

Handling owl:sameAs via Rewriting

Boris Motik, Yavor Nenov, Robert Piro and Ian Horrocks

Department of Computer Science, Oxford University
Oxford, United Kingdom
forename.lastname@cs.ox.ac.uk

Abstract

Rewriting is widely used to optimise owl:sameAs reasoning in materialisation based OWL 2 RL systems. We investigate issues related to both the correctness and efficiency of rewriting, and present an algorithm that guarantees correctness, improves efficiency, and can be effectively parallelised. Our evaluation shows that our approach can reduce reasoning times on practical data sets by orders of magnitude.

1 Introduction

RDF (Manola and Miller 2004) and SPARQL (Harris and Seaborne 2013) are increasingly being used to store and access semistructured data. An OWL ontology (Motik, Patel-Schneider, and Parsia 2012) is often used to enhance query answers with tuples implied by the ontology and data, and the OWL 2 RL profile was specifically designed to allow for tractable rule-based query answering (Motik et al. 2012). In practice, the latter often involves using a forward chaining procedure in which the *materialisation* (i.e., all consequences) of the ontology and data is computed in a preprocessing step, allowing queries to be evaluated directly over the materialised triples. This technique is used by systems such as OwlGres (Stocker and Smith 2008), WebPIE (Urbani et al. 2012), Oracle’s RDF store (Wu et al. 2008), OWLIM SE (Bishop et al. 2011), and RDFox (Motik et al. 2014a).

One disadvantage of materialisation is that the preprocessing step can be costly w.r.t. both the computation and the storage of entailed triples. This problem is exacerbated when materialisation requires equality reasoning—that is, when the owl:sameAs property is used to state equalities between resources. OWL 2 RL/RDF (Motik et al. 2012, Section 4.3) axiomatises the semantics of owl:sameAs using rules such as $\langle s', p, o \rangle \leftarrow \langle s, p, o \rangle \wedge \langle s, \text{owl:sameAs}, s' \rangle$ that, for each pair of equal resources r and r' , ‘copy’ all triples between r and r' . It is well known that such ‘copying’ can severely impact both the materialisation size and time (Kolovski, Wu, and Eadon 2010); what is less obvious is that the increase in computation time due to duplicate derivations may be even more serious (see Section 3).

In order to address this problem, materialisation based systems often use some form of *rewriting*—a well-known

technique for theorem proving with equality (Baader and Nipkow 1998; Nieuwenhuis and Rubio 2001). In the OWL 2 RL setting, rewriting consists of choosing one representative from each set of equal resources, and replacing all remaining resources in the set with the representative. Variants of this idea have been implemented in many of the above mentioned systems, and they have been shown to be very effective on practical data sets (Kolovski, Wu, and Eadon 2010).

Although the idea of rewriting is well known, ensuring its correctness (i.e., ensuring that the answer to an arbitrary SPARQL query is the same with and without rewriting) is not straightforward. In this paper we identify two problems that, we believe, have been commonly overlooked in existing implementations. First, whenever a resource r is rewritten in the data, r must also be rewritten in the rules; hence, the rule set cannot be assumed to be fixed during the course of materialisation, which is particularly problematic if computation is parallelised. Second, it is a common assumption that SPARQL queries can be efficiently evaluated over the materialisation by rewriting them, evaluating them over the rewritten triples, and then ‘expanding’ the answer set (i.e., substituting all representative resources with equal ones in all possible ways). However, such an approach can be incorrect when SPARQL queries are evaluated under bag semantics, or when they contain builtin functions.

We address both issues in this paper and make the following contributions. In Section 3 we discuss the problems related to owl:sameAs in more detail and show how they can lead to both increased computation costs and incorrect query answers. In Section 4 we present an algorithm that generalises OWL 2 RL materialisation, can also handle SWRL rules (Horrocks et al. 2004), rewrites rules as well as data triples, and is *lock-free* (Herlihy and Shavit 2008). The latter means that at least one thread always makes progress, ensuring that the system is less susceptible to adverse thread scheduling decisions and thus scales better to many threads. In Section 5 we show how to modify SPARQL query processing so as to guarantee correctness. Finally, in Section 6 we present a preliminary evaluation of an implementation of our algorithms in the open-source RDFox system. We show that rewriting can reduce the number of materialised triples by a factor of up to 7.8, and can reduce materialisation time by a factor of up to 31.1 on a single thread, with the time saving being largely due to the elimination of dupli-

cate derivations. Our approach also parallelises computation very well in practice,¹ providing a speedup of up to 6.7 with eight physical cores, and up to 9.6 with 16 virtual cores.²

Due to space constraints, in this paper we have only been able to present a high level description of our algorithms, but detailed formalisations and correctness proofs are provided in a technical report (Motik et al. 2014b). The implemented system and all test data sets are available online.³

2 Preliminaries

A *term* is a *resource* (i.e., a constant) or a variable. Unless otherwise stated, s , p , o , and t are terms, and x , y , and z are variables. An *atom* is a triple of terms $\langle s, p, o \rangle$ called the *subject*, *predicate*, and *object*, respectively. A *fact* (or *triple*) is a variable-free atom. A *rule* r is an implication of the form (1), where $h(r) := \langle s, p, o \rangle$ is the *head*, $b(r) := \langle s_1, p_1, o_1 \rangle \wedge \dots \wedge \langle s_n, p_n, o_n \rangle$ is the *body*, and each variable in $h(r)$ also occurs in $b(r)$.

$$\langle s, p, o \rangle \leftarrow \langle s_1, p_1, o_1 \rangle \wedge \dots \wedge \langle s_n, p_n, o_n \rangle \quad (1)$$

A *program* P is a finite set of rules, and $P^\infty(E)$ is the *materialisation* of P on a finite set of *explicit* (i.e., extensional or EDB) facts E (Abiteboul, Hull, and Vianu 1995).

Two styles of OWL 2 RL reasoning are known, corresponding to the RDF- and DL-style semantics of OWL. In the RDF style, an ontology is represented using triples stored with the data in a single RDF graph, and a fixed (i.e., independent from the ontology) set of rules is used to axiomatise the RDF-style semantics (Motik et al. 2012, Section 4.3). While conceptually simple, this approach is inefficient because the fixed program contains complex joins. In the DL style, the rules are derived from and depend on the ontology (Grosz et al. 2003), but they are shorter and contain fewer joins. This approach is complete only if the ontology and the data satisfy conditions from Section 3 of (Motik, Patel-Schneider, and Parsia 2012)—an assumption commonly met in practice. Rewriting can be used with either style of reasoning, but we will use the DL style in our examples and evaluation because the rules are more readable and their evaluation tends to be more efficient.

3 Problems with owl:sameAs

In this section we discuss, by means of an example, the problems that the owl:sameAs property poses to materialisation-based reasoners. The semantics of owl:sameAs can be captured explicitly using program P_{\approx} , consisting of rules (\approx_1) – (\approx_5) , which axiomatises owl:sameAs as a congruence relation (i.e., an equivalence relation satisfying the replacement property). We call each set of resources all of which are equal to each other an *owl:sameAs-clique*.

$$\langle x_i, \text{owl:sameAs}, x_i \rangle \leftarrow \langle x_1, x_2, x_3 \rangle, \text{ for } 1 \leq i \leq 3 \quad (\approx_1)$$

¹Datalog reasoning is PTIME complete in the size of the data, and it is thus believed that sequential computation will be required in some cases.

²In *hyperthreading*, each virtual core has its own architectural state, but several virtual cores share the execution resources of one physical core.

³<https://krr-nas.cs.ox.ac.uk/2015/AAAI/RDFox/Equality/>

$$\begin{aligned} \langle x'_1, x_2, x_3 \rangle &\leftarrow \langle x_1, x_2, x_3 \rangle \wedge \langle x_1, \text{owl:sameAs}, x'_1 \rangle \quad (\approx_2) \\ \langle x_1, x'_2, x_3 \rangle &\leftarrow \langle x_1, x_2, x_3 \rangle \wedge \langle x_2, \text{owl:sameAs}, x'_2 \rangle \quad (\approx_3) \\ \langle x_1, x_2, x'_3 \rangle &\leftarrow \langle x_1, x_2, x_3 \rangle \wedge \langle x_3, \text{owl:sameAs}, x'_3 \rangle \quad (\approx_4) \\ \text{false} &\leftarrow \langle x, \text{owl:differentFrom}, x \rangle \quad (\approx_5) \end{aligned}$$

OWL 2 RL/RDF (Motik et al. 2012, Section 4.3) also makes owl:sameAs symmetric and transitive, but those rules are redundant as they are instances of (\approx_2) and (\approx_4) .

Rules (\approx_1) – (\approx_5) can lead to the derivation of many equivalent triples, as we demonstrate using an example program P_{ex} containing rules (R) – (F_3) ; these correspond directly to SWRL rules, but one could equally use slightly more complex rules obtained from OWL 2 RL axioms.

$$\begin{aligned} \langle x, \text{owl:sameAs}, \text{:USA} \rangle &\leftarrow \langle \text{:Obama}, \text{:presidentOf}, x \rangle \quad (R) \\ \langle x, \text{owl:sameAs}, \text{:Obama} \rangle &\leftarrow \langle x, \text{:presidentOf}, \text{:USA} \rangle \quad (S) \\ \langle \text{:USPresident}, \text{:presidentOf}, \text{:US} \rangle &\quad (F_1) \\ \langle \text{:Obama}, \text{:presidentOf}, \text{:America} \rangle &\quad (F_2) \\ \langle \text{:Obama}, \text{:presidentOf}, \text{:US} \rangle &\quad (F_3) \end{aligned}$$

On $P_{ex} \cup P_{\approx}$, rule (R) derives that :USA is equal to :US and :America, and rules (\approx_1) – (\approx_4) then derive an owl:sameAs triple for each of the nine pairs involving :USA, :America, and :US. The total number of derivations, however, is much higher: we derive each triple once from rule (\approx_1) , three times from rule (\approx_2) , once from rule (\approx_3) ,⁴ and three times from rule (\approx_4) ; thus, we get 66 derivations in total for the nine owl:sameAs triples. Analogously, rule (S) derives that :Obama and :USPresident are equal, and rules (\approx_1) – (\approx_4) derive the two owl:sameAs triples 22 times in total. These owl:sameAs triples lead to further inferences; for example, from (F_1) , rules (\approx_2) and (\approx_4) infer 2×3 triples with subject :Obama or :USPresident, and object :USA, :America, or :US. Each of these six triples is inferred three times from rule (\approx_2) , once from rule (\approx_3) , and three times from rule (\approx_4) , so we get 36 derivations in total.

Thus, for each owl:sameAs-clique of size n , rules (\approx_1) – (\approx_4) derive n^2 owl:sameAs triples via $2n^3 + n^2 + n$ derivations. Moreover, each triple $\langle s, p, o \rangle$ with terms in owl:sameAs-cliques of sizes n_s , n_p , and n_o , respectively, is ‘expanded’ to $n_s \times n_p \times n_o$ triples, each of which is derived $n_s + n_p + n_o$ times. This duplication of facts and derivations is a major source of inefficiency.

To reduce these numbers, we can choose a representative resource for each owl:sameAs-clique and then *rewrite* all triples—that is, replace all resources with their representatives (Stocker and Smith 2008; Urbani et al. 2012; Kolovski, Wu, and Eadon 2010; Bishop et al. 2011). For example, after applying rule (R) , we can choose :USA as the representative of :USA, :US and :America, and, after applying rule (S) , we can choose :Obama as the representative of :Obama and :USPresident. The materialisation of P_{ex} then contains only the triple $\langle \text{:Obama}, \text{:presidentOf}, \text{:US} \rangle$ and, as we show in Section 4, the number of derivations of owl:sameAs triples drops from over 60 to just 6.

⁴Rule (\approx_1) derives $\langle \text{owl:sameAs}, \text{owl:sameAs}, \text{owl:sameAs} \rangle$, so we can map variable x_2 to owl:sameAs in rule (\approx_3) .

Since owl:sameAs triples can be derived continuously during materialisation, rewriting cannot be applied as pre-processing; moreover, to ensure that rewriting does not affect query answers, the resulting materialisation must be equivalent, modulo rewriting, to $[P_{ex} \cup P_{\approx}]^{\infty}(E)$. Thus, we may need to continuously rewrite both triples and rules: rewriting only triples can be insufficient. For example, if we choose :US as the representative of :USA, :US and :America, then rule (S) will not be applicable, and we will fail to derive that :USPresident is equal to :Obama. To the best of our knowledge, no existing system implements rule rewriting; certainly OWLIM SE and Oracle’s RDF store do not,⁵ and so rewriting in these systems is *not* guaranteed to preserve query answers.

Note that the problem is less acute when using a fixed rule set operating on (the triple encoding of) the ontology and data, but it can still arise if owl:sameAs triples involve rdf: or owl: resources (with a fixed rule set, these are the only resources occurring in rule bodies).

4 Parallel Reasoning With Rewriting

The algorithm by Motik et al. (2014a) used in the RDFox system implements a fact-at-a-time version of the seminaïve algorithm (Abiteboul, Hull, and Vianu 1995): it initialises the set of facts T with the input data E , and then computes $P^{\infty}(E)$ by repeatedly applying rules from P to T using N threads until no new facts are derived. The objective of our approach is to adapt the RDFox algorithm to use rewriting and thus reduce both the size of T and the time required to compute it, while ensuring that an arbitrary SPARQL query can be answered over the resulting facts as if the query were evaluated directly over $[P \cup P_{\approx}]^{\infty}(E)$. To achieve this, we use a mapping ρ that maps resources to their representatives. For α a fact, a rule, or a set thereof, $\rho(\alpha)$ is obtained by replacing each resource r in α with $\rho(r)$; moreover, $T^{\rho} := \{\langle s, p, o \rangle \mid \langle \rho(s), \rho(p), \rho(o) \rangle \in T\}$ is the *expansion* of T with ρ . To promote concurrency, we update ρ in a lock-free way, using *compare-and-set* primitives to prevent thread interference. Moreover, we do not lock ρ when computing $\rho(\alpha)$; instead, we only require $\rho(\alpha)$ to be at least as current as α just before the computation. For example, if ρ is the identity as we start computing $\rho(\langle a, b, a \rangle)$, and another thread makes a' the representative of a , then $\langle a, b, a \rangle$, $\langle a', b, a \rangle$, $\langle a, b, a' \rangle$, and $\langle a', b, a' \rangle$ are all valid results.

We also maintain queues R and C of rewritten rules and resources, respectively, for which also use lock-free implementations as described by Herlihy and Shavit (2008).

To extend the original RDFox algorithm with rewriting, we allow each thread to perform three different actions. First, a thread can extract a rule r from the queue R of rewritten rules and apply r to the set of all facts T , thus ensuring that changes to resources in rules are taken into account.

Second, a thread can rewrite outdated facts—that is, facts containing a resource that is not a representative of itself. To avoid iteration over all facts in T , the thread extracts a resource c from the queue C of unprocessed outdated resources, and uses indexes by Motik et al. (2014a) to identify

each fact $F \in T$ containing c . The thread then removes each such F from T , and it adds $\rho(F)$ to T .

Third, a thread can extract and process an unprocessed fact F in T . The thread first checks whether F is outdated (i.e., whether $F \neq \rho(F)$); if so, the thread removes F from T and adds $\rho(F)$ to T . If F is not outdated but is of the form $\langle a, \text{owl:sameAs}, b \rangle$ with $a \neq b$, the thread chooses a representative of the two resources, updates ρ , and adds the other resource to queue C . The thread derives a contradiction if F is of the form $\langle a, \text{owl:differentFrom}, a \rangle$. Otherwise, the thread processes F by partially instantiating the rules in P containing a body atom that matches F , and applying such rules to T as described by Motik et al. (2014a).

Rewriting rules is nontrivial: RDFox uses an index to efficiently identify rules matching a fact, and the index may need updating when ρ changes. Updating the index in parallel would be very complex, so we perform this operation serially: when all threads are waiting (i.e., when all facts have been processed), a single thread updates P to $\rho(P)$, reindexes it, and inserts the updated rules (if any) into the queue R of rules for reevaluation. This is obviously a parallelisation bottleneck, but our experiments have shown that the time used for this process is not significant when programs are of moderate size.

Parallel modification of T can also be problematic, as the following example demonstrates: (1) thread A extracts a current fact F ; (2) thread B updates ρ and deletes an outdated fact F' ; and (3) thread A derives F' from F and writes F' into T , thus undoing the work of thread B. This could be solved via locking, but at the expense of parallelisation. Thus, instead of physically removing facts from T , we just mark them as outdated; then, when matching the body atoms of partially instantiated rules, we simply skip all marked facts. All this can be done lock-free, and we can remove all marked facts in a postprocessing step.

Our rewriting algorithm is presented in full in the accompanying technical report (Motik et al. 2014b). Theorem 1 states several important properties of our algorithm that, taken together, ensure the algorithm’s correctness; a detailed formalisation of the algorithm and a proof of the theorem are provided in the technical report.

Theorem 1. *The algorithm terminates for each finite set of facts E and program P . Let ρ be the final mapping and let T be the final set of unmarked facts.*

1. $\langle a, \text{owl:sameAs}, b \rangle \in T$ implies $a = b$ —that is, ρ captures all equalities.
2. $F \in T$ implies $\rho(F) = F$ —that is, T is minimal.
3. $T^{\rho} = [P \cup P_{\approx}]^{\infty}(E)$ —that is, T and ρ together represent $[P \cup P_{\approx}]^{\infty}(E)$.

Example

Table 1 shows six steps of an application of our algorithm to the example program P_{ex} from Section 3 on one thread. Some resource names have been abbreviated for convenience, and \approx abbreviates owl:sameAs. The \triangleright symbol identifies the last fact extracted from T . Facts are numbered for easier referencing, and their (re)derivation is indicated on the

⁵Personal communication.

right: $R(n)$ or $S(n)$ means that the fact was obtained from fact n and rule R or S ; moreover, we rewrite facts immediately after merging resources, so $W(n)$ identifies a rewritten version of fact n , and $M(n)$ means that a fact was marked outdated because fact n caused ρ to change.

We start by extracting facts from T and, in steps 1 and 2, we apply rule R to facts 2 and 3 to derive facts 4 and 5, respectively. In step 3, we extract fact 4, merge :America into :USA , mark facts 2 and 4 as outdated, and add their rewriting, facts 6 and 7, to T . In step 4 we merge :USA into :US , after which there are no further facts to process. Mapping ρ , however, has changed, so we update P to contain rules (R') and (S'), and add them to the queue R .

$\langle x, \text{owl:sameAs}, \text{:US} \rangle \leftarrow \langle \text{:Obama}, \text{:presidentOf}, x \rangle$ (R')
 $\langle x, \text{owl:sameAs}, \text{:Obama} \rangle \leftarrow \langle x, \text{:presidentOf}, \text{:US} \rangle$ (S')

In step 5 we evaluate the rules in queue R , which introduces facts 9 and 10. Finally, in step 6, we rewrite :USPresident into :Obama and mark facts 1 and 9 as outdated. At this point the algorithm terminates, making only six derivations in total, instead of more than 60 derivations when owl:sameAs is axiomatised explicitly (see Section 3).

5 Rewriting and SPARQL Query Answering

Given a set of unmarked facts T and mapping ρ , we can answer a SPARQL query Q correctly by evaluating Q in the expansion T^ρ , but that is inefficient because it ‘duplicates’ join evaluation for equal triples. An attempt at an improvement would be to evaluate $\rho(Q)$ in T and expand the result—that is, for each answer μ obtained by evaluating $\rho(Q)$ in T , output each answer ν such that $\rho(\nu(x)) = \mu(x)$ for each variable x in the domain of μ . However, using the example program P_{ex} from Section 3, we show that such an approach is *not* complete. Note that, after we finish the materialisation of P_{ex} , we have $\rho(x) = \text{:US}$ for each $x \in \{\text{:USA}, \text{:AM}, \text{:US}\}$ and $\rho(x) = \text{:Obama}$ for each $x \in \{\text{:USPresident}, \text{:Obama}\}$.

The first problem is due to the *bag* semantics of SPARQL, where repeated answers matter. Let Q_1 be as follows:

```
SELECT ?x WHERE { ?x :presidentOf ?y }
```

On T^ρ , query Q_1 produces answers $\mu_1 = \{?x \mapsto \text{:Obama}\}$ and $\mu_2 = \{?x \mapsto \text{:USPresident}\}$, each of which is repeated three times—once for each match of $?y$ to :USA , :US , or :America . In contrast, on T , query $\rho(Q_1)$ yields only one occurrence of μ_1 , and its expansion produces one occurrence of μ_2 ; we thus obtain all answers, but not with the correct cardinalities. This problem arises because variable $?y$ is projected out, so the final expansion step does not take into account the number of times each binding of $?y$ contributes to the result. To solve this problem, we must modify the projection operator to output each projected answer as many times as there are resources in the projected owl:sameAs -clique(s). Thus, we can answer Q_1 as follows: we match the triple pattern of $\rho(Q_1)$ to T as usual, obtaining one answer $\nu_1 = \{?x \mapsto \text{:Obama}, ?y \mapsto \text{:US}\}$; then, we project $?y$ from ν_1 to obtain three occurrences of μ_1 since the owl:sameAs -clique of :US is of size three; finally, we expand each occurrence of μ_1 to μ_2 to obtain all six results.

The second problem is due to SPARQL builtin functions. For example, let Q_2 be as follows:

```
SELECT ?y WHERE {
  ?x :presidentOf :US .
  BIND(STR(?x) AS ?y)
}
```

On T^ρ , query Q_2 produces answers $\tau_1 = \{?y \mapsto \text{“Obama”}\}$ and $\tau_2 = \{?y \mapsto \text{“USPresident”}\}$; in contrast, on T , query $\rho(Q_2)$ yields only τ_1 , which does not expand into τ_2 because strings “Obama” and “USPresident” are not equal. To solve this problem, we must expand answers *before* evaluating builtin functions. Thus, we can answer Q_2 as follows: we match the triple pattern of $\rho(Q_2)$ to T as usual, obtaining $\kappa_1 = \{?x \mapsto \text{:Obama}\}$; then, we expand κ_1 to $\kappa_2 = \{?x \mapsto \text{:USPresident}\}$; next, we evaluate the BIND expression and extend κ_1 and κ_2 with the respective values for $?y$; finally, we project $?x$ to obtain τ_1 and τ_2 . Since we have already expanded $?x$, we should not repeat the projected answers as many times as there are elements in the owl:sameAs -clique for $?x$; instead, we output each projected answer only once to obtain the correct answer cardinalities.

6 Evaluation

We have implemented our approach in RDFox, allowing the system to handle owl:sameAs via rewriting (REW) or the axiomatisation (AX) from Section 3. We then compared the performance of materialisation using these two approaches. In particular, we investigated the scalability of each approach with the number of threads, and we measured the effect that rewriting has on the number of derivations and materialised triples.

Test Data Sets. We used five test data sets, each consisting of an OWL 2 DL ontology and a set of facts. The data sets were chosen because they contain axioms with the owl:sameAs property leading to interesting inferences. Four data sets were derived from real-world applications.

- Claros has been developed in an international collaboration between IT experts and archaeology and classical art research institutions with the aim of integrating disparate cultural heritage databases.⁶
- DBpedia is a crowd-sourced community effort to extract structured information from Wikipedia and make this information available on the Web.⁷
- OpenCyc is an extensive ontology about general human knowledge. It contains hundreds of thousands of terms organised in a carefully designed ontology and can be used as the basis of a wide variety of intelligent applications.⁸
- UniProt is a subset of an extensive knowledge base about protein sequences and functional information.⁹

⁶<http://www.clarosnet.org/XDB/ASP/clarosHome/>

⁷<http://www.dbpedia.org/>

⁸<http://www.cyc.com/platform/opencyc/>

⁹<http://www.uniprot.org/>

Table 1: An Example Run of Materialisation on P_{ex} and One Thread

Step 1			Step 2			Step 3		
1	$\langle :USPres, :presOf, :US \rangle$		1	$\langle :USPres, :presOf, :US \rangle$		1	$\langle :USPres, :presOf, :US \rangle$	
$\triangleright 2$	$\langle :Obama, :presOf, :Am \rangle$		2	$\langle :Obama, :presOf, :Am \rangle$		2	$\langle :Obama, :presOf, :Am \rangle$	$M(4)$
3	$\langle :Obama, :presOf, :US \rangle$		$\triangleright 3$	$\langle :Obama, :presOf, :US \rangle$		3	$\langle :Obama, :presOf, :US \rangle$	
4	$\langle :Am, \approx, :USA \rangle$	$R(2)$	4	$\langle :Am, \approx, :USA \rangle$		$\triangleright 4$	$\langle :Am, \approx, :USA \rangle$	$M(4)$
			5	$\langle :US, \approx, :USA \rangle$	$R(3)$	5	$\langle :US, \approx, :USA \rangle$	
						6	$\langle :Obama, :presOf, :USA \rangle$	$W(2)$
						7	$\langle :USA, \approx, :USA \rangle$	$W(4)$
Step 4			Step 5			Step 6		
1	$\langle :USPres, :presOf, :US \rangle$		1	$\langle :USPres, :presOf, :US \rangle$		1	$\langle :USPres, :presOf, :US \rangle$	$M(9)$
3	$\langle :Obama, :presOf, :US \rangle$	$W(6)$	3	$\langle :Obama, :presOf, :US \rangle$		3	$\langle :Obama, :presOf, :US \rangle$	
$\triangleright 5$	$\langle :US, \approx, :USA \rangle$	$M(5)$	8	$\langle :US, \approx, :US \rangle$	$R'(3)$	8	$\langle :US, \approx, :US \rangle$	
6	$\langle :Obama, :presOf, :USA \rangle$	$M(5)$	9	$\langle :USPres, \approx, :Obama \rangle$	$S'(1)$	$\triangleright 9$	$\langle :USPres, \approx, :Obama \rangle$	$M(9)$
7	$\langle :USA, \approx, :USA \rangle$	$M(5)$	10	$\langle :Obama, \approx, :Obama \rangle$	$S'(3)$	10	$\langle :Obama, \approx, :Obama \rangle$	$W(9)$
8	$\langle :US, \approx, :US \rangle$	$W(5, 7)$						

The ontologies of all data sets other than DBpedia are not in the OWL 2 RL profile, so we first discarded all axioms outside OWL 2 RL, and then we translated the remaining axioms into rules as described in (Grosz et al. 2003).

Our fifth data set was UOBM (Ma et al. 2006)—a synthetic data set that extends the well-known LUBM (Guo, Pan, and Heflin 2005) benchmark. We did not use LUBM because neither its ontology nor its data uses the owl:sameAs property. The UOBM ontology is also outside OWL 2 RL; however, instead of using its OWL 2 RL subset, we used its *upper bound* (Zhou et al. 2013)—an unsound but complete OWL 2 RL approximation of the original ontology; thus, all answers that can be obtained from the original ontology can also be obtained from the upper bound, but not the other way around. Efficient materialisation of the upper bound was critical for the work by Zhou et al. (2013), and it has proved to be challenging due to equality reasoning.

The left-hand part of Table 2 summarises our test data sets: column ‘Rules’ shows the total number of rules, column ‘sA-rules’ shows the number of rules containing the owl:sameAs property in the head, and column ‘Triples before’ shows the number of triples before materialisation.

Test Setting. We conducted our tests on a Dell computer with 128 GB of RAM and two Xeon E5-2643 processors with a total of 8 physical and 16 virtual cores, running 64-bit Fedora release 20, kernel version 3.13.3-201. We have not conducted warm and cold start tests separately since, as a main-memory system, the performance of RDFox should not be affected by the state of the operating system’s buffers. For the AX tests, we extended the relevant program with the seven rules from Section 3. In all cases we verified that the expansion of the rewritten triples is identical to the triples derived using the axiomatisation.

Effect of Rewriting on Total Work. In order to see how rewriting affects the total amount of work, we materialised each test data set in both AX and REW modes while collecting statistics about the inference process; the results are shown in the right-hand part of Table 2. Column ‘Triples after’ shows the number of triples after materialisation; in the

case of REW tests, we additionally show the number of unmarked triples (i.e., of triples relevant to query answering). Column ‘Memory’ shows the total memory use as measured by RDFox’s internal counters. Column ‘Rule appl.’ shows the total number of times a rule has been applied to a triple, and column ‘Derivations’ shows the total number of derivations. Column ‘Merged resources’ shows the number of resources that were replaced with representatives in the course of materialisation. Finally, row ‘factor’ shows the ratio between the respective values in the AX and the REW tests.

As one can see, the reduction in the number of the derived triples is correlated with the number of rewritten constants: on UniProt there is no observable reduction since only five resources are merged; however, equalities proliferate on OpenCyc and so rewriting is particularly effective. In all cases the numbers of marked triples are negligible, suggesting that our decision to mark, rather than delete triples does not have unexpected drawbacks. In contrast, the reduction in the number of rule applications and, in particular, of derivations is much more pronounced than the reduction in the number of derived triples.

Effect of Rewriting on Materialisation Times. In order to see how rewriting affects materialisation times, we measured the wall-clock times needed to materialise our test data sets in AX and REW modes on 1, 2, 4, 8, 12, and 16 threads. For each test, we report average wall-clock time over three runs. Table 3 shows our test results; column ‘sec’ shows the materialisation time in seconds, column ‘spd’ shows the speedup over the single-threaded version, and column ‘ $\frac{AX}{REW}$ ’ shows the speedup of REW over AX.

As one can see from the table, RDFox parallelises computation exceptionally well in both AX and REW modes. When using the eight physical cores of our test server, the speedup is consistently between six and seven, which suggests that the lock-free algorithms and data structures of RDFox are very effective. We believe that the more-than-linear speedup on Claros is due to improved memory locality resulting in fewer CPU cache misses. The speedup continues to increase with hyperthreading, but is less pronounced:

Table 2: Test Data Sets Before and After Materialisation

	Rules	sA-rules	Triples before	Mode	Triples after		Memory (GB)	Rule appl.	Derivations	Merged resources
					unmarked	total				
Claros	1312	42	19M	AX	102M		4.5	867M	11,009M	
				REW	79.5M	79.7M	3.6	149M	128M	12,890
				factor		1.28x	1.28x	5.8x	85.5x	
DBpedia	3384	23	113M	AX	139M		6.9	934M	895M	
				REW	136M	136M	7.0	44.5M	37M	7,430
				factor		1.2x	0.99x	21.0x	24.4x	
OpenCyc	261,067	3,781	2.4M	AX	1,176M		35.9	7,832M	12,890M	
				REW	141M	142M	4.6	309M	281M	361,386
				factor		7.8x	7.8x	25.3x	45.9x	
UniProt	451	60	123M	AX	228M		15.1	1,801M	1,555M	
				REW	228M	228M	15.1	262M	183M	5
				factor		1.0x	1.0x	6.9x	8.5x	
UOBM	279	4	2.2M	AX	36M		1.2	332M	16,152M	
				REW	9.4M	9.7M	0.4	33.8M	4,256M	686
				factor		3.2x	3.2x	9.9x	3.8x	

Table 3: Materialisation Times with Axiomatisation and Rewriting

Test	Claros						DBpedia					OpenCyc				
	AX		REW		AX REW	AX		REW		AX REW	AX		REW		AX REW	
	sec	spd	sec	spd		sec	spd	sec	spd		sec	spd	sec	spd		
1	2042.9	1.0	65.8	1.0	31.1	219.8	1.0	31.7	1.0	6.9	2093.7	1.0	119.9	1.0	17.5	
2	969.7	2.1	35.2	1.9	27.6	114.6	1.9	17.6	1.8	6.5	1326.5	1.6	78.3	1.5	16.9	
4	462.0	4.4	18.1	3.6	25.5	66.3	3.3	10.7	3.0	6.2	692.6	3.0	40.5	3.0	17.1	
8	237.2	8.6	9.9	6.7	24.1	36.1	6.1	5.2	6.0	6.9	351.3	6.0	23.0	5.2	15.2	
12	184.9	11.1	7.9	8.3	23.3	31.9	6.9	4.1	7.7	7.7	291.8	7.2	56.2	2.1	5.5	
16	153.4	13.3	6.9	9.6	22.3	27.5	8.0	3.6	8.8	7.7	254.0	8.2	52.3	2.3	4.9	

Test	UniProt					UOBM				
	AX		REW		AX REW	AX		REW		AX REW
	sec	spd	sec	spd		sec	spd	sec	spd	
1	370.6	1.0	143.4	1.0	2.6	2696.7	1.0	1152.7	1.0	2.3
2	232.3	1.6	86.7	1.7	2.7	1524.6	1.8	599.6	1.9	2.5
4	129.2	2.9	46.5	3.1	2.8	813.3	3.3	318.3	3.6	2.6
8	74.7	5.0	25.1	5.7	3.0	439.9	6.1	177.7	6.5	2.5
12	61.0	6.1	19.9	7.2	3.1	348.9	7.7	152.7	7.6	2.3
16	61.9	6.0	17.1	8.4	3.6	314.4	8.6	137.9	8.4	2.3

virtual cores do not provide additional execution resources, and so they mainly compensate for CPU stalls due to cache misses. The AX mode seems to scale better with the number of threads than the REW mode, and we believe this to be due to contention between threads while accessing the map ρ . Only OpenCyc in REW mode did not scale particularly well: OpenCyc contains many rules, so sequentially updating P and the associated rule index when ρ changes becomes a significant parallelisation bottleneck. Finally, since the materialisation of Claros with more than eight threads in REW mode takes less than ten seconds, these results are difficult to measure and are susceptible to skew.

Our results confirm that rewriting can significantly reduce materialisation times. RDFox was consistently faster in the REW mode than in the AX mode even on UniProt, where the reduction in the number of triples is negligible. This is due to the reduction in the number of derivations, mainly involving rules (\approx_1)–(\approx_5), which is still significant on UniProt. In all cases, the speedup of rewriting is typically much larger than

the reduction in the number of derived triples (cf. Table 2), suggesting that the primary benefit of rewriting lies in less work needed to match the rules, rather than, as commonly thought thus far, in reducing the number of derived triples. This is consistent with the fact that the speedup of rewriting was not so pronounced on UniProt and UOBM, where the reduction in the number of derivations was less significant.

Our analysis of the derivations that RDFox makes on UOBM revealed that, due to the derived owl:sameAs triples, the materialisation contains large numbers of resources connected by the :hasSameHomeTownWith property. This property is also symmetric and transitive so, for each pair of connected resources, the number of times each triple is derived by the transitivity rule is quadratic in the number of connected resources. This leads to a large number of duplicate derivations that do *not* involve equality. Thus, although it is helpful, rewriting does not reduce the number of derivation in the same way as, for example, on Claros, which explains the relatively modest speedup of REW over AX.

7 Conclusion

In this paper we have investigated issues related to the use of rewriting in materialisation based OWL 2 RL systems, and have shown how these issues can lead to both increased computation costs and incorrect query answers. We have addressed these issues by presenting algorithms that guarantee the correctness of query answers for any input ontology and data set. Moreover, we have implemented our algorithms in the RDFox system, a preliminary evaluation of which has shown that our approach parallelises computation very well in practice and can reduce materialisation times on practical data sets by orders of magnitude.

Acknowledgements

This work was supported by the EPSRC projects MaSI³, Score! and DBOnto, and by the EU FP7 project Optique.

References

- Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases*. Addison Wesley.
- Baader, F., and Nipkow, T. 1998. *Term Rewriting and All That*. Cambridge University Press.
- Bishop, B.; Kiryakov, A.; Ognyanoff, D.; Peikov, I.; Tashev, Z.; and Velkov, R. 2011. Owlrim: A family of scalable semantic repositories. *Semantic Web* 2(1):33–42.
- Grosof, B. N.; Horrocks, I.; Volz, R.; and Decker, S. 2003. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proc. WWW*, 48–57.
- Guo, Y.; Pan, Z.; and Heflin, J. 2005. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics* 3(2–3):158–182.
- Harris, S., and Seaborne, A. 2013. SPARQL 1.1 Overview. W3C Recommendation. <http://www.w3.org/TR/sparql11-overview/>.
- Herlihy, M., and Shavit, N. 2008. *The Art of Multiprocessor Programming*. Morgan Kaufmann.
- Horrocks, I.; Patel-Schneider, P. F.; Boley, H.; Tabet, S.; Grosof, B.; and Dean, M. 2004. SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission. <http://www.w3.org/Submission/SWRL/>.
- Kolovski, V.; Wu, Z.; and Eadon, G. 2010. Optimizing Enterprise-Scale OWL 2 RL Reasoning in a Relational Database System. In *Proc. ISWC*, 436–452.
- Ma, L.; Yang, Y.; Qiu, Z.; Xie, G. T.; Pan, Y.; and Liu, S. 2006. Towards a Complete OWL Ontology Benchmark. In *Proc. ESWC*, 125–139.
- Manola, F., and Miller, E. 2004. RDF primer. W3C Recommendation. <http://www.w3.org/TR/rdf-primer/>.
- 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. <http://www.w3.org/TR/owl2-profiles/>.
- Motik, B.; Nenov, Y.; Piro, R.; Horrocks, I.; and Olteanu, D. 2014a. Parallel materialisation of Datalog programs in centralised, main-memory RDF systems. In *Proc. of the 28th Nat. Conf. on Artificial Intelligence (AAAI 14)*, 129–137. AAAI Press.
- Motik, B.; Nenov, Y.; Piro, R.; and Horrocks, I. 2014b. Handling owl:sameAs via Rewriting. arXiv:cs/1411.3622.
- Motik, B.; Patel-Schneider, P. F.; and Parsia, B. 2012. OWL 2 Web Ontology Language Structural Specification and Functional-style Syntax (Second Edition). W3C Recommendation. <http://www.w3.org/TR/owl2-syntax/>.
- Nieuwenhuis, R., and Rubio, A. 2001. Paramodulation-Based Theorem Proving. In Robinson, A., and Voronkov, A., eds., *Handbook of Automated Reasoning*, volume I. Elsevier Science. chapter 7, 371–443.
- Stocker, M., and Smith, M. 2008. Owlgres: A Scalable OWL Reasoner. In *Proc. OWLED: Experiences and Directions Workshop*, 26–27.
- Urbani, J.; Kotoulas, S.; Maassen, J.; van Harmelen, F.; and Bal, H. E. 2012. WebPIE: A Web-scale Parallel Inference Engine using MapReduce. *Journal of Web Semantics* 10:59–75.
- Wu, Z.; Eadon, G.; Das, S.; Chong, E. I.; Kolovski, V.; Annamalai, M.; and Srinivasan, J. 2008. Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules in Oracle. In *Proc. ICDE*, 1239–1248.
- Zhou, Y.; Cuenca Grau, B.; Horrocks, I.; Wu, Z.; and Banerjee, J. 2013. Making the most of your triple store: query answering in OWL 2 using an RL reasoner. In *Proc. WWW*, 1569–1580.