Extracting Bounded-level Modules from Deductive RDF Triplestores

Marie-Christine Rousset, Federico Ulliana

Extracting Bounded-level Modules from Deductive RDF Triplestores

Marie-Christine Rousset
Univ. Grenoble Alpes
CNRS, LIG, 38000, Grenoble, France
IUF (Institut Universitaire de France)
marie-christine.rousset@imag.fr

Federico Ulliana
Université de Montpellier II
CNRS, LIRMM, 34000, Montpellier, France
GrahpIK Team, INRIA Sophia-Antipolis
ulliana@lirmm.fr

Abstract

We present a novel semantics for extracting *bounded-level* modules from RDF ontologies and databases augmented with safe inference rules, à la Datalog. Dealing with a recursive rule language poses challenging issues for defining the module semantics, and also makes module extraction algorithmically unsolvable in some cases. Our results include a set of module extraction algorithms compliant with the novel semantics. Experimental results show that the resulting framework is effective in extracting expressive modules from RDF datasets with formal guarantees, whilst controlling their succinctness.

Introduction

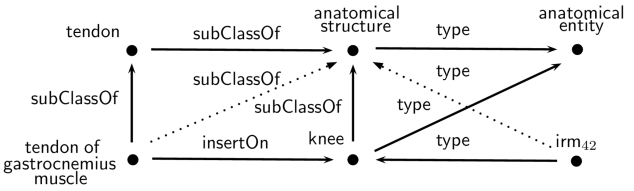
The Semantic Web consolidated a legacy of ontologies and databases today seen as *reference systems* for building new Semantic Web applications. To illustrate, consider a medical application for anatomy, whose goal is to showcase the structure of the human body, the most common pathologies and diseases, and the scientists that contributed to their study. A structural description of human anatomy can be drawn from FMA[1] or My Corporis Fabrica (MyCF).[2] A taxonomy of clinical terms about diseases can be extracted from SNOMED, [3] while biographical informations about scientists implied in studies can be taken from DBPedia.[4] 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) (Grau et al. 2008; Konev et al. 2008), 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 (Suchanek, Kasneci, and Weikum 2007), 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 (De Giacomo, Lenzerini, and Rosati 2011). 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 (Grau et al. 2008), the risk in practice is to output large portions of the initial ontologies, thus jeopardizing the gains of modularization.

Contributions. As a **first** contribution, we propose a unifying framework for RDF ontologies and databases we call *deductive RDF triplestores*, which is based on RDF triplestores equipped with safe 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 **second** and main contribution of the paper is a parametric 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.

[1] fma.biostr.washington.edu
[2] www.mycorporisfabrica.org
[3] www.ihtsdo.org/snomed-ct
[4] www.dbpedia.org



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

Figure 1: Triplestore D_1

The modules in our framework are constituted of both *data* and *rules* entailed by a reference system. Dealing with recursive Datalog rules makes properly defining the parametric modules challenging, and module extraction algorithmically unsolvable in some cases. Therefore, we focus on a class of rules meeting a mild condition on indirect recursion. This is still expressive enough to state for instance transitivity rules. As a **third** contribution, we then provide sound and complete module extraction algorithms, and outline their complexity. Our approach has been implemented on top of an RDF engine and experimentally tested. Proofs and experiment details are reported in (Rousset and Ulliana 2014).

Deductive RDF Triplestores

Data Along the lines of (Calì, Gottlob, and Pieris 2011) and (Libkin, Reutter, and Vrgoc 2013), we assume an infinite universe of constants CONST, which forms the triplestore domain, and an infinite set of variables VARS used in rules. We range over constants by $\mathsf{a}, \mathsf{b}, \mathsf{p}, \mathsf{q}$. We denote by x, y, z and $\bar{x}, \bar{y}, \bar{z}$, variables and sequences of variables belonging to VARS, respectively. A *term*, denoted by u, v is either a constant or a variable. We define RDF triplestores as relational databases over a schema restricted to a single ternary relation $t(\cdot,\cdot,\cdot)$. An atomic formula t is of the form $\mathrm{t} = t(u_1, u_2, u_3)$, with u_i a term. A *triplestore* D is a finite set of atoms of the form $t(u_1, u_2, u_3)$ with $u_i \in$ CONST. Triplestores essentially represent RDF graph data over a relational vocabulary. A path $p_{(u_0,u_n)} = t(u_0, v_1, u_1), t(u_1, v_2, u_2), \ldots, t(u_{n-1}, v_n, u_n)$ 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 $|p_{(u_0,u_n)}| = n$.

Figure 1 presents an RDF triplestore, together with its graph version. The example is inspired by the MyCF ontology (Palombi et al. 2014), which classifies digital representation of human body parts, acquired by IRMs or tomographies, according to anatomical knowledge. For instance, the type edge connecting irm_{42} with knee, corresponds to the triplestore atom $t(\mathsf{irm}_{42}, \mathsf{type}, \mathsf{knee})$, which is the standard RDF syntax for class membership.

Rules Deductive triplestores are equipped with safe Datalog rules. These are first-order logic formulas of the form $\forall \bar{x}\bar{y}.\, \phi(\bar{x}, \bar{y}) \rightarrow t(\bar{x})$ where $\bar{x}$ and $\bar{y}$ are sequences of variables belonging to VARS, and $t(\bar{x})$ and $\phi(\bar{x}, \bar{y})$ are an atom and a conjunction of atoms over the ternary relation $t(\cdot,\cdot,\cdot)$, constituting the *head* and the *body* of a rule, respectively. We denote a rule by r and a set of rules by R. To illustrate, the rules for class subsumption
$r_1 : t(x, \mathsf{type}, y), t(y, \mathsf{subClassOf}, z) \rightarrow t(x, \mathsf{type}, z)$
$r_2 : t(x, \mathsf{subClassOf}, y),\ t(y, \mathsf{subClassOf}, z)$
$\rightarrow t(x, \mathsf{subClassOf}, z)$
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

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

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

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

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.

$r_{\mathsf{dom}} : t(x, \mathsf{domain}, z), t(y, x, y') \rightarrow t(y, \mathsf{type}, z)$
$r_{\mathsf{ran}} : t(x, \mathsf{range}, z'), t(y, x, y') \rightarrow t(y', \mathsf{type}, z')$

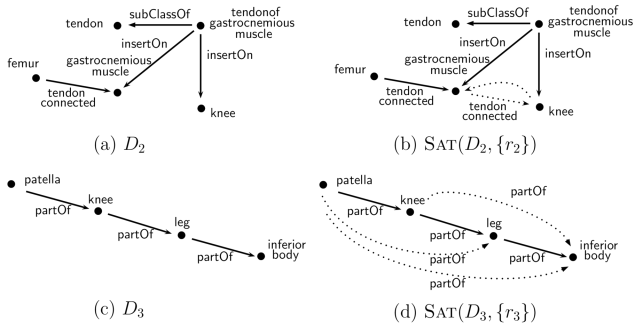
Datalog semantics is defined as the least fix-point of the immediate consequence operator (denoted by $\vdash_{\mathbf{1}}$).

Definition 1 (Datalog Inference) *The single-step Datalog inference, denoted by $D, R \vdash_{\mathbf{1}} D'$, holds when $D'{=}D \cup \{\mu(\mathit{head}(r)) \mid \mu(\mathit{body}(r)) \in D$ and $r \in R\}$, with μ an homomorphism from variables to constants. Then, we write $D, R \vdash D'$ when there exists a positive integer n such that $D_i, R \vdash_{\mathbf{1}} D_{i+1}$ for all $0{\leq}i{<}n$, with $D{=}D_0$ and $D'{\subseteq}D_n$. The triplestore D saturated with R, is defined as* $\textsc{Sat}(D, R) = \{\mathrm{t} \in D' \mid D, R \vdash D'\}$.

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 2 (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 $r \in R'$.*

A *deductive RDF triplestore* is a pair $\langle D, R\rangle$ where D is a triplestore and R is a finite set of (possibly recursive) rules. Triplestore entailment, denoted by $\langle D, R\rangle \vdash \langle D', R'\rangle$, holds when $D, R \vdash D'$ and $R \vdash R'$.



(a) D_2 (b) $\text{SAT}(D_2, \{r_2\})$ (c) D_3 (d) $\text{SAT}(D_3, \{r_3\})$

Figure 2: Triplestore examples

Bounded-level Modules

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 $t(v_1, u, v_2) : \Sigma$, if u is a property of Σ or $u \in \text{VARS}$. 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 2(c) it holds that $D_3 : (\,\mathsf{partOf}, \mathsf{subClassOf}\,)^2[\,\mathsf{knee}\,]$. However, it does not hold that $D_3 : (\,\mathsf{subClassOf}\,)^1[\,\mathsf{knee}\,]$.

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 $\text{SAT}(D, R)$. To see this, consider D_2 in Figure 2(a) and a rule inferring pairs of organs (y, z) physically connected by a tendon

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

A user interested in the organs directly and indirectly connected to the femur of this 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 $D_M = \{t(\mathsf{femur}, \mathsf{tendonConnected}, \mathsf{gastroc.Muscle})\}$. This dataset has however to be considered incomplete.

As shown in Figure 2(b), the rule r_2 entails on D_2 also the fact $t(\mathsf{gastroc.Muscle}, \mathsf{tendonConnected}, \mathsf{knee})$. This forms a path of length two together with the original triple $t(\mathsf{femur}, \mathsf{tendonConnected}, \mathsf{gastroc.Muscle})$, 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 $\text{SAT}(D, R)$ 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 2(c) and the recursive rule r_3 defining the transitivity of partOf

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

The saturated triplestore $\text{SAT}(D_3, r_3)$ is depicted in Figure 2(d). It contains $t(\mathsf{patella}, \mathsf{partOf}, \mathsf{knee})$ but also $t(\mathsf{patella}, \mathsf{partOf}, \mathsf{leg})$ and $t(\mathsf{patella}, \mathsf{partOf}, \mathsf{inferiorBody})$. More generally, it contains all triples of the form $\mathrm{t_b} = t(\mathsf{patella}, \mathsf{partOf}, \mathsf{b})$ 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 $\mathrm{t_b}$ 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* to the module the recursive rules. This leads to the following novel definition of module.

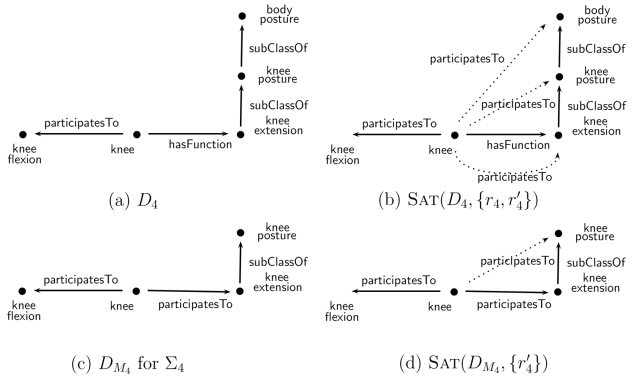
Definition 3 (Module) *Let $\langle D, R\rangle$ be a deductive triplestore and $\Sigma = (\,\mathsf{p}_1, \ldots, \mathsf{p}_n\,)^k[\,\mathsf{a}\,]$ a signature. Then, $M{=}\langle D_M, R_M\rangle$ is a module for Σ on $\langle D, R\rangle$ if*

1. $\langle D_M, R_M\rangle : \Sigma$
2. $\langle D, R\rangle \vdash \langle D_M, R_M\rangle$
3. *if $p_{(\mathsf{a},\mathsf{b})} : \Sigma$ 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., (Grau et al. 2008)), 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.

To illustrate the definition, consider the triplestore D_4 of Figure 3(a) equipped with the rules below.

$r_4 : t(x, \mathsf{hasFunction}, y) \rightarrow t(x, \mathsf{participatesTo}, y)$
$r_4' : t(x, \mathsf{participatesTo}, y),\ t(y, \mathsf{subClassOf}, z) \rightarrow$
$t(x, \mathsf{participatesTo}, z)$



(a) D_4 (b) $\textsc{Sat}(D_4, \{r_4, r'_4\})$

(c) D_{M_4} for Σ_4 (d) $\textsc{Sat}(D_{M_4}, \{r'_4\})$

Figure 3: Triplestore and module examples

Figure 3(b) depicts $\textsc{Sat}(D_4, \{r_4, r'_4\})$. Consider now Σ_4=(participatesTo, subClassOf)^2[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 t_1=t(knee, participatesTo, bodyPosture) is included in the module dataset, which is not desirable. In contrast, $\mathrm{t}_2 = t$(knee, participatesTo, kneePosture) is expected to be in a module for the signature Σ_4. A structure satisfying Definition 3 is $M_4 = \langle D_{M_4}, R_{M_4} \rangle$ with D_{M_4} depicted in Figure 3(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 3(d).

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

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 set of Datalog rules 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.

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

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

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 3, 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.

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

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 $t(y, \mathsf{partOf}, x)$ in the body of r_1 unifies with the conclusion of r_2, it can be replaced by $t(y, \mathsf{p}, x)$, so as to get the rule $\bar{r} = t(x, \mathsf{q}, y), t(y, \mathsf{p}, x) \rightarrow t(x, \mathsf{q}, y)$. 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 5 *Consider $\Sigma = (\, \mathsf{p}, \mathsf{q} \,)^3[\, \mathsf{a}_1 \,]$ and R with*

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

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

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

However, there exist infinitely many unfoldings of rule r_2 with itself that yield expressions of the form $t(x_1, \mathsf{p}, x_2), t(x_2, \mathsf{p}, x_3), t(x_3, \mathsf{p}, x_4) \rightarrow t(x_1, \mathsf{q}, x_4)$ 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*.

To avoid ending up with infinite sets of module rules, we devised an unfolding algorithm based on a *breadth-first* strategy. Algorithm MRE 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 $\textsc{MRE}(N_{ToUnfold}, R, \Sigma)$. The set $N_{ToUnfold} \subseteq R$ is constituted of all rules $r \in R$ that conclude on a property of interest, that is $head(r) \,\colon\, \Sigma$. Any rule belonging to $N_{ToUnfold}$ (whose premises use properties that are

Algorithm 1: MRE($N_{ToUnfold}, R_{ToApply}, \Sigma$)

```
1  forall the r1 ∈ N_ToUnfold do
2      if r1 ⫶ Σ then
3          R_M ← r1
4          remove r1 from R_ToApply
5      else
6          forall the r2 ∈ R_ToApply s.t. r1 ≠ r2 do
7              forall the r ∈ RuleUnfolding(r1, r2) do
8                  if r ⫶ Σ then
9                      R_M ← r
10                 R_M ← MRE({r}, R_ToApply \ {r, r2}, Σ)
11 return R_M
```

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 ($R_{ToApply} = R$). 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 $r_1 : t(x, \mathsf{p}, y), t(x, \mathsf{p}, z) \rightarrow t(x, \mathsf{p}, y)$ with $r_2 : t(x, \mathsf{partOf}, y) \rightarrow t(x, \mathsf{p}, y)$ are

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

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 6 (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 $r \colon \Sigma$ it holds*

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

Furthermore, when $R^{\mathsf{Rec}} \colon \Sigma$ 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 3, 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 5 with $D = \{t(\mathsf{a}_1, \mathsf{p}, \mathsf{a}_2), t(\mathsf{a}_2, \mathsf{p}, \mathsf{a}_3), t(\mathsf{a}_3, \mathsf{p}, \mathsf{a}_4)\}$. Recall that $\Sigma = (\,\mathsf{p}, \mathsf{q}\,)^3[\,\mathsf{a}_1\,]$, and then notice that the recursive rule $r_2 \not\colon \Sigma$. Here, module data extraction yields $D_M = D$. Observe now that the atom $t(\mathsf{a}_1, \mathsf{q}, \mathsf{a}_4)$ belongs to $\textsc{Sat}(D_M, R)$. As MRE outputs the set $R_M = \{t(x, \mathsf{p}, y), t(y, \mathsf{p}, z) \rightarrow t(x, \mathsf{q}, z)\}$, the triple $t(\mathsf{a}_1, \mathsf{q}, \mathsf{a}_4)$ does not belong to $\textsc{Sat}(D_M, R_M)$, while it should. Hence, $\langle D_M, R_M\rangle$ does not satisfy Definition 3.

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 $r \colon \Sigma$. As Example 5 illustrates, the extracted R_M must mimic an *infinite* set of rules of the form $t(x_1, \mathsf{p}, x_2), t(x_2, \mathsf{p}, x_3) \ldots t(x_{n-1}, \mathsf{p}, x_n) \rightarrow t(x_1, \mathsf{q}, x_n)$. One may think of capturing this infinite set by adding a recursive rule $r_{\mathsf{p}} : t(x, \mathsf{p}, y), t(y, \mathsf{p}, z) \rightarrow t(x, \mathsf{p}, z)$ together with $\bar{r} : t(x_1, \mathsf{p}, x_2), t(x_2, \mathsf{p}, x_3) \rightarrow t(x_1, \mathsf{q}, x_3)$. However, adding this recursive rule makes infer triples using p that are not entailed by the reference system, thereby violating point 2 of Definition 3. 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 (Hillebrand et al. 1995).

Finally, note that Theorem 6 is actually stronger than what required by Definition 3, 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 4 and 6.

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 $\{1, 2, 3, 5, 10\}$.

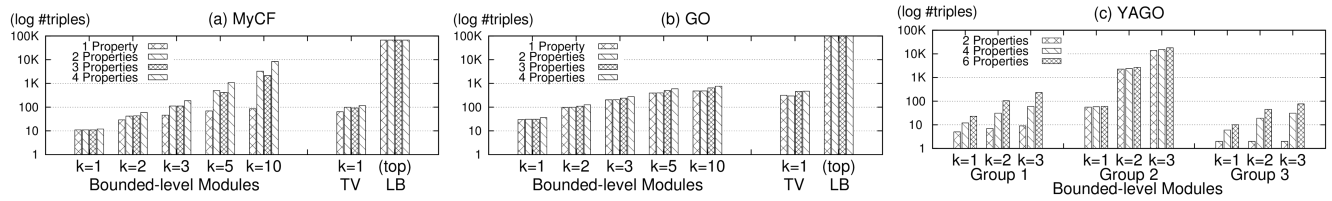


Figure 4: Size of Extracted Modules

Closest competitor approaches Relevant methods to our work are Traversal Views (Noy and Musen 2004) and Locality-based modules (Grau et al. 2008). 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 Figure 4(a-b) reports 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 4(c) we report the test on Yago2* with our approach, since LB does not support this RDF dataset.

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 4(c). 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 4(a) 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 4(b). Hence, by having modules made of *both* data and rules our approach allows to retain succinctness wrt TV.

LB extraction for *top-locality* modules has been tested thanks to the available prototype `www.cs.ox.ac.uk/isg/tools/ModuleExtractor/`. For MyCF and GO, it outputs almost the whole ontology (Figures 4(a-b)). This is due to ontology axioms that cannot be ignored for the logical completeness of the method.

To conclude, the experiments outline the advantages of bounded-level modules as i) flexibility, to accomodate diverse Semantic Web datasets, and ii) succinctness, when dealing with transitive properties and rules.

Related Work and Conclusion

Module extraction has been extensively studied for DL. Related works employ basically two approaches that consist in inferring a conservative extension of an ontology (Grau et al. 2008; Konev et al. 2008), and forgetting non-interesting relations of ontology schemas (Grau and Motik 2012). Approximations and heuristics have also been devised to mitigate the complexity costs (Nortje, Britz, and Meyer 2013), and have been validated by statistical analysis (Vescovo et al. 2013). Differently from our approach, all of these techniques focus exclusively on the ontology schema, and do not permit to modularize Semantic Web datasets starting also from ontology instances or RDF facts. The closest work to our, at least in spirit, is (Noy and Musen 2004). However, it does not consider inference, which makes module extraction challenging. Answering Datalog queries over RDF has been investigated in (Libkin, Reutter, and Vrgoc 2013) and (Bry et al. 2008).

We presented a novel approach for the extraction of bounded-level modules from deductive RDF triplestores, which provides new means to reuse of Linked-Data datasets, and favorizes the development of Semantic Web applications. The key contribution of the work is a novel module semantics allowing to bound their size, and compliant module extraction algorithms. As shown by our experiments, the resulting framework allows to efficiently extract expressive modules from Semantic Web ontologies and databases with formal guarantees, whilst effectively controlling their succinctness. Future works include the study of module robustness to updates (Goasdoue and Rousset 2013), the extension towards Datalog$\pm$ rules (Arenas, Gottlob, and Pieris 2014), and the connections between module extraction and recursive Datalog optimisation (Hillebrand et al. 1995).

Acknowledgements

This work has been partially supported by the project PAGODA (ANR-12-JS02-007-01) and by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01).

References

Arenas, M.; Gottlob, G.; and Pieris, A. 2014. Expressive languages for querying the semantic web. In *Proceedings of the 33st Symposium on Principle of Database Systems (PODS-14)*.

Bry, F.; Furche, T.; Ley, C.; Linse, B.; and Marnette, B. 2008. RDFLog: Its like Datalog for RDF. In *Proceedings of the Workshop on (Constraint) Logic Programming (WLP-08)*.

Calì, A.; Gottlob, G.; and Pieris, A. 2011. New expressive languages for ontological query answering. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI-11)*.

De Giacomo, G.; Lenzerini, M.; and Rosati, R. 2011. Higher-order description logics for domain metamodeling. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI-11)*.

Goasdoue, F., and Rousset, M.-C. 2013. Robust module-based data management. *IEEE Transactions on Knowledge and Data Engineering (TKDE-13)*.

Grau, B. C., and Motik, B. 2012. Reasoning over ontologies with hidden content: The import-by-query approach. *Journal of Artificial Intelligence Research (JAIR-12)*.

Grau, B. C.; Horrocks, I.; Kazakov, Y.; and Sattler, U. 2008. Modular reuse of ontologies: Theory and practice. *Journal of Artificial Intelligence Research (JAIR-08)*.

Hillebrand, G. G.; Kanellakis, P. C.; Mairson, H. G.; and Vardi, M. Y. 1995. Undecidable boundedness problems for datalog programs. *Journal of Logic Programming (JLP-95)*.

Konev, B.; Lutz, C.; Walther, D.; and Wolter, F. 2008. Semantic modularity and module extraction in description logics. In *Proceedings of the European Conference on Artificial Intelligence (ECAI-08)*.

Libkin, L.; Reutter, J. L.; and Vrgoc, D. 2013. Trial for RDF: adapting graph query languages for RDF data. In *Proceedings of the 32st Symposium on Principles of Database Systems (PODS-13)*.

Nortje, R.; Britz, K.; and Meyer, T. 2013. Reachability modules for the description logic $\mathcal{SRIQ}$. In *Logic for Programming, Artificial Intelligence, and Reasoning*, Lecture Notes in Computer Science.

Noy, N. F., and Musen, M. A. 2004. Specifying ontology views by traversal. In *Proceedings of the International Semantic Web Conference (ISWC-04)*.

Palombi, O.; Ulliana, F.; Favier, V.; Léon, J.-C.; and Rousset, M.-C. 2014. My Corporis Fabrica: an ontology-based tool for reasoning and querying on complex anatomical models. *Journal of Biomedical Semantics (JBS-14)*.

Rousset, M.-C., and Ulliana, F. 2014. Extracting bounded-level modules from deductive RDF triplestores. Technical report. http://www.lirmm.fr/~ulliana/papers/aaai15-full.pdf.

Suchanek, F. M.; Kasneci, G.; and Weikum, G. 2007. Yago: A core of semantic knowledge. In *Proceedings of the World Wide Web Conference (WWW-07)*.

Vescovo, C. D.; Klinov, P.; Parsia, B.; Sattler, U.; Schneider, T.; and Tsarkov, D. 2013. Empirical study of logic-based modules: Cheap is cheerful. In *International Workshop on Description Logics (DL-03)*.