# DaRLing: A Datalog rewriter for OWL 2 RL ontological reasoning under SPARQL queries∗

ALESSIO FIORENTINO, JESSICA ZANGARI and MARCO MANNA

*Department of Mathematics and Computer Science (DeMaCS), University of Calabria, Rende, Italy*
(*e-mail:* `lastname@mat.unical.it`) - *https://www.mat.unical.it*

## Abstract

The W3C Web Ontology Language (OWL) is a powerful knowledge representation formalism at the basis of many semantic-centric applications. Since its unrestricted usage makes reasoning undecidable already in case of very simple tasks, expressive yet decidable fragments have been identified. Among them, we focus on OWL 2 RL, which offers a rich variety of semantic constructors, apart from supporting all RDFS datatypes. Although popular Web resources - such as DBpedia - fall in OWL 2 RL, only a few systems have been designed and implemented for this fragment. None of them, however, fully satisfy all the following desiderata: (*i*) being freely available and regularly maintained; (*ii*) supporting query answering and SPARQL queries; (*iii*) properly applying the *sameAs* property without adopting the unique name assumption; (*iv*) dealing with concrete datatypes. To fill the gap, we present *DaRLing*, a freely available Datalog rewriter for OWL 2 RL ontological reasoning under SPARQL queries. In particular, we describe its architecture, the rewriting strategies it implements, and the result of an experimental evaluation that demonstrates its practical applicability.

*KEYWORDS*: Datalog, OWL 2 RL, SPARQL, Query Answering

## 1 Introduction

*Ontology-mediated query answering* is an emerging paradigm at the basis of many semantic-centric applications (Bienvenu 2016). In this setting, a classical data source is reinterpreted via an ontology, which provides a semantic conceptual view of the data. As a direct and positive effect, the knowledge provided by the ontology can be used to improve query answering. Among the formalisms that are capable to express such a conceptual layer, the *Web Ontology Language* (OWL) is certainly the most popular one (Smith et al. 2004; W3C OWL Working Group 2012). But an ontology-mediated query (OMQ) has also a second component other than the ontology: the actual query that specifies, in a semantic way and via the ontological vocabulary, which part of the data one is interested in. And the most suitable formalism used to specify a query that complements an OWL ontology is definitely the *SPARQL Protocol and RDF Query Language* (SPARQL), representing – as for OWL – a W3C standard (Harris and Seaborne 2018).

OWL is a very powerful formalism. But its unrestricted usage makes reasoning undecidable already in case of very simple tasks such as fact entailment. Hence, expressive yet decidable

Table 1: Main tools for ontology-mediated query answering over OWL 2 RL knowledge bases.

| Tool | License | Latest release | Query language | owl:sameAs | Datatypes |
|---|---|---|---|---|---|
| Clipper | Free | Dec 2015 | SPARQL-BGP | under UNA | No |
| *DaRLing* | Free | Jul 2020 | SPARQL-BGP | Yes | Yes |
| DReW | Free | Mar 2013 | SPARQL-BGP | No | No |
| Orel | Free | Feb 2010 | ground queries | No | No |
| OWL2DLV | Commercial | Jun 2019 | SPARQL-BGP | under UNA | Yes |
| OwlOntDB | - | - | SPARQL-DL$_E$ | under UNA | No |
| RDFox | Commercial | Jun 2020 | SPARQL 1.1 | Yes | Yes |

fragments have been identified. Among them, we focus in this paper on the one called OWL 2 RL (Motik et al. 2012). From the knowledge representation point of view, OWL 2 RL enables scalable reasoning without scarifying too much the expressiveness. Indeed, it supports all RDFS datatypes and provides a rich variety of semantic constructors, such as: *inverseOf*, *transitiveProperty*, *reflexiveProperty*, *equivalentClass*, *disjointWith*, *unionOf*, *minCardinality*, *allValuesFrom*, *someValuesFrom*, and *sameAs* – among others. But the simple fact of allowing *someValuesFrom* only in the left-hand-side of an axiom guarantees that conjunctive query answering can be performed in polynomial time in data complexity (when the OMQ is considered fixed) and in nondeterministic polynomial time in the general case (the latter being exactly the same computational complexity of evaluating a single conjunctive query over a relational database).

Although a number of important Web semantic resources – such as DBpedia[1] and FOAF[2] – trivially fall in OWL 2 RL, only a few systems have been designed and implemented in this setting. None of them, however, fully satisfy all the following desiderata: (*i*) being freely available and regularly maintained; (*ii*) supporting ontology-mediated query answering; (*iii*) properly applying the *sameAs* property without adopting the unique name assumption; (*iv*) dealing with concrete *datatypes*.

To fill this gap, we present *DaRLing*[3], a freely available Datalog rewriter for OWL 2 RL ontological reasoning under SPARQL queries. Table 1 reports the main tools supporting or implementing natively ontology-mediated query answering over knowledge bases that fall in the RL profile of OWL 2, or beyond. Concerning the query language, apart from Orel, all the tools support SPARQL patterns: SPARQL 1.1, SPARQL-BGP (Harris and Seaborne 2018), and SPARQL-DL$_E$ (Sirin and Parsia 2007). Finally, the row of OwlOntDB contains some missing value because the system is currently not available. Hence, none of the existing systems fully meet conditions (*i*)-(*iv*) above. (A deeper comparison and discussion is reported in Section 6.)

*DaRLing* can take in input an RDF dataset (ABox) $\mathcal{A}$, an OWL 2 RL ontology (TBox) $\mathcal{T}$ and a SPARQL query $q(\mathbf{x})$, and constructs an equivalent program $P$ with an output predicate *ans* of arity $|\mathbf{x}|$. Formally, for each $|\mathbf{x}|$-tuple of domain constants, $\mathcal{A} \cup \mathcal{T} \models q(\mathbf{t})$ if, and only if, the atom *ans*($\mathbf{t}$) can be derived via $P$, where $P$ is a Datalog program using inequality and stratified negation. The system builds on top of the OWL API. It supports different input formats and

---

[1] See https://wiki.dbpedia.org/
[2] See http://www.foaf-project.org/
[3] See https://demacs-unical.github.io/DaRLing/

knowledge bases organized in multiple files. Moreover, it can produce a suitable rewriting also if some inputs are missing. For example, in case the ABox is missing, then the generated program is simply equivalent to the pair TBox plus query.

In what follows, after some background material (Section 2), we describe *DaRLing*'s architecture (Section 3) and the rewriting strategies it implements (Sections 4 and 5). We discuss related work (Section 6) and eventually we report the result of an experimental evaluation (Section 7) before drawing some conclusions (Section 8).

## 2 Background

We assume that the reader is familiar with the basic notions about Datalog (Ceri et al. 1989) and Description Logics (Baader et al. 2008). In this section we provide the syntactic notation used in the remainder of the paper.

### 2.1 Datalog

A *term* is either a constant or a variable. If $t_1, \ldots, t_k$ are terms and $p$ is a *predicate symbol* of arity $k$, then $p(t_1, \ldots, t_k)$ is a *predicate atom*. A *built-in atom* has form $t \prec u$, where $t$ and $u$ are terms, and $\prec \in \{ ``<" ``\leq", ``>", ``\geq", ``=", ``\neq" \}$. A *classical literal l* is of the form *a* or *not a*, where *a* is a predicate atom; in the former case *l* is *positive*, otherwise *negative*. If *l* is a classical literal or a built-in atom, then *l* is called *naf-literal*. A *rule r* is of the form

$$h \leftarrow b_1 \wedge \ldots \wedge b_n.$$

where $n \geq 0$; *h* (the *head* of *r*) is a predicate atom, and $b_1, \ldots, b_n$ (the *body literals* of *r*) are naf-literals. A rule *r* is *safe* if each variable has an occurrence in at least a positive classical body literal of *r*. A *program* is a finite set *P* of safe rules stratified with respect to negation. A program, rule, or literal is *ground* if it contains no variables. A ground rule with an empty body is a *fact*.

### 2.2 The OWL 2 RL profile

Description logics (DLs) are a family of formal knowledge representation languages used to describe and reason about the "concepts" of an application domain. Among the most relevant applications of description logics there is the OWL Web Ontology Language[4], a knowledge representation language standardised by the World Wide Web Consortium (W3C) and designed to facilitate the development of Semantic Web applications. The current version of the OWL specification[5] is OWL 2 (Grau et al. 2008), developed by the W3C OWL Working Group. The syntactic elements of OWL 2 are almost analogous to those of a DL, with the main difference that *concepts* and *roles* are called *classes* and *properties* respectively. OWL 2 Profiles (i.e., OWL 2 EL, OWL 2 QL, and OWL 2 RL)[6] are syntactic restrictions of OWL 2 that offer significant benefits from a computational point of view at the expense of the expressive power. In the following we provide the notation related to the DL underlying the OWL 2 RL profile.

Let $N_C$ (*atomic concepts*), $N_R$ (*role names*), and $N_I$ (*individuals*) be mutually disjoint discrete

---

[4] See http://www.w3.org/TR/owl2-overview/
[5] See https://www.w3.org/TR/owl2-syntax/
[6] See http://www.w3.org/TR/owl2-profiles/

sets. A *role* is either $r \in N_R$ or an inverse role $r^-$ with $r \in N_R$. We denote by $R^-$ the *inverse* of a role $R$ defined by $R^- := r^-$ when $R = r$ and $R^- := r$ when $R = r^-$. The set of *concepts* is the smallest set such that: (*i*) $\top$, $\bot$, and every atomic concept $A \in N_C$ is a concept; (*ii*) if $C$ and $D$ are concepts and $R$ is a role, then $C \sqcap D$, $C \sqcup D$, $\neg C$, $\forall R.C$, $\exists R.C$, $\geq nR.C$ and $\leq nR.C$, for $n \geq 1$, are concepts.

*Subconcept* and *Superconcept Expressions* are recursively defined as follows: (*i*) $\top$ and every atomic concept $A \in N_C$ is both a Subconcept and a Superconcept Expression; (*ii*) if $C$ and $D$ are Subconcept Expressions and $R$ is a role, then $C \sqcap D$, $C \sqcup D$, $\exists R.C$ and $\geq 1R.C$ are Subconcept Expressions; (*iii*) if $C$ and $D$ are Superconcept Expressions and $R$ is a role, then $C \sqcap D$, $\neg C$, $\forall R.C$, and $\leq 1R.C$ are Superconcept Expressions.

A *knowledge base* (KB) is any pair $\mathcal{K} = (\mathcal{A}, \mathcal{T})$ where:

(*i*) $\mathcal{A}$, the ABox (*assertional box*), is a finite set of *assertions* of the form $A(a)$ or $R(a,b)$, with $a, b \in N_I$, $A \in N_C$, and $R \in N_R$;

(*ii*) $\mathcal{T}$, the TBox (*terminological box*), is a finite set of: (*a*) *concept inclusions* (CIs) of the form $C \sqsubseteq D$, where $C$ and $D$ are concepts; (*b*) *role inclusions* (RIs) of the form $R \sqsubseteq S$, where $R$ and $S$ are roles; and (*c*) *transitivity axioms* of the form trans($R$), where $R$ is a role.

We will generally refer to the elements of an ABox or a TBox by calling them *axioms*. We say that a TBox belongs to the OWL 2 RL profile if for each CI $C \sqsubseteq D$ it turns out that $C$ is a Subconcept Expression and $D$ is a Superconcept Expression; a KB belongs to the OWL 2 RL profile if its TBox belongs to it.

*Example 2.1*
The TBox

| | |
|---|---|
| `linkedViaTrain` $\sqsubseteq$ `linked` | `trans(linkedViaTrain)` |
| `CommutingArea` $\sqsubseteq$ $\exists$`linked.Capital` | $\exists$`linked.Capital` $\sqsubseteq$ `DesirableArea` |
| `Capital` $\sqsubseteq$ `DesirableArea` | |

does not fall in the OWL 2 RL profile. In fact, in `CommutingArea` $\sqsubseteq$ $\exists$`linked.Capital`, the concept $\exists$`linked.Capital` is not a Superconcept Expression. $\square$

*Positive* and *negative occurrences* of a concept $C$ in concepts are defined as follows:

- $C$ occurs positively in itself;
- $C$ occurs positively (resp., negatively) in $\neg C_-$ or $\leq nS.C_-$ if $C$ occurs negatively (resp., positively) in $C_-$; and
- $C$ occurs positively (resp., negatively) in $C_+ \sqcap D_+$, $C_+ \sqcup D_+$, $\exists R.C_+$, $\forall R.C_+$ or $\geq nS.C_+$ if $C$ occurs positively (resp., negatively) in $C_+$ or in $D_+$.

We say that a concept $C$ occurs positively (resp., negatively) in an axiom $C_- \sqsubseteq C_+$ if $C$ occurs positively (resp., negatively) in $C_+$, or negatively (resp., positively) in $C_-$; $C$ occurs positively (resp., negatively) in $\mathcal{T}$ if $C$ occurs positively (resp., negatively) in some axiom of $\mathcal{T}$.

## 3 The *DaRLing* rewriter: system overview

*DaRLing* is an open-source Datalog rewriter for OWL 2 RL ontologies available on the online webpage https://demacs-unical.github.io/DaRLing/. The rewriter implements the translation techniques described in Section 4 and supports the `owl:sameAs` management method described

in Section 5. In addition to the Datalog rewriting of OWL 2 RL ontologies, *DaRLing* also supports the Datalog translation of datasets in RDF/XML or Turtle format and SPARQL queries containing only basic graph patterns. The architecture of *DaRLing* is synthesized in Figure 1.
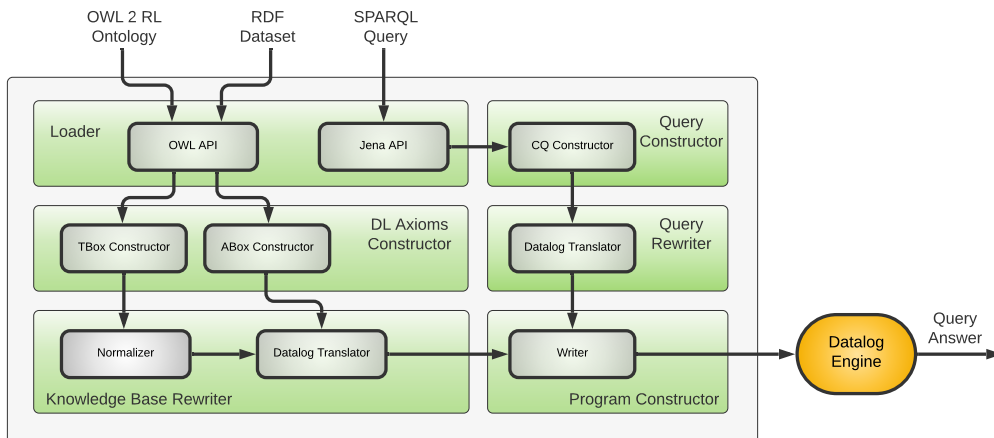


Fig. 1: *DaRLing* Architecture

*DaRLing* uses the OWL API (Horridge and Bechhofer 2009) to load the RL fragment of OWL 2 ontologies and datasets into internal data structures representing DL TBoxes and ABoxes, respectively. The system supports the loading of the datatypes *xsd:string* and *xsd:integer*, provided by the OWL 2 datatype map (a list of the datatypes that can be used in OWL 2 ontologies) for the representation of strings and integers. A rewrite module is implemented for the translation of a TBox and/or an ABox into a Datalog program. More in detail, a part of the axioms (those "directly rewritable") is passed to the *Datalog Translator*, whereas the remaining part is first subjected to the normalization procedure described in Section 4.

*DaRLing* also allows for Datalog translation of SPARQL queries containing *basic graph patterns* (BGPs) (i.e., sets of triple patterns forming a graph). SPARQL queries are parsed using Jena [7] – a Java API which can be used to create and manipulate RDF graphs – then they are translated into conjunctive Datalog queries.

The system supports different input formats and knowledge bases organized in multiple files. More in detail: (*i*) the ontology (resp., the dataset) can be contained in a single file or in a folder containing multiple files with one of the extensions *.owl* or *.rdfs* (resp., *.owl*, *.rdf* or *.ttl*); (*ii*) one or more queries have to be contained in a file with the *.SPARQL* extension. Moreover, *DaRLing* can produce a suitable rewriting also if some inputs are missing: for each file (or folder) received as input, a single *.asp* file containing the respective Datalog translation is returned as output. For example, in case the ABox is missing, then the generated program is simply equivalent to the pair TBox plus query.

By default, *DaRLing* rewrites under the *Unique Name Assumption* (UNA). However, it is possible to explicitly choose to enable rewriting with the `owl:sameAs` management mode described in Section 5 if at least one between ontology and query is given as input. The semantics of

---

[7] See https://jena.apache.org/

`owl:sameAs`, indeed, presuppose the enabling of matches between syntactically different but equivalent individuals. Therefore, its management is strictly linked to its propagation among the join variables of the Datalog rewriting of the ontology or the query (cfr. Section 5).

## 4 From OWL 2 RL to Datalog

In this section we describe how the TBox underlying an OWL 2 RL ontology is rewritten into Datalog. Since translating RIs and transitive axioms is almost trivial, we will focus only on the CIs. In particular, the rewriting process takes place as follows: ($i$) a class of axioms for which we provide a direct translation is identified; ($ii$) the remaining part of the axioms is subjected to a normalization procedure before being translated.

### 4.1 A direct translation for a class of CIs

For the first step we need to introduce the concepts of the $\mathcal{ELI}$ description logic. $\mathcal{ELI}$-*concepts* are inductively defined as follows: ($a$) $\top$ and each $A \in N_C$ is an $\mathcal{ELI}$-concept, and ($b$) if $C, D$ are $\mathcal{ELI}$-concepts and $R$ is a role, then $C \sqcap D$, $\exists R.C$ and $\geq 1R.C$ are $\mathcal{ELI}$-concepts. We show how to get the equivalent Datalog rule for an axiom of the form $\sqcup C_i \sqsubseteq \sqcap A_i$, where $\sqcup C_i$ is a disjunction of $\mathcal{ELI}$-concepts and $\sqcap A_i$ is a conjunction of atomic concepts. Since $\sqcup C_i \sqsubseteq \sqcap A_i$ is equivalent to the axioms $C_j \sqsubseteq A_k$ for each $C_j$ and $A_k$, it is sufficient to show how the translation works on CIs of the form $C \sqsubseteq A$, where $C$ is an $\mathcal{ELI}$-concept and $A$ is atomic. Algorithm 1 shows a recursive algorithm for generating literals starting from an $\mathcal{ELI}$-concept $C$. Intuitively, for each conjunction $\sqcap C_i$ (possibly consisting of a single clause), we have a variable common to each clause $C_i$. For every clause $C_i$ of the form $\exists R.D$ or $\geq 1R.D$ we introduce a fresh variable obtained by adding $i$ to the subscript of the variable shared by the conjunction to which $C_i$ belongs.

---

**Algorithm 1:** translate$\mathcal{ELI}$(C, Var, clause, bodyLiterals)

**Input:** $\mathcal{ELI}$-concept $C$, String Var, int clause, Set<Literal> bodyLiterals
**Result:** Addition of Datalog literals to bodyLiterals
**if** *C is atomic* **then**
  | bodyLiterals.add(C(Var));
**else if** *C has form* $\sqcap C_i$ **then**
  | **foreach** *i* **do**
  |   | translate$\mathcal{ELI}$($C_i$, Var, $i$, bodyLiterals);
**else if** *C has form* $\exists R.D$ $\mid$ $\geq 1R.D$ **then**
  | String newVar = Var + '_' + clause ;
  | **if** *R is an inverse role* **then**
  |   | bodyLiterals.add($R^-$(newVar,Var)) ;
  | **else**
  |   | bodyLiterals.add($R$(Var,newVar))
  | translate$\mathcal{ELI}$($D$, newVar, 1, bodyLiterals);

---

The translation of an axiom $C \sqsubseteq A$ (as above) is a rule whose head is the literal $A(X)$ and whose body literals are obtained by invoking Algorithm 1 on $C$ with Var $= X$ and clause $= 1$.

*Example 4.1*

The concept inclusion $\exists r.(\exists s.(C \sqcap D)) \sqcap\, \geq 1t.(E \sqcap \exists u^-.F) \sqsubseteq A$ translates directly to the following Datalog rule:

$$A(X) \leftarrow r(X,X_1) \wedge s(X_1,X_{1,1}) \wedge C(X_{1,1}) \wedge D(X_{1,1}) \wedge$$
$$t(X,X_2) \wedge E(X_2) \wedge u(X_{2,1},X_2) \wedge F(X_{2,1}).$$

In more detail, the variable shared by clauses $r.(\exists s.(C \sqcap D))$ and $\geq 1t.(E \sqcap \exists u^-.F)$ is $X$. The recursive call on these two clauses generates the body literals $\{r(X,X_1),\ s(X_1,X_{1,1}),\ C(X_{1,1}),\ D(X_{1,1})\}$ and $\{t(X,X_2),\ E(X_2),\ u(X_{2,1},X_2),\ F(X_{2,1})\}$ respectively.                     □

### 4.2  Enhancing the normalization procedure

For the second phase of the rewriting process, we transform the remaining axioms of the TBox (those not directly translatable) into a "normalized" form, from which the Datalog translation is immediate. We say that a TBox $\mathcal{T}$ is in *normalized form* if each concept inclusion axiom in $\mathcal{T}$ has form $\sqcap A_i \sqsubseteq C$, where $\sqcap A_i$ is a conjunction of atomic concepts and $C$ is a concept of the form $\perp$, $A$, $\forall R.A$ or $\leq 1R.A$, with $A$ atomic. Table 2 shows how the Datalog translation of a TBox in normalized form takes place; in particular, $A, A_1, \ldots, A_n$ are atomic concepts and $R(X,Y) = r(Y,X)$ if $R = r^-$. We bring our axioms to a normalized form readapting a normalization procedure described by Kazakov (2009), which is applicable to any Horn-$\mathcal{SHIQ}$ TBox and preserves the logical consequences of the ontological axioms. If on the one hand normalization allows us to easily translate a given ontology into Datalog, on the other, it could significantly increase the number of axioms. In what follows we describe, with the help of some examples, a version of the normalization procedure ad-hoc for OWL 2 RL. We also show how we enhance that procedure in order to avoid, where possible, an unnecessary growth of the number of axioms.

Normalization aims at reducing the complex structure of axioms by introducing fresh concept names for substructures and substituting them. Intuitively, the transformation works as follows: let $C$ be a complex concept containing $D$ as a sub-expression; then, a fresh concept name $A_D$ is introduced and constrained to extensionally coincide with $D$. This enables us to exchange all occurrences of $D$ in $C$ by $A_D$.

Formally, given a OWL 2 RL TBox $\mathcal{T}$, for every (sub-)concept $C$ in $\mathcal{T}$ we introduce a fresh

Table 2: Translation of Concept Inclusions in normalized form.

| Concept Inclusion (CI) | Equivalent Datalog Rule |
|---|---|
| $A_1 \sqcap \cdots \sqcap A_n \sqsubseteq \perp$ | $\perp \leftarrow A_1(X) \wedge \ldots \wedge A_n(X).$ |
| $A_1 \sqcap \cdots \sqcap A_n \sqsubseteq A$ | $A(X) \leftarrow A_1(X) \wedge \ldots \wedge A_n(X).$ |
| $A_1 \sqcap \cdots \sqcap A_n \sqsubseteq \forall R.A$ | $A(Y) \leftarrow R(X,Y) \wedge A_1(X) \wedge \ldots \wedge A_n(X).$ |
| $A_1 \sqcap \cdots \sqcap A_n \sqsubseteq\, \leq 1R.A$ | $sameAs(Y_1,Y_2) \leftarrow A_1(X) \wedge \ldots \wedge A_n(X) \wedge$ |
| | $R(X,Y_1) \wedge R(X,Y_2) \wedge A(Y_1) \wedge A(Y_2) \wedge Y_1 \neq Y_2.$ |

atomic concept $A_C$ and define a function $st(C)$ by:

$$st(A) = A \ (A \text{ atomic}); \qquad st(\bot) = \bot; \qquad st(\top) = \top;$$
$$st(\neg C) = \neg A_C; \qquad st(C \sqcap D) = A_C \sqcap A_D; \qquad st(C \sqcup D) = A_C \sqcup A_D;$$
$$st(\forall R.C) = \forall R.A_C; \qquad st(\exists R.C) = \exists R.A_C; \qquad st(\geq nR.C) = \geq nR.A_C;$$
$$st(\leq nR.C) = \leq nR.A_C \ .$$

The result of applying *structural transformation* to $\mathcal{T}$ is an ontology $\mathcal{T}'$ that contains all role inclusions and transitivity axioms in $\mathcal{T}$ in addition to the following axioms:

- $A_C \sqsubseteq st(C)$ for every $C$ occurring positively in $\mathcal{T}$
- $st(C) \sqsubseteq A_C$ for every $C$ occurring negatively in $\mathcal{T}$
- $A_C \sqsubseteq A_D$ for every concept inclusion $C \sqsubseteq D \in \mathcal{T}$

*Example 4.2*
The axiom $\exists r.(B \sqcap C) \sqsubseteq \forall s^-.D$ will be transformed into:

$$(R.1) \ A_{\forall s^-.D} \sqsubseteq \forall s^-.A_D \qquad (R.2) \ A_D \sqsubseteq D \qquad (R.3) \ \exists r.(A_{B \sqcap C}) \sqsubseteq A_{\exists r.(B \sqcap C)}$$
$$(R.4) \ A_B \sqcap A_C \sqsubseteq A_{B \sqcap C} \qquad (R.5) \ B \sqsubseteq A_B \qquad (R.6) \ C \sqsubseteq A_C$$
$$(R.7) \ A_{\exists r.(B \sqcap C)} \sqsubseteq A_{\forall s^-.D}$$

where $(R.1)$-$(R.2)$ derive from the positive occurrences of the concepts $\forall s^-.D$ and $D$, whereas $(R.3)$-$(R.6)$ derive from the negative occurrences of $\exists r.(B \sqcap C)$, $B \sqcap C$, $B$ and $C$, respectively. $\square$

By applying structural transformation to $\mathcal{T}$, we obtain a TBox $\mathcal{T}'$ containing only concept inclusions of the form $A_1 \sqsubseteq A_2$, $A \sqsubseteq st(C_+)$ and $st(C_-) \sqsubseteq A$, where $C_+$ occurs positively and $C_-$ occurs negatively in $\mathcal{T}$. Since $\mathcal{T}$ belongs to OWL 2 RL, $C_+$ can only be of the form $\top, A, \neg C$, $C \sqcap D, \forall R.C$ or $\leq 1R.C$, whereas $C_-$ can only be of the form $\top, A, C \sqcap D, C \sqcup D, \exists R.C$ or $\geq 1R.C$. Therefore, axioms in $\mathcal{T}'$ which do not appear in normalized form are transformed as follows:

$$A \sqsubseteq st(\neg C) = \neg A_C \qquad \Longrightarrow A \sqcap A_C \sqsubseteq \bot;$$
$$A \sqsubseteq st(C \sqcap D) = A_C \sqcap A_D \qquad \Longrightarrow A \sqsubseteq A_C, A \sqsubseteq A_D;$$
$$A_C \sqcup A_D = st(C \sqcup D) \sqsubseteq A \qquad \Longrightarrow A_C \sqsubseteq A, A_D \sqsubseteq A;$$
$$\exists R.A_C = st(\exists R.C) \sqsubseteq A \qquad \Longrightarrow A_C \sqsubseteq \forall R^-.A;$$
$$\geq 1R.A_C = st(\geq 1R.C) \sqsubseteq A \qquad \Longrightarrow A_C \sqsubseteq \forall R^-.A.$$

*Example 4.3*
The axiom $(R.3)$ of the Example 4.2 is not in normalized form and will be replaced with the axiom $A_{B \sqcap C} \sqsubseteq \forall r^-.A_{\exists r.(B \sqcap C)}$. $\square$

The rewriting process takes a huge advantage from the fact that many axioms – those described in Section 4.1 – are not subject to the normalization procedure. We further enhance that procedure by providing that: (*i*) no fresh concept is introduced for $\top, \bot$ and all the atomic concepts in the TBox; (*ii*) axioms already in normalized form are not subjected to the normalization process. The following example shows how this significantly reduces the number of rewritten axioms (and consequently the number of Datalog rules that derive from them).

*Example 4.4*

With the concept inclusion of Example 4.2, through the application of the structural transformation we produce in addition to $(R.3)$ and $(R.7)$ only the two axioms:

$$(R'.1)\ A_{\forall s^-.D} \sqsubseteq \forall s^-.D \qquad\qquad (R'.2)\ B \sqcap C \sqsubseteq A_{B \sqcap C}.$$

Then $(R.3)$ is replaced as in Example 4.3. Eventually, a CI already in normalized form like $A \sqcap B \sqsubseteq \forall r.C$, for which the standard normalization would generate 6 further axioms, is not subjected to the normalization procedure. □

## 5 Handling `owl:sameAs` via Datalog

The `sameAs` role derives from the `owl:sameAs` property which is used by many OWL 2 ontologies to declare equalities between resources. The assertion `sameAs`$(a,b)$ states that the individuals *a* and *b* are synonyms, i.e., *a* can be replaced with *b* without affecting the meaning of the ontology. As well as logic programming approaches, Datalog works under the *Unique Name Assumption* (UNA), i.e., presumes that different names represent different objects of the world. With the following example we highlight the need of handling the `owl:sameAs` in order to enable the match of equivalent constants for each join between variables in the body of a rule.

*Example 5.1*

Let us consider an ontology featuring the rule

$$DogOwner(X) \leftarrow hasPet(X,Y) \wedge Dog(Y).$$

together with the following set of facts:

$$hasPet(\text{“Peter”}, \text{“Brian”}). \quad Dog(\text{“BrianGriffin”}). \quad sameAs(\text{“Brian”}, \text{“BrianGriffin”}).$$

Note how, despite that *sameAs*("Brian","BrianGriffin") has the purpose of making the constants "Brian" and "BrianGriffin" interchangeable, the fact *DogOwner*("Peter") is not derived as it should. □

As mentioned in Section 3, in order to allow the rewriting with the non-UNA, at least one between ontology and query must be given as input to the system. Below we provide a way to handle reasoning on a Datalog program deriving from a Web ontology that contains the `owl:sameAs` property. The idea is to simulate the reflexivity, the symmetry and the transitivity of the `owl:sameAs` through a fresh binary predicate which connects all the elements of a `owl:sameAs`-clique to a representative (the lexicographic minimum) of that clique. To this end, given $N \geq 0$, we add the following block of rules to our Datalog program:

$$sameAs(X,Y) \leftarrow sameAs(Y,X). \tag{1}$$

$$noStart(X_1) \leftarrow sameAs(X_0,X_1) \wedge X_0 < X_1. \tag{2.1}$$

$$noStart(X_2) \leftarrow sameAs(X_0,X_1) \wedge sameAs(X_1,X_2) \wedge X_0 < X_2. \tag{2.2}$$

$$\vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots$$

$$noStart(X_N) \leftarrow sameAs(X_0,X_1) \wedge \ldots \wedge sameAs(X_{N-1},X_N) \wedge X_0 < X_N. \tag{2.N}$$

$$sameComp(X,Y) \leftarrow sameAs(X,Y) \wedge not\ noStart(X) \wedge X < Y. \tag{3}$$

$$sameComp(X,Z) \leftarrow sameComp(X,Y) \wedge sameAs(Y,Z) \wedge X < Y \wedge X < Z. \tag{4}$$

$$sameComp(X,X) \leftarrow sameComp(X,Y). \tag{5}$$

Here, rule (1) is the symmetric closure of the *sameAs* predicate, representing the pairs of equivalent individuals. Rules (2.1)-(2.N) map into the unary predicate *noStart* all individuals that are greater than another individual at a distance less than or equal to $N$ with respect to the *sameAs* predicate. Note that by $N = 0$ we mean that rules (2.1)-(2.N) are not considered and *noStart* is not populated. Given a constant $c$ that is not part of the extension of *noStart*, rules (3) and (4) compute the pairs $(c, d)$ where $c < d$ and $d$ can be reached from $c$ via the *sameAs* transitive closure. Since the minimum constant $c_{\min}$ of every owl:sameAs-clique $C$ can not populate the *noStart* predicate, it turns out that a fact $sameComp(c_{\min}, d)$ is generated for each constant of $C$. Eventually, rule (5) is the reflexive closure of *sameComp* with respect to its first argument.

Even though rules (1)-(5) ensure, for any $N \geq 0$, that each element of a owl:sameAs-clique is linked to the minimum of that clique by means of the *sameComp* predicate, many $sameComp(c, d)$ facts could arise, where $c$ is not the minimum of any owl:sameAs-clique. The purpose of rules (2.1)-(2.N) is precisely to avoid the generation of these extra facts as much as possible. As we will see later, the *sameComp* predicate is used whenever any join variable occurs in a rule or query in order to enable matches between individuals belonging to the same clique, therefore it is very important to keep its growth under control. It is easy to foresee that the larger $N$ is chosen the more the extension size of *sameComp* is reduced (due to the growing size of the extension of *noStart*), but at the expense of a possible greater consumption of time due to the computation of the paths in the generation of *noStart* instances. The choice of $N$ must therefore be weighted, according to the application domain at hand, in such a way that ideally, both the time and the extension size of *sameComp* are minimized. More detailed considerations are reported in Section 7 where we considered different values of $N$ for the real-world domain of DBpedia and experimentally identified $N = 2$ as the best compromise in the trade-off between time and space.

Further attempts have been made to manage owl:sameAs via Datalog (starting from the most naive one in which the reflexive, symmetrical and transitive closure is performed over the *sameAs* predicate directly, passing through more refined techniques in which the transitive closure relies on a fresh predicate name) but none of them was found to be applicable in practice. This reinforces our approach and highlights how important is handling owl:sameAs in an optimal way.

In the following we specify how we rewrite any rule in which appears at least a join variable. For each rule $r$, let $J_r$ (the join variables of $r$) be the set of the variables occurring more than once in the body of $r$, and $2^{J_r}$ be the power set of $J_r$. For $X \in J_r$ we denote with $\#_X^r$ the number of occurrences of the variable $X$ in the body of $r$. For each $V \in 2^{J_r}$ we produce a new rule $r_V$ obtained from $r$ as follows: for any $X \in V$ and $1 \leq i \leq \#_X^r$, we substitute the $i$-th occurrence of $X$ with a fresh variable $X_i$ and add a new body literal $sameComp(X, X_i)$. For instance, with the rule in Example 5.1 we have that $J_r = \{Y\}$ and we get the following additional rule in the ontology:

$$DogOwner(X) \leftarrow hasPet(X, Y_1) \wedge Dog(Y_2) \wedge sameComp(Y, Y_1) \wedge sameComp(Y, Y_2).$$

Intuitively, since in general the *sameAs* (and consequently *sameComp*) predicate does not involve all the constants of a program, any possible combination in which some join variables enables matches between constants syntactically different but linked by the owl:sameAs relationship has to be considered. We observe that, in most cases, only a few join variables occur in programs deriving from web ontologies. However, in order to prevent a potentially exponential growth of the program, we rewrite rules that have more than 3 join variables by projecting the *sameComp* predicate on each argument in which appears such a variable. By doing this we avoid all the possible combinations of the variables and we get a single new rule that binds *sameComp* to all the join variables simultaneously.

## 6 Related work

In this section we mention the tools in the literature supporting OWL 2 RL reasoning, whose approach is to express inference tasks for OWL 2 in terms of inference tasks for Datalog. Orel (Krötzsch et al. 2010) is a reasoning system which subsumes both the EL and the RL profile of the OWL 2 ontology language and its approach is based on a bottom-up materialisation of consequences in a database. In particular, ontological information are stored as facts, whereas logical ramifications are governed by "meta-rules" that resemble the rules of a deduction calculus. However, Orel supports neither SPARQL nor conjunctive queries. OwlOntDB (Faruqui and MacCaull 2012) works under the unique name assumption (UNA) to translate OWL 2 RL ontologies into Datalog programs, but it is no longer available. DReW (Xiao et al. 2012) is a query answering system which supports OWL 2 RL and OWL 2 EL (modulo datatypes). It uses DLV as underlying Datalog engine and has not been conceived to generate ontology rewritings. RDFox (Nenov et al. 2015) is a main-memory RDF store supporting Datalog reasoning with an efficient handling of `owl:sameAs` and SPARQL. After the initial development at University of Oxford, the system is now available commercially from Oxford Semantic Technologies, a spinout of the University backed by Samsung Ventures and Oxford Sciences Innovation. Among systems aforementioned, only RDFox handles `owl:sameAs`, whereas datatypes are supported by OwlOntDB and RDFox. Clipper (Eiter et al. 2012) is a reasoner for conjunctive query answering over Horn-$\mathcal{SHIQ}$ ontology. Being more oriented to DL languages rather than OWL ontologies, Clipper lacks the support of the *datatypes* constructs and manages the `owl:sameAs` property under the UNA. It can also be used to generate ontology rewritings only. OWL2DLV (Allocca et al. 2019) is a commercial system, builts on the ASP reasoner DLV2-SERVER (Leone et al. 2019), for evaluating SPARQL queries over very large OWL 2 knowledge bases whose associated DL falls within Horn-$\mathcal{SHIQ}$. As well as Clipper, OWL2DLV works under the UNA, but it cannot be used to generate ontology rewritings only. MASTRO (Calvanese et al. 2011) and Ontop (Calvanese et al. 2017) are open-source tools for Ontology-Based Data Access (OBDA) in which the ontology lies in the QL fragment of OWL 2. Ontop has its roots in MASTRO and is implemented through a query rewriting technique which avoids materializing triples. DLV$^{\exists}$ (Leone et al. 2019) is an effective system for fast query answering over `shy`, an easily recognizable class of strongly parsimonious programs that strictly generalizes Datalog while preserving its complexity even under conjunctive queries. Among the systems dedicated to ontological query answering in the context of existential rules there are also Graal (Baget et al. 2015) and VLog (Carral et al. 2019). These systems are not ad-hoc for OWL 2 RL and typically capture more expressive ontologies and then support features such as the *skolem* and the *restricted (standard) chase* for reasoning over existential rules.

## 7 Experimental evaluation

To demonstrate the practical applicability of *DaRLing*, we designed and conducted an experimental evaluation based on the following working hypotheses: (*i*) over synthetic OWL 2 RL benchmarks, *DaRLing*'s output is comparable with the one produced by existing tools in terms of both number of produced rules and quality of the rewriting (the latter measured via execution time fixed the Datalog engine); (*ii*) over real-world OWL 2 RL knowledge bases, *DaRLing*'s rewriting strategy enables scalable query answering even in case the UNA is not a viable option.

### 7.1 Set-up

Since *DaRLing* is a rewriter which does not rely on any Datalog engine, according to Table 1 and Section 6, the only tool that can be fairly tested against *DaRLing* is the part of Clipper providing the Datalog rewriting of an OMQ, hereinafter called *Clipper-Rew*. Moreover, since these rewriters are independent from the evaluation phase, for the purpose of our testing, the choice of the Datalog engine is immaterial. In our case, we simply opted for I-DLV[8] (Calimeri et al. 2016), an Answer Set Programming instantiator and full-fledged state-of-the-art Datalog reasoner. Indeed, the system when fed with a disjunction-free and stratified under negation program is able to fully evaluate it and compute its perfect model. As benchmarks, we relied on LUBM (Guo et al. 2005), Adolena, Stock Exchange, Vicodì[9] and DBpedia (Auer et al. 2007).

LUBM is an Horn-$\mathcal{SHIQ}$ ontology over a university domain with synthetic data and 14 queries. We restricted to the fragment falling in RL and considered its 14 canonical queries. Adolena, Stock Exchange and Vicodì have been derived from a well-established benchmark (Leone et al. 2019). They are expressed in DL-Lite$_R$ and provided with 5 queries each. Adolena (Abilities and Disabilities OntoLogy for ENhancing Accessibility) has been developed for the South African National Accessibility Portal. Stock Exchange is an ontology of the domain of financial institution within the EU. Vicodì is an ontology of European history. For LUBM we adopted 6 datasets, wherease for each of Adolena, Stock Exchange and Vicodì, we used 5 datasets, downloaded from https://www.mat.unical.it/dlve.

Differently from the other aforementioned benchmarks, that are synthetic and assume the UNA, DBpedia is a real-world knowledge base requiring a proper handling of the owl:sameAs property. More in detail, DBpedia is a well-known KB falling in OWL 2 RL and created with the aim of sharing on the Web the multilingual knowledge collected by Wikimedia projects in a machine-readable format. The dataset has been extracted from the latest stable release of the whole DBpedia dataset. The considered part consists of the English edition of Wikipedia and is composed by about half a billion RDF triples. We inherited a set of 10 queries from an application conceived to query DBpedia in natural language (Leone et al. 2019). All tests were performed on a machine having two 2.8GHz AMD Opteron 6320 processors and 128 GB of RAM. All rewritings for each benchmark and executables are available at https://demacs-unical.github.io/DaRLing.

### 7.2 Quality

In this former set of experiments, we first generated *Clipper-Rew* and *DaRLing* rewritings for all queries of LUBM, Adolena, Stock Exchange and Vicodì; then, over these rewritings and for all considered datasets, we executed I-DLV under two different scenarios: in the scenario *materialize* the system is forced to materialize the whole ontology and then prompted to answer to each query individually; in the scenario *query-driven* the system still runs each query one by one, but performs a more efficient evaluation tailored on the query at hand by enabling the magic sets technique (Alviano et al. 2019).

Figure 2 reports average running times in seconds of I-DLV executions over all datasets on LUBM, Adolena, Stock Exchange and Vicodì, respectively. In particular, for better highlighting

---

[8] See https://github.com/DeMaCS-UNICAL/I-DLV
[9] See http://www.vicodi.org

| | Materialize | | Query-Driven | |
|---|---|---|---|---|
| | C-Rew | DaRL | C-Rew | DaRL |
| **1. LUBM** | | | | |
| ABox | 28.33 | | | |
| TBox | 5.30 | 4.75 | - | - |
| q1 | 13.85 | 15.43 | 0.80 | 1.10 |
| q2 | 14.04 | 14.36 | 4.37 | 3.28 |
| q3 | 13.67 | 13.69 | 0.79 | 1.10 |
| q4 | 16.32 | 16.87 | 3.51 | 3.41 |
| q5 | 13.21 | 14.55 | 2.19 | 4.17 |
| q6 | 15.33 | 14.71 | 1.72 | 2.13 |
| q7 | 14.38 | 14.38 | 1.13 | 1.19 |
| q8 | 15.15 | 14.99 | 2.26 | 2.38 |
| q9 | 15.02 | 15.53 | 2.56 | 2.71 |
| q10 | 14.17 | 13.84 | 1.09 | 0.93 |
| q11 | 12.94 | 13.43 | 0.07 | 0.13 |
| q12 | 13.10 | 12.87 | 0.05 | 0.05 |
| q13 | 13.32 | 13.92 | 2.86 | 4.35 |
| q14 | 14.79 | 15.45 | 1.54 | 1.54 |

| **2. Adolena** | | | |
|---|---|---|---|
| ABox | 4.78 | | |
| TBox 11.13 | 11.28 | - | - |
| q1 | 0.36 0.47 | 6.14 | 6.1 |
| q2 | 1.03 0.58 | 6.72 | 6.63 |
| q3 | 47.75 46.2 | 57.08 | 56.58 |
| q4 | 1.13 0.89 | 6.82 | 6.83 |
| q5 | 51.94 49.68 | 61.08 | 58.95 |
| **3. Stock Exchange** | | | |
| ABox | 1.13 | | |
| TBox 3.80 | 3.30 | - | - |
| q1 | 0.05 0.04 | 0.16 | 0.16 |
| q2 | 0.81 0.85 | 3.47 | 2.93 |
| q3 | 28.51 29.37 | 31.14 | 30.01 |
| q4 | 7.68 7.77 | 10.62 | 9.7 |
| q5 | 265.72 258.36 | 270.85 | 257.1 |
| **4. Vicodì** | | | |
| ABox | 7.75 | | |
| TBox 9.65 | 9.89 | - | - |
| q1 | 7.63 7.36 | 0.13 | 0.05 |
| q2 | 12.55 12.69 | 11.37 | 11.31 |
| q3 | 9.50 9.33 | 3.19 | 3.37 |
| q4 | 11.18 11.33 | 3.69 | 3.47 |
| q5 | 11.13 11.44 | 3.69 | 3.51 |

Fig. 2: Experiments on LUBM, Adolena, Stock Exchange and Vicodì. C-Rew stands for Clipper-Rew, DaRL for DaRLing. Times are in seconds.

differences in performance, for each execution we extracted from total time the static and fixed times spent on ABox loading and on TBox materialization over the ABox. These fixed times are reported in the table as *ABox* and *TBox*, respectively. Notice that TBox materializing times are not reported in case of scenario *query-driven* as there no materialization is done. Results show that in most cases performance achieved by I-DLV when using *DaRLing* outputs is comparable w.r.t. *Clipper-Rew*. Some worsening is observable especially on LUBM queries 5 and 13. This is reasonable since *Clipper-Rew* requires to take into account the query at hand for properly translating the query and the TBox, thus generating an output optimized on the basis of the query. *DaRLing* instead follows a different principle as it is designed to produce general and query-independent TBox rewritings without specific query-oriented enhancements. On Adolena and Stock Exchange, I-DLV times with *DaRLing* rewritings are generally, slightly better than with *Clipper-Rew*. Regarding Vicodì, I-DLV performance are practically the same since *Clipper-Rew* and *DaRLing* produced almost identical rewritings. This is mainly because Vicodì TBox, when rewritten by both *DaRLing* and *Clipper-Rew* into Datalog, consists of linear rules, i.e., rules having only an atom in body; thus, when the presence of joins is limited both approaches result almost equivalent.

### 7.3 Scalability

We also experimented on DBpedia with two types of rewritings of 10 DBpedia queries generated via *DaRLing*. In particular, we first measured the costs, in terms of both time and space, of

Table 3: Costs of the DBpedia *sameComp* materialization (time limit set to 10 hours).

| Parameter | N = 0 | N = 1 | N = 2 | N = 3 | N = 4 |
|---|---|---|---|---|---|
| *Time (seconds)* | 5,033 | 2,673 | 864 | 26,355 | timeout |
| *#sameComp* | 523M | 342M | 103M | 77M | - |
| *Memory (GB)* | 35 | 23 | 7 | 5 | - |

materializing the *sameComp* predicate by varying the value of *N* (see Section 5). To this end, we generated via I-DLV the materialization of the whole TBox under the UNA and filtered out the tuples of the *sameAs* predicate. The resulting dataset has been paired with rules (1)-(5) by considering different values of *N*. Table 3 reports the values of the following three parameters by varying *N*: the time (in seconds) needed for materializing the *sameComp* predicate, the extension size of the *sameComp* predicate and the memory consumption in GB.

Ideally, the optimal value of *N* should be the one minimizing both the time and the space (the memory consumption or, equivalently, the extension size of *sameComp*). As it is evident from Table 3, such a unique value does not exist. Indeed, the minimum value of time is reached for $N = 2$, whereas the minimum value of space is reached for $N = 3$. Between the two values, the best compromise seems to be offered by $N = 2$, since a small blow-up in terms of space is highly reward in terms of time saving.

We thus decided to focus on $N = 2$ and to evaluate, per each of the 10 queries, performance when I-DLV is provided with *DaRLing* rewritings generated for this value of *N*. Results are reported in Figure 3. We restricted experiments to *DaRLing* as we purposely want to investigate scalability of *DaRLing* under the non-UNA. In addition, comparisons with other tools (cfr. Table 1) would result unfair since to our knowledge, *DaRLing* is the only open-source project empowered with an ad-hoc handling of the `owl:sameAs` property. As in quality-measuring experiments, we considered both the *materialize* and *query-driven* scenarios. We reported execution times in seconds. *ABox* and *TBox* times, as in the above results, represent times spent on ABox loading and on TBox materialization over the ABox, respectively. Columns report for each query running times from which *ABox* and *TBox* times have been subtracted. In the *mate-*

| | *Materialize* | *Query-Driven* |
|---|---|---|
| *ABox* | | 3,209 |
| *TBox* | 4,048 | - |
| q1 | 27.86 | 1,574 |
| q2 | 44.14 | 1,576 |
| q3 | 46.17 | 3,453 |
| q4 | 46.58 | 6,192 |
| q5 | 77.29 | 1,636 |
| q6 | 43.94 | 1,544 |
| q7 | 1.05 | 1,416 |
| q8 | 19.50 | 1,419 |
| q9 | 0.01 | 1,496 |
| q10 | 0.11 | 1,377 |

Fig. 3: Experiments on DBpedia for $N = 2$. Times are in seconds.

*rialize* scenario, we observe that in general, once the TBox is materialized, the system is able to compute query answers in at most 1.3 minutes. A greater effort is paid if instead of materializing the TBox, the system is requested to perform a query-driven computation. Notably, the total time needed to answer the hardest query (namely q4 in the *query-driven* scenario) is 36% of the time spent for materializing just the *sameComp* predicate for $N = 3$ (see Table 3). This behaviour reinforces the choice of $N = 2$ as the best value for this domain. Concerning memory, in both scenarios and for each query, the memory peak is around 62 GB (including loading and materialization).

## 8 Conclusions

In this paper we have presented *DaRLing*, a Datalog-based rewriter for OWL 2 RL ontological reasoning under SPARQL queries. To demonstrate its practical applicability, we have designed and conducted an experimental evaluation based on two working hypotheses, which have been confirmed. The first release of *DaRLing* demonstrates to produce more general rewritings equivalent or sightly differing from the ones generated by Clipper, the closer open-source competitor of *DaRLing*. As additional feature, *DaRLing* can be used for transparently handling the `owl:sameAs` property independently from the Datalog reasoner at hand albeit requiring extra work due to the intrinsic need of computing the transitive closure. Such costs are strictly dependent from the ontology at hand; our experimentation in an unfriendly setting of a large ontology such as DBpedia proved a not taken for granted applicability of the approach.

Concerning our future plans, the ultimate objective is to develop a freely available customizable SPARQL endpoint for OWL 2 RL ontological reasoning complying with the W3C Recommendations. To this end, the next steps are: (*i*) extending the rewriting to enable the meta-reasoning, namely SPARQL queries where variables may range also over the given schema; (*ii*) handling `owl:sameAs` also over concepts and roles; (*iii*) enriching the set of supported datatypes; (*iv*) specializing the rewriting of the TBox by taking into account also the query; (*v*) specializing the rewriting of the query by taking into account also the TBox.

## References

ALLOCCA, C., CALIMERI, F., CIVILI, C., COSTABILE, R., CUTERI, B., FIORENTINO, A., FUSCÀ, D., GERMANO, S., LABOCCETTA, G., MANNA, M., PERRI, S., REALE, K., RICCA, F., VELTRI, P., AND ZANGARI, J. 2019. Large-scale reasoning on expressive horn ontologies. In *Proceedings of Datalog 2.0*. CEUR Workshop Proceedings, vol. 2368. CEUR-WS.org, 10–21.

ALVIANO, M., LEONE, N., VELTRI, P., AND ZANGARI, J. 2019. Enhancing magic sets with an application to ontological reasoning. *Theory Pract. Log. Program. 19*, 5-6, 654–670.

AUER, S., BIZER, C., KOBILAROV, G., LEHMANN, J., CYGANIAK, R., AND IVES, Z. G. 2007. Dbpedia: A nucleus for a web of open data. In *Proceedings of ISWC'07*. LNCS, vol. 4825. Springer, 722–735.

BAADER, F., HORROCKS, I., AND SATTLER, U. 2008. Description logics. In *Handbook of Knowledge Representation*. Foundations of Artificial Intelligence, vol. 3. Elsevier, 135–179.

BAGET, J., LECLÈRE, M., MUGNIER, M., ROCHER, S., AND SIPIETER, C. 2015. Graal: A toolkit for query answering with existential rules. In *Proceedings of RuleML'15*. LNCS, vol. 9202. Springer, 328–344.

BIENVENU, M. 2016. Ontology-mediated query answering: Harnessing knowledge to get more from data. In *Proceedings of IJCAI'16*. IJCAI/AAAI Press, 4058–4061.

CALIMERI, F., FUSCÀ, D., PERRI, S., AND ZANGARI, J. 2016. I-DLV: The new intelligent grounder of DLV. In *Proceedings of AI*IA'16*. LNCS, vol. 10037. Springer, 192–207.

CALVANESE, D., COGREL, B., KOMLA-EBRI, S., KONTCHAKOV, R., LANTI, D., REZK, M., RODRIGUEZ-MURO, M., AND XIAO, G. 2017. Ontop: Answering SPARQL queries over relational databases. *Semantic Web 8,* 3, 471–487.

CALVANESE, D., GIACOMO, G. D., LEMBO, D., LENZERINI, M., POGGI, A., RODRIGUEZ-MURO, M., ROSATI, R., RUZZI, M., AND SAVO, D. F. 2011. The MASTRO system for ontology-based data access. *Semantic Web 2,* 1, 43–53.

CARRAL, D., DRAGOSTE, I., GONZÁLEZ, L., JACOBS, C. J. H., KRÖTZSCH, M., AND URBANI, J. 2019. Vlog: A rule engine for knowledge graphs. In *Proceedings of ISWC'19*. LNCS, vol. 11779. Springer, 19–35.

CERI, S., GOTTLOB, G., AND TANCA, L. 1989. What you always wanted to know about datalog (and never dared to ask). *IEEE Trans. Knowl. Data Eng. 1,* 1, 146–166.

EITER, T., ORTIZ, M., SIMKUS, M., TRAN, T., AND XIAO, G. 2012. Query rewriting for horn-shiq plus rules. In *Proceedings of AAAI'12*. AAAI Press.

FARUQUI, R. U. AND MACCAULL, W. 2012. O wl O nt DB: A scalable reasoning system for OWL 2 RL ontologies with large aboxes. In *Proceedings of FHIES'12*. LNCS, vol. 7789. Springer, 105–123.

GRAU, B. C., HORROCKS, I., MOTIK, B., PARSIA, B., PATEL-SCHNEIDER, P. F., AND SATTLER, U. 2008. OWL 2: The next step for OWL. *J. Web Semant. 6,* 4, 309–322.

GUO, Y., PAN, Z., AND HEFLIN, J. 2005. LUBM: A benchmark for OWL knowledge base systems. *J. Web Sem. 3,* 2-3, 158–182.

HARRIS, S. AND SEABORNE, A. 2018. *SPARQL 1.1 Query Language*. W3C Recommendation. World Wide Web Consortium.

HORRIDGE, M. AND BECHHOFER, S. 2009. The OWL API: A java API for working with OWL 2 ontologies. In *Proceedings of OWLED'09*. CEUR Workshop Proceedings, vol. 529. CEUR-WS.org.

KAZAKOV, Y. 2009. Consequence-driven reasoning for horn SHIQ ontologies. In *Proceedings of IJCAI'09*. 2040–2045.

KRÖTZSCH, M., MEHDI, A., AND RUDOLPH, S. 2010. Orel: Database-driven reasoning for OWL 2 profiles. In *Proceedings of DL'10*. CEUR Workshop Proceedings, vol. 573. CEUR-WS.org.

LEONE, N., ALLOCCA, C., ALVIANO, M., CALIMERI, F., CIVILI, C., COSTABILE, R., FIORENTINO, A., FUSCÀ, D., GERMANO, S., LABOCCETTA, G., CUTERI, B., MANNA, M., PERRI, S., REALE, K., RICCA, F., VELTRI, P., AND ZANGARI, J. 2019. Enhancing DLV for large-scale reasoning. In *Proceedings of LPNMR'19*. LNCS, vol. 11481. Springer, 312–325.

LEONE, N., MANNA, M., TERRACINA, G., AND VELTRI, P. 2019. Fast query answering over existential rules. *ACM Trans. Comput. Log. 20,* 2, 12:1–12:48.

MOTIK, B., CUENCA GRAU, B., HORROCKS, I., WU, Z., FOKOUE, A., AND LUTZ, C. 2012. *OWL 2 Web Ontology Language Profiles (Second Edition)*. W3C Recommendation. World Wide Web Consortium.

NENOV, Y., PIRO, R., MOTIK, B., HORROCKS, I., WU, Z., AND BANERJEE, J. 2015. Rdfox: A highly-scalable RDF store. In *Proceedings of ISWC'15*. LNCS, vol. 9367. Springer, 3–20.

SIRIN, E. AND PARSIA, B. 2007. SPARQL-DL: SPARQL query for OWL-DL. In *Proceedings of OWLED'07*. CEUR Workshop Proceedings, vol. 258. CEUR-WS.org.

SMITH, M. K., WELTY, C., AND MCGUINNESS, D. L. 2004. *OWL Web Ontology Language Guide*. W3C Recommendation. World Wide Web Consortium.

W3C OWL WORKING GROUP. 2012. *OWL 2 Web Ontology Language Document Overview (Second Edition)*. W3C Recommendation. World Wide Web Consortium.

XIAO, G., EITER, T., AND HEYMANS, S. 2012. The drew system for nonmonotonic dl-programs. In *Proceedings of CSWS'12*. Springer, 383–390.