsertOn, but also between rdfs:subClassOf and mcf:IsInvolvedIn , rdfs:subClassOf and mcf:participatesTo, mcf:participatesTo and mcf:IsInvolvedIn , mcf:PartOf and mcf:InsertOn respectively. The first rule says that if a given class representing an anatomical entity ?a (e.g., Sartorius) is a subclass of an anatomical entity ?c (e.g., Muscle) that is known to be inserted on an anatomical entity ?b (e.g., Bone), then ?a is inserted on ?b (Sartorius inserts on a Bone).

(?a rdfs:subClassOf ?c) , (?c mcf:InsertOn ?b) → (?a mcf:InsertOn ?b)

(?a mcf:IsInvolvedIn ?c), (?c rdfs:subClassOf ?b) → (?a mcf:IsInvolvedIn ?b)

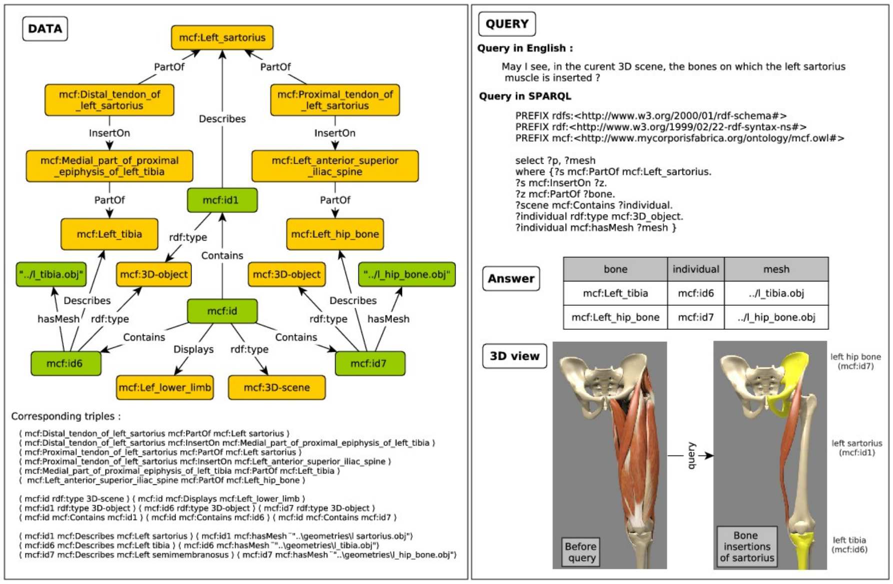
(?a mcf:participatesTo ?c) , (?c rdfs:subClassOf ?b) → (?a mcf:participatesTo ?b)

(?a mcf:participatesTo ?c), (?c mcf:IsInvolvedIn ?b) → (?a mcf:participatesTo ?b)

(?a mcf:InsertOn ?c), (?c mcf:PartOf ?b) → (?a mcf:InsertOn ?b)

The following rule crosses the anatomy domain and the 3D domain and expresses that the conventional color for visualizing bones in anatomy is yellow:

(?x rdf:type 3D-object), (?x mcf:Describes ?y) , (?y rdfs:subClassOf Bone)
→ (?x mcf:hasColour yellow)



Corresponding triples :

```
( mcf:Distal_tendon_of_left_sartorius mcf:PartOf mcf:Left sartorius )
( mcf:Distal_tendon_of_left_sartorius mcf:InsertOn mcf:Medial_part_of_proximal_epiphysis_of_left_tibia )
( mcf:Proximal_tendon_of_left_sartorius mcf:PartOf mcf:Left sartorius )
( mcf:Proximal_tendon_of_left_sartorius mcf:InsertOn mcf:Left_anterior_superior_iliac_spine )
( mcf:Medial_part_of_proximal_epiphysis_of_left_tibia mcf:PartOf mcf:Left_tibia )
(  mcf:Left_anterior_superior_iliac_spine mcf:PartOf mcf:Left_hip_bone )

( mcf:id rdf:type 3D-scene ) ( mcf:id mcf:Displays mcf:Left_lower_limb )
( mcf:id1 rdf:type 3D-object ) ( mcf:id6 rdf:type 3D-object ) ( mcf:id7 rdf:type 3D-object )
( mcf:id mcf:Contains mcf:id1 ) ( mcf:id mcf:Contains mcf:id6 ) ( mcf:id mcf:Contains mcf:id7 )

( mcf:id1 mcf:Describes mcf:Left sartorius ) ( mcf:id1 mcf:hasMesh ˜"..\geometries\l sartorius.obj")
( mcf:id6 mcf:Describes mcf:Left tibia ) ( mcf:id6 mcf:hasMesh ˜"..\geometries\l_tibia.obj")
( mcf:id7 mcf:Describes mcf:Left semimembranosus ) ( mcf:id7 mcf:hasMesh ˜"..\geometries\l_hip_bone.obj")
```

Query in English :

May I see, in the curent 3D scene, the bones on which the left sartorius muscle is inserted ?

Query in SPARQL

```
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX mcf:<http://www.mycorporisfabrica.org/ontology/mcf.owl#>

select ?p, ?mesh
where {?s mcf:PartOf mcf:Left_sartorius.
?s mcf:InsertOn ?z.
?z mcf:PartOf ?bone.
?scene mcf:Contains ?individual.
?individual rdf:type mcf:3D_object.
?individual mcf:hasMesh ?mesh }
```

bone	individual	mesh
mcf:Left_tibia	mcf:id6	../l_tibia.obj
mcf:Left_hip_bone	mcf:id7	../l_hip_bone.obj

Fig. 11. Illustration of ontology-based querying and visualization using MyCF

Figure 11 illustrates a complete example from query to 3D visualization. Data are presented as a graph with corresponding RDF triples on the bottom. The query is explained in English and translated in SPARQL. The answers are used to select and highlight corresponding 3D models in the 3D scene.

We have extended this rule-based approach for 3D spatio-temporal modeling of human enbryo development in [37]. It results in a unified description of both the knowledge of the organs evolution and their 3D representations enabling to visualize dynamically the embryo evolution.

In an ongoing work, following a similar methodology for ontology-based integration of data extracted from several heterogeneous sources, we are developing Onto-SIDES to offer personalized and interactive services for student progress monitoring on top of the national e-learning and evaluation platform of French medical schools.

6 Conclusion

We have shown that Datalog rules on top of RDF triples provides a good trade-off between expressivity and scalability for reasoning in the setting of Linked Data. It would be worthwhile to investigate the usefulness in practice and the scalability of the Datalog extension proposed in [8] allowing for value invention and stratified negation.

References

[1] Serge Abiteboul, Zoë Abrams, Stefan Haar, and Tova Milo, 'Diagnosis of asynchronous discrete event systems: datalog to the rescue!', in *Proceedings of the Twenty-fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 13-15, 2005, Baltimore, Maryland, USA*, pp. 358–367. ACM, (2005).

[2] Serge Abiteboul, Richard Hull, and Victor Vianu, *Foundations of Databases*, Addison-Wesley, 1995.

[3] Mustafa Al-Bakri, Manuel Atencia, Jerome David, Steffen Lalande, and Marie-Christine Rousset, 'Uncertaintly-sensitive reasoning for inferring sameas facts in linked data', in *Proceedings of the European Conference on Artificial Intelligence (ECAI 2016), August 2016, The Hague, Netherland*, (2016).

[4] Mustafa Al-Bakri, Manuel Atencia, Steffen Lalande, and Marie-Christine Rousset, 'Inferring same-as facts from linked data: an iterative import-by-query approach', in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pp. 9–15. AAAI Press, (2015).

[5] Dean Allemang and James Hendler, *Semantic Web for the Working Ontologist: Modeling in RDF, RDFS and OWL*, Morgan Kaufmann, 2011.

[6] Antoine Amarilli, Pierre Bourhis, and Pierre Senellart, 'Provenance circuits for trees and treelike instances', in *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pp. 56–68. Springer, (2015).

[7] Arvind Arasu, Christopher Ré, and Dan Suciu, 'Large-scale deduplication with constraints using dedupalog', in *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pp. 952–963. IEEE Computer Society, (2009).

[8] Marcelo Arenas, Georg Gottlob, and Andreas Pieris, 'Expressive languages for querying the semantic webl', in *Proceedings of the International Conference on Principles Of Database Systems (PODS 2014)*, (2014).

[9] Manuel Atencia, Mustafa Al-Bakri, and Marie-Christine Rousset, 'Trust in networks of ontologies and alignments', *Journal of Knowledge and Information Systems*, (2013). Doi: 10.1007/s10115-013-0708-9.

[10] Manuel Atencia, Jérôme David, and Jérôme Euzenat, 'Data interlinking through robust linkkey extraction', in *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pp. 15–20. IOS Press, (2014).

[11] Manuel Atencia, Jérôme David, and François Scharffe, 'Keys and pseudo-keys detection for web datasets cleansing and interlinking', in *Knowledge Engineering and Knowledge Management - 18th International Conference, EKAW 2012, Galway City, Ireland, October 8-12, 2012. Proceedings*, volume 7603 of *LNCS*, pp. 144–153. Springer, (2012).

[12] Matthias Bröcheler, Lilyana Mihalkova, and Lise Getoor, 'Probabilistic similarity logic', in *UAI 2010, Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, July 8-11, 2010*, pp. 73–82. AUAI Press, (2010).

[13] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz, 'A general datalog-based framework for tractable query answering over ontologies', *Journal of Web Semantics*, **14**, 57–83, (2012).

[14] 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', *Journal of Automated Reasoning*, **39**(3), 385–429, (2007).

[15] A.K. Chandra and P.M. Merlin, 'Optimal implementation of conjunctive queries in relational databases', in *Proceedings of the 9th ACM Symposium on Theory of Computing*, pp. 77–90, (1975).

[16] Peter Christen, *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*, Data-Centric Systems and Applications, Springer, 2012.

[17] Nilesh Dalvi and Dan Suciu, 'The dichotomy of probabilistic inference for unions of conjunctive queries.', *Journal of the ACM*, **59**(6), 17–37, (2012).

[18] Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati, 'Higher-order description logics for domain metamodeling', in *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI-11)*, (2011).

[19] Jérôme Euzenat and Pavel Shvaiko, *Ontology Matching, Second Edition*, Springer, 2013.

[20] Alfio Ferrara, Andriy Nikolov, and François Scharffe, 'Data linking for the semantic web', *International Journal on Semantic Web and Information Systems*, **7**(3), 46–76, (2011).

[21] Charles Forgy, 'Rete: A fast algorithm for the many patterns/many objects match problem', *Artificial Intelligence*, **19**(1), 17–37, (1982).

[22] Norbert Fuhr, 'Probabilistic models in information retrieval', *The Computer Journal*, **3**(35), 243–255, (1992).

[23] Norbert Fuhr, 'Probabilistic datalog: implementing logical information retrieval for advanced applications', *Journal of the American Society for Information Science*, **51**(2), 95–110, (2000).

[24] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler, 'Modular reuse of ontologies: Theory and practice', *Journal of Artificial Intelligence Research (JAIR-08)*, (2008).

[25] Bernardo Cuenca Grau and Boris Motik, 'Reasoning over ontologies with hidden content: The import-by-query approach', *Journal of Artificial Intelligence Research (JAIR)*, **45**, 197–255, (2012).

[26] Tom Heath and Christian Bizer, *Linked Data : Evolving the Web into a Global Data Space*, Morgan and Claypool, 2011.

[27] Heinrich Herre, ‘General formal ontology (gfo) : A foundational ontology for conceptual modelling’, in *Theory and Applications of Ontology*, eds., Roberto Poli and Leo Obrst, volume 2, Springer, Berlin, (2010).

[28] Gerd G. Hillebrand, Paris C. Kanellakis, Harry G. Mairson, and Moshe Y. Vardi, ‘Undecidable boundedness problems for datalog programs’, *Journal of Logic Programming (JLP-95)*, (1995).

[29] Knut Hinkelmann and Helge Hintze, ‘Computing cost estimates for proof strategies’, in *Extensions of Logic Programming, 4th International Workshop, ELP’93, St. Andrews, U.K., March 29 - April 1, 1993, Proceedings*, volume 798 of *LNCS*, pp. 152–170. Springer, (1993).

[30] R. Hoehndorf, A.-C. Ngonga Ngomo, and J. Kelso, ‘Applying the functional abnormality ontology pattern to anatomical functions’, *Journal of Biomedical Semantics*, **1**(4), 1–15, (2010).

[31] Aidan Hogan, Antoine Zimmermann, Jürgen Umbrich, Axel Polleres, and Stefan Decker, ‘Scalable and distributed methods for entity matching, consolidation and disambiguation over linked data corpora’, *Journal of Web Semantics*, **10**, 76–110, (2012).

[32] Boris Konev, Carsten Lutz, Dirk Walther, and Frank Wolter, ‘Semantic modularity and module extraction in description logics’, in *Proceedings of the European Conference on Artificial Intelligence (ECAI-08)*, (2008).

[33] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, A. Oltramari, and L. Schneider, ‘WonderWeb deliverable D17. the WonderWeb library of foundational ontologies and the DOLCE ontology’, Technical report, ISTC-CNR, (2002).

[34] Axel-Cyrille Ngonga Ngomo and Sören Auer, ‘LIMES - A time-efficient approach for large-scale link discovery on the web of data’, in *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pp. 2312–2317. IJCAI/AAAI, (2011).

[35] Natalya Fridman Noy and Mark A. Musen, ‘Specifying ontology views by traversal’, in *Proceedings of the International Semantic Web Conference (ISWC-04)*, (2004).

[36] Olivier Palombi, Federico Ulliana, Vincent Favier, Jean-Claude Leon, and Marie-Christine Rousset, ‘My corporis fabrica: an ontology-based tool for reasoning and querying on complex anatomical models’, *Journal of Biomedical Semantics (JOBS 2014)*, **5**(20), (2014).

[37] Pierre-Yves Rabattu, Benoit Masse, Federico Ulliana, Marie-Christine Rousset, Damien Rohmer, Jean-Claude Leon, and Olivier Palombi, ‘My corporis fabrica embryo: An ontology-based 3d spatio-temporal modeling of human embryo development’, *Journal of Biomedical Semantics (JOBS 2015)*, **6**(36), (2015).

[38] Cornelius Rosse and Jose L. V. Mejino, ‘A reference ontology for biomedical informatics: the foundational model of anatomy’, *J. of Biomedical Informatics*, **36**, 500, (2003).

[39] Marie-Christine Rousset and Federico Ulliana, ‘Extractiong bounded-level modules from deductive triplestores’, in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. AAAI Press, (2015).

[40] Fatiha Saïs, Nathalie Pernelle, and Marie-Christine Rousset, ‘Combining a logical and a numerical method for data reconciliation’, *Journal on Data Semantics*, **12**, 66–94, (2009).

[41] Parag Singla and Pedro M. Domingos, ‘Entity resolution with markov logic’, in *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China*, pp. 572–582. IEEE Computer Society, (2006).

[42] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum, ‘Yago: A core of semantic knowledge’, in *Proceedings of the World Wide Web Conference (WWW-07)*, (2007).

[43] Dan Suciu, Dan Olteanu, Christopher Ré, and Chistoph Koch, *Probabilistic Databases*, Morgan & Claypool, 1995.

[44] Danai Symeonidou, Vincent Armant, Nathalie Pernelle, and Fatiha Saïs, 'Sakey: Scalable almost key discovery in RDF data', in *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, volume 8796 of *LNCS*, pp. 33–49. Springer, (2014).

[45] Rémi Tournaire, Jean-Marc Petit, Marie-Christine Rousset, and Alexandre Termier, 'Discovery of probabilistic mappings between taxonomies: principles and experiments', *Journal on Data Semantics*, **15**, 66–101, (2011).

[46] Jacopo Urbani, Frank van Harmelen, Stefan Schlobach, and Henri E. Bal, 'QueryPIE: backward reasoning for OWL horst over very large knowledge bases', in *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, volume 7031 of *LNCS*, pp. 730–745. Springer, (2011).

[47] Laurent Vieille, 'Recursive axioms in deductive databases: the query/subquery approach', in *Expert Database Conf.*, pp. 253–267, (1986).

[48] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov, 'Silk - A link discovery framework for the web of data', in *Proceedings of the WWW2009 Workshop on Linked Data on the Web, LDOW 2009, Madrid, Spain, April 20, 2009.*, volume 538 of *CEUR Workshop Proceedings*. CEUR-WS.org, (2009).