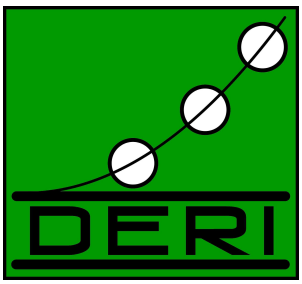

Scalable Authoritative OWL Reasoning for the Web

Aidan Hogan Andreas Harth
Axel Polleres

DERI Technical Report 2009-04-21
April 21, 2009

Please Cite As:
Aidan Hogan, Andreas Harth and Axel Polleres.
Scalable Authoritative OWL Reasoning for the Web.
International Journal on Semantic Web and Information Systems, 5(2), pages 49-90, April-June 2009.

DERI Galway
IDA Business Park
Galway, Ireland
www.deri.ie

Scalable Authoritative OWL Reasoning for the Web *

Aidan Hogan Andreas Harth Axel Polleres

Digital Enterprise Research Institute
National University of Ireland, Galway

April 21, 2009

Abstract

In this paper we discuss the challenges of performing reasoning on large scale RDF datasets from the Web. Using ter-Horst's pD* fragment of OWL as a base, we compose a rule-based framework for application to web data: we argue our decisions using observations of undesirable examples taken directly from the Web. We further temper our OWL fragment through consideration of "authoritative sources" which counter-acts an observed behaviour which we term "ontology hijacking": new ontologies published on the Web re-defining the semantics of existing entities resident in other ontologies. We then present our system for performing rule-based forward-chaining reasoning which we call SAOR: *Scalable Authoritative OWL Reasoner*. Based upon observed characteristics of web data and reasoning in general, we design our system to scale: our system is based upon a separation of terminological data from assertional data and comprises of a lightweight in-memory index, on-disk sorts and file-scans. We evaluate our methods on a dataset in the order of a hundred million statements collected from real-world web sources and present scale-up experiments on a dataset in the order of a billion statements collected from the Web.

1 Introduction

Information attainable through the Web is unique in terms of scale and diversity. The Semantic Web movement aims to bring order to this information by providing a stack of technologies, the core of which is the Resource Description Framework (RDF) for publishing data in a machine-readable format: there now exists millions of RDF data-sources on the Web contributing billions of statements. The Semantic Web technology stack includes means to supplement instance data being published in RDF with ontologies described in RDF Schema (RDFS) [4] and the Web Ontology Language (OWL) [2, 41], allowing people to formally specify a domain of discourse, and providing machines a more sapient understanding of the data. In particular, the enhancement of assertional data (i.e., instance data) with terminological data (i.e., structural data) published in ontologies allows for deductive reasoning: i.e., inferring implicit knowledge.

In particular, our work on reasoning is motivated by the requirements of the Semantic Web Search Engine (SWSE) project: `http://swse.deri.org/`, within which we strive to offer search, querying and browsing over data taken from the Semantic Web. Reasoning over aggregated web data is useful, for example: to infer new assertions using terminological knowledge from ontologies and therefore provide a more complete dataset; to unite fractured knowledge (as is common on the Web in the absence of restrictive formal agreement on identifiers) about individuals collected from disparate sources; and to execute mappings between domain descriptions and thereby provide translations from one conceptual model to another. The ultimate goal here is to provide a "global knowledge-base", indexed by machines, providing querying over both the explicit

*A preliminary version of this article has been accepted at ASWC 2008 [24]. Compared to that version, we have added significant material. The added contributions in this version include (i) a better formalisation of authoritative reasoning, (ii) improvements in the algorithms, and (iii) respectively updated experimental results with additional metrics on a larger dataset. We thank the anonymous reviewers of this and related papers for their valuable feedback. This work has been supported by Science Foundation Ireland project Lion (SFI/02/CE1/I131), European FP6 project inContext (IST-034718), COST Action "Agreement Technologies" (IC0801) and an IRCSET Postgraduate Research Scholarship.

knowledge published on the Web and the implicit knowledge inferable by machine. However, as we will show, complete inferencing on the Web is an infeasible goal, due firstly to the complexity of such a task and secondly to noisy web data; we aim instead to strike a comprise between the above goals for reasoning and what is indeed feasible for the Web.

Current systems have had limited success in exploiting ontology descriptions for reasoning over RDF web data. While there exists a large body of work in the area of reasoning algorithms and systems that work and scale well in confined environments, the distributed and loosely coordinated creation of a world-wide knowledge-base creates new challenges for reasoning:

- the system has to perform on web scale, with implications on the completeness of the reasoning procedure, algorithms and optimisations;
- the method has to perform on collaboratively created knowledge-bases, which has implications on trust and the privileges of data publishers.

With respect to the first requirement, many systems claim to inherit their scalability from the underlying storage – usually some relational database system – with many papers having been dedicated to optimisations on database schemata and access (c.f. [35, 44, 48, 25]). With regards the second requirement, there have been numerous papers dedicated to the inter-operability of a small number of usually trustworthy ontologies (c.f. [13, 31, 27]). We leave further discussion of related work to Section 6, except to state that the combination of web scale and web tolerant reasoning has received little attention in the literature and that our approach is novel.

Our system, which we call "Scalable Authoritative OWL Reasoner" (SAOR), is designed to accept as input a web knowledge-base in the form of a body of statements as produced by a web crawl and to output a knowledge-base enhanced by forward-chaining reasoning over a given fragment of OWL. In particular, we choose forward-chaining to avoid the runtime complexity of query-rewriting associated with backward-chaining approaches: in the web search scenario, the requirement for low query response times and resource usage preclude the applicability of query-rewriting for many reasoning tasks.

SAOR adopts a standard rule-based approach to reasoning whereby each rule consists of (i) an 'antecedent': a clause which identifies a graph pattern that, when matched by the data, allows for the rule to be executed and (ii) a 'consequent': the statement(s) that can be inferred given data that match the antecedent. Within SAOR, we view reasoning as a once-off rule-processing task over a given set of statements. Since the rules are all known *a-priori*, and all require simultaneous execution, we can design a task-specific system that offers much greater optimisations over more general rule engines. Firstly, we categorise the known rules according to the composition of their antecedents (e.g., with respect to arity, proportion of terminological and assertional patterns, etc.) and optimise each group according to the observed characteristics. Secondly, we do not use an underlying database or native RDF store and opt for implementation using fundamental data-structures and primitive operations; our system is built from scratch specifically (and only) for the purpose of performing pre-runtime forward-chaining reasoning which gives us greater freedom in implementing appropriate task-specific optimisations.

This paper is an extended version of [24], in which we presented an initial *modus-operandi* of SAOR; we provided some evaluation of a set of rules which exhibited linear scale and concluded that using dynamic index structures, in SAOR, for more complex rulesets, was not a viable solution for a large-scale reasoner. In this paper, we provide extended discussion of our fragment of OWL reasoning and additional motivation for our deliberate incompleteness in terms of computational complexity and impediments posed by web data considerations. We also describe an implementation of SAOR which abandons dynamic index structures in favour of batch processing techniques known to scale: namely sorts and file-scans. We present new evaluation of the adapted system over a dataset of 147m triples collected from 665k web sources and also provide scale-up evaluation of our most optimised ruleset on a dataset of 1.1b statements collected from 6.5m web sources.

Specifically, we make the following contributions in this paper:

- We discuss and apply a selected rule-based subset of OWL reasoning, i) to be computationally efficient, ii) to avoid an explosion of inferred statements, iii) to be tolerant to noisy web data and iv) to protect existing specifications from undesirable contributions made in independent locations. That is, our system implements a positive fragment of OWL Full which has roots in ter Horst's pD* [43] entailment

rules and our system includes analysis of the authority of sources to counter-act the problem of *ontology hijacking* in web data (Section 3).

- We describe a scalable, optimised method for performing rule-based forward-chaining reasoning for our fragment of OWL. In particular, we refine our algorithm to capitalise on the similarities present in different rule antecedent patterns and the low volume of terminological data relative to assertional data. We implement the system using on-disk batch processing operations known to scale: sorts and scans (Section 4).
- We show experimentally that a forward-chaining materialisation approach is feasible on web data, showing that, by careful materialisation through our tailored OWL ruleset, we can avoid an explosion of inferred statements. We present evaluation with respect to computation of our most expressive ruleset on a dataset of 147m statements collected from 665k sources and present scale-up measurements by applying our most optimised ruleset on a dataset of 1.1b statements collected from 6.5m sources. We also reveal that the most computationally efficient segment of our reasoning is the most productive with regards inferred output statements (Section 5).

We discuss related work in Section 6 and conclude with Section 7.

2 Preliminaries

Before we continue, we briefly introduce some concepts prevalent throughout the paper. We use notation and nomenclature as is popular in the literature, particularly from [22].

RDF Term Given a set of URI references $\mathcal{U}$, a set of blank nodes $\mathcal{B}$, and a set of literals $\mathcal{L}$, the set of *RDF terms* is denoted by $\mathcal{RDFT}erm = \mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$. The set of blank nodes $\mathcal{B}$ is a set of existensially quantified variables. The set of literals is given as $\mathcal{L} = \mathcal{L}_p \cup \mathcal{L}_t$, where $\mathcal{L}_p$ is the set of *plain literals* and $\mathcal{L}_t$ is the set of *typed literals*. A typed literal is the pair $l = (s,t)$, where s is the lexical form of the literal and $t \in \mathcal{U}$ is a datatype URI. The sets $\mathcal{U}$, $\mathcal{B}$, $\mathcal{L}_p$ and $\mathcal{L}_t$ are pairwise disjoint.

RDF Triple A triple $t = (s, p, o) \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$ is called an *RDF triple*. In a triple (s, p, o), s is called subject, p predicate, and o object.

RDF Triple in Context/RDF Quadruple A pair (t, c) with a triple $t = (s, p, o)$ and $c \in \mathcal{U}$ is called a *triple in context* c [16, 20]. We may also refer to (s, p, o, c) as the *RDF quadruple* or quad q with context c.

We use the term 'RDF statement' to refer generically to triple or quadruple where differentiation is not pertinent.

RDF Graph/Web Graph An *RDF graph* $\mathcal{G}$ is a set of RDF triples; that is, a subset of $(\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$.

We refer to a *web graph* $\mathcal{W}$ as a graph derived from a given web location (i.e., a given document). We call the pair $(\mathcal{W}, c)$ a web graph $\mathcal{W}$ in context c, where c is the web location from which $\mathcal{W}$ is retrieved. Informally, $(\mathcal{W}, c)$ is represented as the set of quadruples (t_w, c) for all $t_w \in \mathcal{W}$.

Generalised Triple A triple $t = (s, p, o) \in (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L}) \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L}) \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$ is called a *generalised triple*.

The notions of generalised quadruple, generalised statement and generalised graph follow naturally. Our definition of "generalised" is even more liberal than that described in [43] wherein blank nodes are allowed in the predicate position: we also allow literals in the subject and predicate position. Please note that we may refer generically to a "triple", "quadruple", "graph" etc. where a distinction between the "generalised" and "RDF" versions is not pertinent.

Merge The *merge* $\mathrm{M}(\mathcal{S})$ of a set of graphs $\mathcal{S}$ is the union of the set of all graphs $\mathcal{G}'$ for $\mathcal{G} \in \mathcal{S}$ and $\mathcal{G}'$ derived from $\mathcal{G}$ such that $\mathcal{G}'$ contains a unique set of blank nodes for $\mathcal{S}$.

Web Knowledge-base Given a set $\mathcal{S}_\mathcal{W}$ of RDF web graphs, our view of a *web knowledge-base* $\mathbb{KB}$ is taken as a set of pairs $(\mathcal{W}', c)$ for each $\mathcal{W} \in \mathcal{S}_\mathcal{W}$, where $\mathcal{W}'$ contains a unique set of blank nodes for $\mathcal{S}_\mathcal{W}$ and c denotes the URL location of $\mathcal{W}$.

Informally, $\mathbb{KB}$ is a set of quadruples retrieved from the Web wherein the set of blank nodes are unique for a given document and triples are enhanced by means of context which tracks the web location from which each triple is retrieved. We use the abbreviated notation $\mathcal{W} \in \mathbb{KB}$ or $\mathcal{W}' \in \mathbb{KB}$ where we mean $\mathcal{W} \in \mathcal{S}_\mathcal{W}$ for $\mathcal{S}_\mathcal{W}$ from which $\mathbb{KB}$ is derived or $(\mathcal{W}', c) \in \mathbb{KB}$ for some c.

Class We refer to a *class* as an RDF term which appears in either

- o of a triple t where p is `rdf:type`; or
- s of a triple t where p is `rdf:type` and o is `rdfs:Class` or `:Class`[1].

Property We refer to a *property* as an RDF term which appears in either

- p of a triple t; or
- s of a triple t where p is `rdf:type` and o is `rdf:Property`.

Membership Assertion We refer to a triple t as a *membership assertion* of the property mentioned in predicate position p. We refer to a triple t with predicate `rdf:type` as a membership assertion of the class mentioned in the object o. For a class or property v, we denote a membership assertion as *m(v)*.

Meta-class A *meta-class* is a class of classes or properties; i.e., the members of a meta-class are either classes or properties. The set of RDF(S) and OWL meta-classes is as follows: {`rdf:Property`, `rdfs:Class`, `rdfs:ContainerMembershipProperty`, `:AnnotationProperty`, `:Class`, `:DatatypeProperty`, `:DeprecatedClass`, `:DeprecatedProperty`, `:FunctionalProperty`, `:InverseFunctionalProperty`, `:ObjectProperty`, `:OntologyProperty`, `:Restriction`, `:SymmetricProperty`, `:TransitiveProperty` }.

Meta-property A *meta-property* is one which has a meta-class as it's domain. Meta-properties are used to describe classes and properties. The set of RDFS and OWL meta-properties is as follows: {`rdfs:domain`, `rdfs:range`, `rdfs:subClassOf`, `rdfs:subPropertyOf`, `:allValuesFrom`, `:cardinality`, `:complementOf`, `:disjointWith`, `:equivalentClass`, `:equivalentProperty`, `:hasValue`, `:intersectionOf`, `:inverseOf`, `:maxCardinality`, `:minCardinality`, `:oneOf`, `:onProperty`, `:someValuesFrom`, `:unionOf`}.

Terminological Triple We define a *terminological triple* as one of the following:

1. a membership assertion of a meta-class;
2. a membership assertion of a meta-property;
3. a triple in a non-branching, non-cyclic path $t^r_0, ..., t^r_n$ where $t^r_0 = (s_0, p_0, o_0)$ for $p_0 \in$ {`:intersectionOf`, `:oneOf`, `:unionOf` }; $t^r_k = (o_{k-1}$, `rdf:rest`, $o_k)$ for $1 \leq k \leq n$, $o_{k-1} \in \mathcal{B}$ and $o_n =$`rdf:nil`; or a triple $t^f_k = (o_k$, `rdf:first`, $e_k)$ with o_k for $0 \leq k < n$ as before.

We refer to triples $t^r_1, ..., t^r_n$ and all triples t^f_k as *terminological collection triples*, whereby RDF collections are used in a union, intersection or enumeration class description.

[1] Throughout this paper, we assume that `http://www.w3.org/2002/07/owl#` is the default namespace with prefix ":", i.e. we write e.g. just ":`Class`", ":`disjointWith`", etc. instead of using the commonly used `owl:` prefix. Other prefixes such as `rdf:`, `rdfs:`, `foaf:` are used as in other common documents. Moreover, we often use the common abbreviation '`a`' as a convenient shortcut for `rdf:type`.

Triple Pattern, Basic Graph Pattern A *triple pattern* is defined as a generalised triple where, in all positions, variables from the infinite set $\mathcal{V}$ are allowed; i.e.: $tp = (s_v, p_v, o_v) \in (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V})$. A set (to be read as conjunction) of triple patterns $\mathcal{GP}$ is also called a *basic graph pattern*.

We use – following SPARQL notation [38] – alphanumeric strings preceded by '`?`' to denote variables in this paper: e.g., `?X`. Following common notation, such as is used in SPARQL [38] and Turtle[2], we delimit triples in the same basic graph pattern by '.' and we may group triple patterns with the same subject or same subject-predicate using ';' and ',' respectively. Finally, we denote by $\mathcal{V}(tp)$ (or $\mathcal{V}(\mathcal{GP})$, resp.) the set of variables appearing in tp (or in $\mathcal{GP}$, resp.).

Instance A triple $t = (s, p, o)$ (or, resp., a set of triples, i.e., a graph $\mathcal{G}$) is an *instance* of a triple pattern $tp = (s_v, p_v, o_v)$ (or, resp., of a basic graph pattern $\mathcal{GP}$) if there exists a mapping $\mu : \mathcal{V} \cup \mathcal{RDFT}erm \rightarrow \mathcal{RDFT}erm$ which maps every element of $\mathcal{RDFT}erm$ to itself, such that $t = \mu(tp) = (\mu(s_v), \mu(p_v), \mu(o_v))$ (or, resp., and slightly simplifying notation, $G = \mu(\mathcal{GP})$).

Terminological/Assertional Pattern We refer to a *terminological -triple/-graph pattern* as one whose instance can only be a terminological triple or, resp., a set thereof. We denote a *terminological collection pattern* by `?x` p `(?e`$_1$`, ..., ?e`$_n$`)` . where $p \in \{$`:intersectionOf`, `:oneOf`, `:unionOf` $\}$ and `?e`$_k$ is mapped by the object of a terminological collection triple (o$_k$, `rdf:first`, e$_k$) for $o_k \in \{o_0, ..., o_{n-1}\}$ as before. An *assertional pattern* is any pattern which is not terminological.

Inference Rule We define an *inference rule* r as the pair $(\mathcal{A}nte, \mathcal{C}on)$, where the *antecedent* $\mathcal{A}nte$ and the *consequent* $\mathcal{C}on$ are basic graph patterns such that $\mathcal{V}(\mathcal{C}on)$ and $\mathcal{V}(\mathcal{A}nte)$ are non-empty, $\mathcal{V}(\mathcal{C}on) \subseteq \mathcal{V}(\mathcal{A}nte)$ and $\mathcal{C}on$ does not contain blank nodes[3]. In this paper, we will typically write inference rules as:

$$\mathcal{A}nte \Rightarrow \mathcal{C}on \tag{1}$$

Rule Application and Closure We define a *rule application* in terms of the immediate consequences of a rule r or a set of rules $\mathcal{R}$ on a graph $\mathcal{G}$ (here slightly abusing the notion of the immediate consequence operator in Logic Programming: cf. for example [30]). That is, if r is a rule of the form (1), and $\mathcal{G}$ is a set of RDF triples, then:

$$T_r(\mathcal{G}) = \{\mu(\mathcal{C}on) \mid \exists \mu \text{ such that } \mu(\mathcal{A}nte) \subseteq G\}$$

and accordingly $T_{\mathcal{R}}(\mathcal{G}) = \bigcup_{r \in \mathcal{R}} T_r(\mathcal{G})$. Also, let $\mathcal{G}_{i+1} = \mathcal{G}_i \cup T_{\mathcal{R}}(\mathcal{G}_i)$ and $\mathcal{G}_0 = \mathcal{G}$; we now define the *exhaustive application* of the $T_{\mathcal{R}}$ operator on a graph $\mathcal{G}$ as being upto the least fixpoint (the smallest value for n) such that $\mathcal{G}_n = T_{\mathcal{R}}(\mathcal{G}_n)$. We call $\mathcal{G}_n$ the *closure* of $\mathcal{G}$ with respect to ruleset $\mathcal{R}$, denoted as $Cl_{\mathcal{R}}(\mathcal{G})$. Note that we may also use the intuitive notation $Cl_{\mathcal{R}}(\mathbb{KB})$, $T_{\mathcal{R}}(\mathbb{KB})$ as shorthand for the more cumbersome $Cl_{\mathcal{R}}(\bigcup_{\mathcal{W}' \in \mathbb{KB}} \mathcal{W}')$, $T_{\mathcal{R}}(\bigcup_{\mathcal{W}' \in \mathbb{KB}} \mathcal{W}')$.

Ground Triple/Graph A *ground triple* or *ground graph* is one without existential variables.

Herbrand Interpretation Briefly, a *Herbrand interpretation* of a graph $\mathcal{G}$ treats URI references, blank nodes, typed literals and plain literals analogously as denoting their own syntactic form. As such, a Herbrand interpretation represents a ground view of an RDF graph where blank nodes are treated as *Skolem* names instead of existential variables; i.e., blank nodes are seen to represent the entities that they assert the existence of, analogously to a URI reference. Henceforth, we view blank nodes as their Skolem equivalents (this also applies to blank nodes as mentioned in the above notation) and only treat the ground case of RDF graphs.

Let us elaborate in brief why this treatment of blank nodes as Skolem constants is sufficient for our purposes. In our scenario, we perform forward-chaining materialisation for query-answering and not "real"

[2] `http://www.dajobe.org/2004/01/turtle/`

[3] Unlike some other rule systems for RDF, the most prominent of which being CONSTRUCT statements in SPARQL, we forbid blank nodes; i.e., we forbid existential variables in rule consequents which would require the "invention" of blank nodes.

entailment checks between RDF graphs. This enables us to treat all blank nodes as Skolem names [22]. It is well known that simple entailment checking of two RDF graphs [22] – i.e., checking whether an RDF graph $\mathcal{G}_1$ entails $\mathcal{G}_2$ – can be done using the ground "skolemised" version of $\mathcal{G}_1$. That is $\mathcal{G}_1 \models \mathcal{G}_2$ iff $sk(\mathcal{G}_1) \models \mathcal{G}_2$. Likewise, given a set of inference rules $\mathcal{R}$, where we denote entailment with respect to $\mathcal{R}$ as $\models_{\mathcal{R}}$, it is again well known that such entailment can be reduced to simple entailment with prior computation of the inference closure with respect to $\mathcal{R}$. That is, $\mathcal{G}_1 \models_{\mathcal{R}} \mathcal{G}_2$ iff $Cl_{\mathcal{R}}(sk(\mathcal{G}_1)) \models \mathcal{G}_2$, cf. [22, 18]. In this paper we focus on the actual computation of $Cl_{\mathcal{R}}(sk(\mathcal{G}_1))$ for a tailored ruleset $\mathcal{R}$ in between RDFS and OWL Full.

3 Pragmatic Inferencing for the Web

In this section we discuss the inference rules which we use to approximate OWL semantics and are designed for forward-chaining reasoning over web data. We justify our selection of inferences to support in terms of observed characteristics and examples taken from the Web. We optimise by restricting our fragment of reasoning according to three imperatives: *computational feasibility (CF)* for scalability, *reduced output* statements *(RO)* to ease the burden on consumer applications and, finally, *web tolerance (WT)* for avoiding undesirable inferences given noisy data and protecting publishers from unwanted, independent third-party contributions. In particular, we adhere to the following high-level restrictions:

1. we are incomplete *(CF, RO, WT)* - Section 3.1;
2. we deliberately ignore the explosive behaviour of classical inconsistency *(CF, RO, WT)* - Section 3.1;
3. we follow a rule-based, finite, forward-chaining approach to OWL inference *(CF)* - Section 3.2;
4. we do not invent new blank nodes *(CF, RO, WT)* - Section 3.2;
5. we avoid inference of *extended-axiomatic* triples *(RO)* - Section 3.2;
6. we focus on inference of non-terminological statements *(CF)* - Section 3.2;
7. we do not consider `:sameAs` statements as applying to terminological data *(CF, WT)* - Section 3.2;
8. we separate and store terminological data in-memory *(CF)* - Section 3.3;
9. we support limited reasoning for *non-standard use* of the RDF(S) and OWL vocabularies *(CF, RO, WT)* - Section 3.3;
10. we ignore *non-authoritative* (third-party) terminological statements from our reasoning procedure to counter an explosion of inferred statements caused by *hijacking* ontology terms *(RO, WT)* - Section 3.4.

3.1 Infeasibility of Complete Web Reasoning

Reasoning over RDF data is enabled by the description of RDF terms using the RDFS and OWL standards; these standards have defined entailments determined by their semantics. The semantics of these standards differs in that RDFS entailment is defined in terms of "if" conditions (intensional semantics), and has a defined set of complete standard entailment rules [22]. OWL semantics uses "iff" conditions (extensional semantics) without a complete set of standard entailment rules. RDFS entailment has been shown to be decidable and in P for the ground case [43], whilst OWL Full entailment is known to be undecidable [26]. Thus, the OWL standard includes two restricted fragments of OWL whose entailment is known to be decidable from work in description logics: (i) OWL DL whose worst-case entailment is in NEXPTIME (ii) OWL Lite whose worst-case entailment is in EXPTIME [26].

Although entailment for both fragments is known to be decidable, and even aside from their complexity, most OWL ontologies crawlable on the Web are in any case OWL Full: idealised assumptions made in OWL DL are violated by even very commonly used ontologies. For example, the popular Friend Of A Friend

(FOAF) vocabulary [5] deliberately falls into OWL Full since, in the FOAF RDF vocabulary[4], `foaf:name` is defined as a sub-property of the core RDFS property `rdfs:label` and `foaf:mbox_sha1sum` is defined as both an `:InverseFunctionalProperty` and a `:DatatypeProperty`: both are disallowed by OWL DL (and, of course, OWL Lite). In [3], the authors identified and categorised OWL DL restrictions violated by a sample group of 201 OWL ontologies (all of which were found to be in OWL Full); these include incorrect or missing typing of classes and properties, complex object-properties (e.g., functional properties) declared to be transitive, inverse-functional datatype properties, etc. In [46], a more extensive survey with nearly 1,300 ontologies was conducted: 924 were identified as being in OWL Full.

Taking into account that most web ontologies are in OWL Full, and also the undecidability/computational-infeasiblity of OWL Full, one could conclude that complete reasoning on the Web is impractical. However, again for most web documents only categorisable as OWL Full, infringements are mainly syntactic and are rather innocuous with no real effect on decidability ([46] showed that the majority of web documents surveyed were in the base expressivity for Description Logics after patching infringements).

The main justification for the infeasibility of complete reasoning on the Web is inconsistency.

Consistency cannot be expected on the Web; for instance, a past web crawl of ours revealed the following:

```
w3:timbl a foaf:Person; foaf:homepage <http://w3.org/> .
w3:w3c a foaf:Organization; foaf:homepage <http://w3.org/> .
foaf:homepage a :InverseFunctionalProperty .
foaf:Organization :disjointWith foaf:Person .
```

These triples together infer that Tim Berners-Lee is the same as the W3C and thus cause an inconsistency.[5] Aside from such examples which arise from misunderstanding of the FOAF vocabulary, there might be cases where different parties deliberately make contradictive statements; resolution of such contradictions could involve "choosing sides". In any case, the explosive nature of contradiction in classical logics suggests that it is not desirable within our web reasoning scenario.

3.2 Rule-based Web Reasoning

As previously alluded to, there does not exist a standard entailment for OWL suitable to our web reasoning scenario. However, incomplete (wrt. OWL Full) rule-based inference (i.e., reasoning as performed by logic progamming or deductive database engines) may be considered to have greater potential for scale, following the arguments made in [12] and may be considered to be more robust with respect to preventing explosive inferencing through inconsistencies. Several rule expressible non-standard OWL fragments; namely OWL-DLP [15], OWL$^-$ [10] (which is a slight extension of OWL-DLP), OWLPrime [47], pD* [42, 43], and Intensional OWL [9, Section 9.3]; have been defined in the literature and enable incomplete but sound RDFS and OWL Full inferences.

In [42, 43], pD* was introduced as a combination of RDFS entailment, datatype reasoning and a distilled version of OWL with rule-expressible intensional semantics: pD* entailment maintains the computational complexity of RDFS entailment, which is in NP in general and P for the ground case. Such improvement in complexity has obvious advantages in our web reasoning scenario; thus SAOR's approach to reasoning is inspired by the pD* fragment to cover large parts of OWL by positive inference rules which can be implemented in a forward-chaining engine.

Table 1 summarises the pD* ruleset. The rules are divided into D*-entailment rules and P-entailment rules. D*-entailment is essentially RDFS entailment [22] combined with some datatype reasoning. P-entailment is introduced in [43] as a set of rules which applies to a property-related subset of OWL.

Given pD*, we make some amendments so as to align the ruleset with our requirements. Table 2 provides a full listing of our own modified ruleset, which we compare against pD* in this section. Note that this table highlights characteristics of the rules which we will discuss in Section 3.3 and Section 3.4; for the moment we point out that **rule′** is used to indicate an amendment to the respective pD* rule. Please also note that we use the notation **rulex*** to refer to all rules with the prefix **rulex**.

[4] `http://xmlns.com/foaf/spec/index.rdf`

[5] Tim (now the same entity as the W3C) is asserted to be a member of the two disjoint classes: `foaf:Person` and `foaf:Organization`.

pD*	rule	where
	D*-entailment rules	
lg	?x ?P ?l . ⇒ ?v ?P _:*bl* .	?l∈ $\mathcal{L}$[a]
gl	?x ?P _:*bl* . ⇒ ?x ?P ?l .	?l∈ $\mathcal{L}$
rdf1	?x ?P ?y . ⇒ ?P a rdf:Property .	
rdf2-D	?x ?P ?l . ⇒ _:*bl* ?type ?t .	?l= $(s,t) \in \mathcal{L}_t$
rdfs1	?x ?P ?l . ⇒ _:*bl* a Literal .	?l∈ $\mathcal{L}_p$
rdfs2	?P rdfs:domain ?C . ?x ?P ?y . ⇒ ?x a ?C .	
rdfs3	?P rdfs:range ?C . ?x ?P ?y . ⇒ ?y a ?C .	?y $\in \mathcal{U} \cup \mathcal{B}$
rdfs4a	?x ?P ?y . ⇒ ?x a rdfs:Resource .	
rdfs4b	?x ?P ?y . ⇒ ?y a rdfs:Resource .	?y∈ $\mathcal{U} \cup \mathcal{B}$
rdfs5	?P rdfs:subProperty ?Q . ?Q rdfs:subProperty ?R . ⇒ ?P rdfs:subProperty ?R .	
rdfs6	?P a rdf:Property . ⇒ ?P rdfs:subProperty ?P .	
rdfs7	?P rdfs:subProperty ?Q . ?x ?P ?y . ⇒ ?x ?Q ?y .	?Q∈ $\mathcal{U} \cup \mathcal{B}$
rdfs8	?C a rdfs:Class . ⇒ ?C rdfs:subClassOf rdfs:Resource .	
rdfs9	?C rdfs:subClassOf ?D . ?x a ?C . ⇒ ?x a ?D .	
rdfs10	?C a rdfs:Class . ⇒ ?C rdfs:subClassOf ?C .	
rdfs11	?C rdfs:subClassOf ?D . ?D rdfs:subClassOf ?E . ⇒ ?C rdfs:subClassOf ?E .	
rdfs12	?P a rdfs:ContainerMembershipProperty . ⇒ ?P rdfs:subPropertyOf rdfs:member .	
rdfs13	?D a rdfs:Datatype . ⇒ ?D rdfs:subClassOf rdfs:Literal .	
	***P*-entailment rules**	
rdfp1	?P a :FunctionalProperty . ?x ?P ?y , ?z . ⇒ ?y :sameAs ?z .	?y $\in \mathcal{U} \cup \mathcal{B}$
rdfp2	?P a :InverseFunctionalProperty . ?x ?P ?z . ?y ?P ?z . ⇒ ?x :sameAs ?y .	
rdfp3	?P a :SymmetricProperty . ?x ?P ?y . ⇒ ?y ?P ?x .	?y $\in \mathcal{U} \cup \mathcal{B}$
rdfp4	?P a :TransitiveProperty . ?x ?P ?y . ?y ?P ?z . ⇒ ?x ?P ?z .	
rdfp5a	?x ?P ?y . ⇒ ?x :sameAs ?x .	
rdfp5b	?x ?P ?y . ⇒ ?y :sameAs ?y .	?y∈ $\mathcal{U} \cup \mathcal{B}$
rdfp6	?x :sameAs ?y . ⇒ ?y :sameAs ?x .	?y∈ $\mathcal{U} \cup \mathcal{B}$
rdfp7	?x :sameAs ?y . ?y :sameAs ?z . ⇒ ?x :sameAs ?z .	
rdfp8a	?P :inverseOf ?Q . ?x ?P ?y . ⇒ ?y ?Q ?x .	?y,?Q∈ $\mathcal{U} \cup \mathcal{B}$
rdfp8b	?P :inverseOf ?Q . ?x ?Q ?y . ⇒ ?y ?P ?x .	?y $\in \mathcal{U} \cup \mathcal{B}$
rdfp9	?C a :Class ; :sameAs ?D . ⇒ ?C rdfs:subClassOf ?D .	
rdfp10	?P a :Property ; :sameAs ?Q . ⇒ ?P rdfs:subPropertyOf ?Q .	
rdfp11	?x :sameAs ?_x . ?y :sameAs ?_y . ?x ?P ?y .⇒ ?_x ?P ?_y .	?_x $\in \mathcal{U} \cup \mathcal{B}$
rdfp12a	?C :equivalentClass ?D . ⇒ ?C rdfs:subClassOf ?D .	
rdfp12b	?C :equivalentClass ?D . ⇒ ?D rdfs:subClassOf ?C .	?D∈ $\mathcal{U} \cup \mathcal{B}$
rdfp12c	?C rdfs:subClassOf ?D . ?D rdfs:subClassOf ?C . ⇒ ?C :equivalentClass ?D .	
rdfp13a	?P :equivalentProperty ?Q . ⇒ ?P rdfs:subPropertyOf ?Q .	
rdfp13b	?P :equivalentProperty ?Q . ⇒ ?Q rdfs:subPropertyOf ?P .	?Q∈ $\mathcal{U} \cup \mathcal{B}$
rdfp13c	?P rdfs:subPropertyOf ?Q . ?Q rdfs:subPropertyOf ?P . ⇒ ?P :equivalentProperty ?Q .	
rdfp14a	?C :hasValue ?y ; :onProperty ?P . ?x ?P ?y . ⇒ ?x a ?C .	
rdfp14b	?C :hasValue ?y ; :onProperty ?P . ?x a ?C . ⇒ ?x ?P ?y .	?P∈ $\mathcal{U} \cup \mathcal{B}$
rdfp15	?C :someValuesFrom ?D ; :onProperty ?P . ?x ?P ?y . ?y a ?D . ⇒ ?x a ?C .	
rdfp16	?C :allValuesFrom ?D ; :onProperty ?P . ?x a ?C; ?P ?y . ⇒ ?y a ?D .	?y∈ $\mathcal{U} \cup \mathcal{B}$

Table 1: Ter-Horst rules from [42, 43] in Turtle-style syntax

[a] `_:bl` is a *surrogate blank node* given by an injective function on the literal `?l`

pD* Rules Directly Supported From the set of pD* rules, we directly support rules **rdfs2**, **rdfs9**, **rdfp2**, **rdfp4**, **rdfp7**, and **rdfp17**.

pD* Omissions: Extended-Axiomatic Statements We avoid pD* rules which specifically produce what we term *extended-axiomatic* statements mandated by RDFS and OWL semantics. Firstly, we do not infer the set of pD* axiomatic triples, which are listed in [43, Table 3] and [43, Table 6] for RDF(S) and OWL respectively; according to pD*, these are inferred for the empty graph. Secondly, we do not materialise membership assertions for `rdfs:Resource` which would hold for every URI and blank node in a graph. Thirdly, we do not materialise reflexive `:sameAs` membership assertions, which again hold for every URI and blank node in a graph. We see such statements as inflationary and orthogonal to our aim of reduced output.

pD* Amendments: `:sameAs` Inferencing From the previous set of omissions, we do not infer reflexive `:sameAs` statements. However, such reflexive statements are required by pD* rule **rdfp11**. We thus fragment the rule into **rdfp11′** and **rdfp11″** which allows for the same inferencing without such reflexive statements.

In a related issue, we wittingly do not allow `:sameAs` inferencing to interfere with terminological data: for example, we do not allow `:sameAs` inferencing to affect properties in the predicate position of a triple or classes in the object position of an `rdf:type` triple. In [23] we showed that `:sameAs` inferencing through `:InverseFunctionalProperty` reasoning caused fallacious equalities to be asserted due to noisy web data.

SAOR	rule	where
	$\mathcal{R}0$: only terminological patterns in antecedent	
rdfc0	?C :oneOf (?x$_1$ ... ?x$_n$) . ⇒ ?x$_1$ ... ?x$_n$ a ?C .	?C ∈ $\mathcal{B}$
	$\mathcal{R}1$: at least one terminological/only one assertional pattern in antecedent	
rdfs2	**?P** rdfs:domain ?C . ?x ?P ?y . ⇒ ?x a ?C .	
rdfs3′	**?P** rdfs:range ?C . ?x ?P ?y . ⇒ ?y a ?C .	
rdfs7′	**?P** rdfs:subPropertyOf ?Q . ?x ?P ?y . ⇒ ?x ?Q ?y .	
rdfs9	**?C** rdfs:subClassOf ?D . ?x a ?C . ⇒ ?x a ?D .	
rdfp3′	**?P** a :SymmetricProperty . ?x ?P ?y . ⇒ ?y ?P ?x .	
rdfp8a′	**?P** :inverseOf ?Q . ?x ?P ?y . ⇒ ?y ?Q ?x .	
rdfp8b′	?P :inverseOf **?Q** . ?x ?Q ?y . ⇒ ?y ?P ?x .	
rdfp12a′	**?C** :equivalentClass ?D . ?x a ?C . ⇒ ?x a ?D .	
rdfp12b′	?C :equivalentClass **?D** . ?x a ?D . ⇒ ?x a ?C .	
rdfp13a′	**?P** :equivalentProperty ?Q . ?x ?P ?y . ⇒ ?y ?Q ?x .	
rdfp13b′	?P :equivalentProperty **?Q** . ?x ?Q ?y . ⇒ ?y ?P ?x .	
rdfp14a′	?C :hasValue ?y ; :onProperty **?P** . ?x ?P ?y . ⇒ ?x a ?C .	?C ∈ $\mathcal{B}$
rdfp14b′	?C :hasValue ?y ; :onProperty ?P . ?x a ?C . ⇒ ?x ?P ?y .	?C ∈ $\mathcal{B}$
rdfc1	?C :unionOf (?C$_1$...**?C**$_i$...?C$_n$) . ?x a ?C$_i$ [a] . ⇒ ?x a ?C .	?C ∈ $\mathcal{B}$
rdfc2	?C :minCardinality 1 ; :onProperty **?P** . ?x ?P ?y . ⇒ ?x a ?C .	?C ∈ $\mathcal{B}$
rdfc3a	?C :intersectionOf (?C$_1$ ... ?C$_n$) . ?x a ?C . ⇒ ?x a ?C$_1$, ..., ?C$_n$.	?C ∈ $\mathcal{B}$
rdfc3b	?C :intersectionOf (**?C**$_1$) . ?x a ?C$_1$. ⇒ ?x a ?C . [b]	?C ∈ $\mathcal{B}$
	$\mathcal{R}2$: at least one terminological/multiple assertional patterns in antecedent	
rdfp1′	**?P** a :FunctionalProperty . ?x ?P ?y , ?z . ⇒ ?y :sameAs ?z .	
rdfp2	**?P** a :InverseFunctionalProperty . ?x ?P ?z . ?y ?P ?z . ⇒ ?x :sameAs ?y .	
rdfp4	**?P** a :TransitiveProperty . ?x ?P ?y . ?y ?P ?z . ⇒ ?x ?P ?z .	
rdfp15′	?C :someValuesFrom **?D** ; :onProperty **?P** . ?x ?P ?y . ?y a ?D . ⇒ ?x a ?C .	?C ∈ $\mathcal{B}$
rdfp16′	?C :allValuesFrom ?D ; :onProperty ?P . ?x a ?C ; ?P ?y . ⇒ ?y a ?D .	?C ∈ $\mathcal{B}$
rdfc3c	?C :intersectionOf (**?C**$_1$ **... ?C**$_n$) . ?x a ?C$_1$, ..., ?C$_n$. ⇒ ?x a ?C .	?C ∈ $\mathcal{B}$
rdfc4a	?C :cardinality 1 ; :onProperty ?P . ?x a ?C ; ?P ?y , ?z . ⇒ ?y :sameAs ?z .	?C ∈ $\mathcal{B}$
rdfc4b	?C :maxCardinality 1 ; :onProperty ?P . ?x a ?C ; ?P ?y , ?z . ⇒ ?y :sameAs ?z .	?C ∈ $\mathcal{B}$
	$\mathcal{R}3$: only assertional patterns in antecedent	
rdfp6′	?x :sameAs ?y . ⇒ ?y :sameAs ?x .	
rdfp7	?x :sameAs ?y . ?y :sameas ?z . ⇒ ?x :sameAs ?z .	
rdfp11′	?x :sameAs ?_x ; ?P ?y .⇒ ?_x ?P ?y . [c]	
rdfp11″	?y :sameAs ?_y . ?x ?P ?y .⇒ ?x ?P ?_y . [c]	

Table 2: Supported rules in Turtle-style syntax. Terminological patterns are underlined whereas assertional patterns are not; further, rules are grouped according to arity of terminological/assertional patterns in the antecedent. The source of a terminological pattern instance must speak authoritatively for at least one boldface variable binding for the rule to fire.

[a] $?C_i \in \{?C_1, ..., ?C_n\}$

[b] **rdfs3b** is a special case of **rdfs3c** with one A-Box pattern and thus falls under $\mathcal{R}1$.

[c] Only where `?p` is not an RDFS/OWL property used in any of our rules (e.g., see $\mathcal{P}_{SAOR}$, Section 3.3)

This is the primary motivation for us also omitting rules **rdfp9**, **rdfp10** and the reason why we place the restriction on `?p` for our rule **rdfp11″**; we do not want noisy equality inferences to be reflected in the terminological segment of our knowledge-base, nor to affect the class and property positions of membership assertions.

pD* Omissions: Terminological Inferences From pD*, we also omit rules which infer only terminological statements: namely **rdf1**, **rdfs5**, **rdfs6**, **rdfs8**, **rdfs10**, **rdfs11**, **rdfs12**, **rdfs13**, **rdfp9**, **rdfp10**, **rdfp12*** and **rdfp13***. As such, our use-case is query-answering over assertional data; we therefore focus in this paper on materialising assertional data.

We have already motivated omission of inference through `:sameAs` rules **rdfp9** and **rdfp10**. Rules **rdf1**, **rdfs8**, **rdfs12** and **rdfs13** infer memberships of, or subclass/subproperty relations to, RDF(S) classes and properties; we are not interested in these primarily syntactic statements which are not directly used

in our inference rules. Rules **rdfs6** and **rdfs10** infer reflexive memberships of `rdfs:subPropertyOf` and `rdfs:subClassOf` meta-properties which are used in our inference rules; clearly however, these reflexive statements will not lead to unique assertional inferences through related rules **rdfs7′** or **rdfs9** respectively. Rules **rdfs5** and **rdfs11** infer transitive memberships again of `rdfs:subPropertyOf` and `rdfs:subClassOf`; again however, exhaustive application of rules **rdfs7′** or **rdfs9** respectively ensures that all possible assertional inferences are materialised without the need for the transitive rules. Rules **rdfp12c** and **rdfp13c** infer additional `:equivalentClass`/`:equivalentProperty` statements from `rdfs:subClassOf`/`rdfs:subPropertyOf` statements where assertional inferences can instead be conducted through two applications each of rules **rdfs9** and **rdfs7′** respectively.

pD* Amendments: Direct Assertional Inferences The observant reader may have noticed that we did not dismiss inferencing for rules **rdfp12a**,**rdfp12b**/**rdfp13a**,**rdfp13b** which translate `:equivalentClass`/`:equivalentProperty` to `rdfs:subClassOf`/`rdfs:subPropertyOf`. In pD*, these rules are required to support indirect assertional inferences through rules **rdfs9** and **rdfs7** respectively; we instead support assertional inferences directly from the `:equivalentProperty`/`:equivalentClass` statements using symmetric rules **rdfp12a′**,**rdfp12b′**/**rdfp13a′**,**rdfp13b′**.

pD* Omissions: Existential Variables in Consequent We avoid rules with existential variables in the consequent; such rules would require adaptation of the T_r operator so as to "invent" new blank nodes for each rule application, with undesireable effects for forward-chaining reasoning regarding termination. For example, like pD*, we only support inferences in one direction for `:someValuesFrom` and avoid a rule such as:

```
?C :someValuesFrom ?D ; :onProperty ?P . ?x a ?C ⇒ ?x ?P _:b .  _:b a ?D .
```

Exhaustive application of the rule to, for example, the following data (more generally where `?D` is a subclass of `?C`):

```
ex:Person rdfs:subClassOf [:someValuesFrom ex:Person ; :onProperty ex:mother .]
_:Tim a ex:Person .
```

would infer infinite triples of the type:

```
_:Tim ex:mother _:b0 .
_:b0 a ex:Person ; ex:mother _:b1 .
_:b1 a ex:Person ; ex:mother _:b2 .
...
```

In fact, this rule is listed in [43] as **rdf-svx** which forms an extension of pD* entailment called *pD*sv*. This rule is omitted from pD* and from SAOR due to obvious side-effects on termination and complexity.

Unlike pD*, we also avoid inventing so called "surrogate" blank nodes for the purposes of representing a literal in intermediary inferencing steps (Rules **lg**, **gl**, **rdf2-D**, **rdfs1** in RDFS/D* entailment). Thus, we also do not support datatype reasoning (Rule **rdf2-D**) which involves the creation of surrogate blank nodes. Although surrogate blank nodes are created according to a direct mapping from a finite set of literals (and thus, do not prevent termination), we view "surrogate statements" as inflationary.

pD* Amendments: Relaxing Literal Restrictions Since we do not support surrogate blank nodes as representing literals, we instead relax restrictions placed on pD* rules. In pD*, blank nodes are allowed in the predicate position of triples; however, the restriction on literals in the subject and predicate position still applies: literals are restricted from travelling to the subject or predicate position of a consequent (see *where* column, Table 1). Thus, surrogate blank nodes are required in pD* to represent literals in positions where they would otherwise not be allowed.

We take a different approach whereby we allow literals directly in the subject and predicate position for intermediate inferences. Following from this, we remove pD* literal restrictions on rules **rdfs3**, **rdfs7**, **rdfp1**, **rdfp3**, **rdfp6**, **rdfp8***, **rdfp14b**, **rdfp16** for intermediate inferences and omit any inferred non-RDF statements from being written to the final output.

Additions to pD* In addition to pD*, we also include some "class based entailment" from OWL, which we call C-entailment. We name such rules using the **rdfc*** stem, following the convention from P-entailment. We provide limited support for enumerated classes (**rdfc0**), union class descriptions (**rdfc1**), intersection class descriptions (**rdfc3***)[6], as well as limited cardinality constraints (**rdfc2**, **rdfc4***).

pD* Amendments: Enforcing OWL Abstract Syntax Restrictions Finally, unlike pD*, we enforce blank nodes as mandated by the OWL Abstract Syntax [37], wherein certain abstract syntax constructs (most importantly in our case: *unionOf(description$_1$...description$_n$)*, *intersectionOf(description$_1$...description$_n$)*, *oneOf(iID$_1$...iID$_n$)*, *restriction(ID allValuesFrom(range))*, *restriction(ID someValuesFrom(required))*, *restriction(ID value(value))*, *restriction(ID maxCardinality(max))*, *restriction(ID minCardinality(min))*, *restriction(ID cardinality(card))* and *SEQ item$_1$...item$_n$*) are strictly mapped to RDF triples with blank nodes enforced for certain positions: such mapping is necessitated by the idiosyncrasies of representing OWL in RDF. Although the use of URIs in such circumstances is allowed by RDF, we enforce the use of blank nodes for terminological patterns in our ruleset; to justify, let us look at the following problematic example of OWL triples taken from two sources:

Example 3.1 :
 # FROM SOURCE `<ex:>`
`ex:Person :onProperty ex:parent ; :someValuesFrom ex:Person .`
 # FROM SOURCE `<ex2:>`
`ex:Person :allValuesFrom ex2:Human .` ◇

According to the abstract syntax mapping, neither of the restrictions should be identified by a URI (if blank nodes were used instead of `ex:Person` as mandated by the abstract syntax, such a problem could not occur as each web-graph is given a unique set of blank nodes). If we consider the RDF-merge of the two graphs, we will be unable to distinguish which restriction the `:onProperty` value should be applied to. As above, allowing URIs in these positions would enable "syntactic interference" between data sources. Thus, in our ruleset, we always enforce blank-nodes as mandated by the OWL abstract syntax; this specifically applies to pD* rules **rdfp14***, **rdfp15** and **rdfp16** and to all of our C-entailment rules **rdfc***. We denote the restrictions in the **where** column of Table 2. Indeed, in our treatment of terminological collection statements, we enforced blank nodes in the subject position of `rdf:first`/`rdf:rest` membership assertions, as well as blank nodes in the object position of non-terminating `rdf:rest` statements; these are analogously part of the OWL abstract syntax restrictions.

3.3 Separation of T-Box from A-Box

Aside from the differences already introduced, our primary divergence from the pD* fragment and traditional rule-based approaches is that we separate terminological data from assertional data according to their use of the RDF(S) and OWL vocabulary; these are commonly known as the "T-Box" and "A-Box" respectively (loosely borrowing Description Logics terminology). In particular, we require a separation of T-Box data as part of a core optimisation of our approach; we wish to perform a once-off load of T-Box data from our input knowledge-base into main memory.

Let $\mathcal{P}_{SAOR}$ and $\mathcal{C}_{SAOR}$ be, resp., the exact set of RDF(S)/OWL meta-properties and -classes used in our inference rules; viz. $\mathcal{P}_{SAOR}$ = { `rdfs:domain, rdfs:range, rdfs:subClassOf, rdfs:subPropertyOf, :allValuesFrom, :cardinality, :equivalentClass, :equivalentProperty, :hasValue, :intersectionOf, :inverseOf, :maxCardinality, :minCardinality, :oneOf, :onProperty, :sameAs, :someValuesFrom, :unionOf` } & $\mathcal{C}_{SAOR}$ = { `:FunctionalProperty, :InverseFunctionalProperty, :SymmetricProperty, :TransitiveProperty` }; our T-Box is a set of terminological triples restricted to only include membership assertions for $\mathcal{P}_{SAOR}$ and $\mathcal{C}_{SAOR}$ and the set of terminological collection statements. Table 2 identifies T-Box patterns by underlining.

[6] In [43], rules using RDF collection constructs were not included (such as our rules **rdfc0**,**rdfc1**,**rdfc3***) as they have variable antecedent-body length and, thus, can affect complexity considerations. It was informally stated that `:intersectionOf` and `:unionOf` could be supported under pD* through reduction into subclass relations; however no rules were explicitly defined and our rule **rdfc3b** could not be supported in this fashion. We support such rules here since we are not so concerned for the moment with theoretical worst-case complexity, but are more concerned with the practicalities of web reasoning.

Statements from the input knowledge-base that match these patterns are all of the T-Box statements we consider in our reasoning process: inferred statements or statements that do not match one of these patterns are not considered being part of the T-Box, but are treated purely as assertional. We now define our T-Box:

Definition 1 (T-Box) *Let $\mathcal{T}_\mathcal{G}$ be the union of all graph pattern instances from a graph $\mathcal{G}$ for a terminological (underlined) graph pattern in Table 2; i.e., $\mathcal{T}_\mathcal{G}$ is itself a graph. We call $\mathcal{T}_\mathcal{G}$ the* T-Box of $\mathcal{G}$.

Also, let $\mathcal{P}^{domP}_{SAOR}$ = { `rdfs:domain, rdfs:range, rdfs:subPropertyOf, :equivalentProperty, :inverseOf` } and $\mathcal{P}^{ranP}_{SAOR}$ = { `rdfs:subPropertyOf, :equivalentProperty, :inverseOf, :onProperty` }, We call ϕ a *property in T-Box* $\mathcal{T}$ if there exists a triple $t \in \mathcal{T}$ where

- $s = \phi$ and $p \in \mathcal{P}^{domP}_{SAOR}$
- $p \in \mathcal{P}^{ranP}_{SAOR}$ and $o = \phi$
- $s = \phi$, p=`rdf:type` and $o \in \mathcal{C}_{SAOR}$

Similarly, let $\mathcal{P}^{domC}_{SAOR}$ = { `rdfs:subClassOf, :allValuesFrom, :cardinality, :equivalentClass, :hasValue, :intersectionOf, :maxCardinality, :minCardinality, :oneOf, :onProperty, :someValuesFrom, :unionOf` }, $\mathcal{P}^{ranC}_{SAOR}$ = { `rdf:first, rdfs:domain, rdfs:range, rdfs:subClassOf, :allValuesFrom, :equivalentClass, :someValuesFrom` }. We call χ a *class in T-Box* $\mathcal{T}$ if there exists a triple $t \in \mathcal{T}$ where

- $p \in \mathcal{P}^{domC}_{SAOR}$ and $s = \chi$
- $p \in \mathcal{P}^{ranC}_{SAOR}$ and $o = \chi$

We define the signature of a T-Box $\mathcal{T}$ to be the set of all properties and classes in $\mathcal{T}$ as above, which we denote by $sig(\mathcal{T})$.

For our knowledge-base $\mathbb{KB}$, we define our *T-Box* $\mathbb{T}$ as the set of all pairs $(\mathcal{T}_{\mathcal{W}'}, c)$ where $(\mathcal{W}', c) \in \mathbb{KB}$ and $\mathcal{T}_{\mathcal{W}'} \neq \emptyset$. Again, we may use the intuitive notation $\mathcal{T}_{\mathcal{W}'} \in \mathbb{T}$. We define our A-Box $\mathbb{A}$ as containing all of the statements in $\mathbb{KB}$, including $\mathbb{T}$ and the set of class and property membership assertions possibly using identifiers in $\mathcal{P}_{SAOR} \cup \mathcal{C}_{SAOR}$; i.e., unlike description logics, our $\mathbb{A}$ is synonymous with our $\mathbb{KB}$. We use the term A-Box to distinguish data that are stored on-disk (which includes T-Box data also stored in memory).

We now define our notion of a *$\mathcal{T}$-split inference rule*, whereby part of the antecedent is a basic graph pattern strictly instantiated by a static T-Box $\mathcal{T}$.

Definition 2 ($\mathcal{T}$-split inference rule) *We define a* $\mathcal{T}$-split inference rule *as a triple $(Ante_\mathcal{T}, Ante_\mathcal{G}, Con)$, where $Ante_\mathcal{T}$ is a basic graph pattern matched by a static T-Box $\mathcal{T}$ and $Ante_\mathcal{G}$ is matched by data in the graph $\mathcal{G}$, Con does not contain blank nodes, $\mathcal{V}(Con) \neq \emptyset$, $\mathcal{V}(Con) \subseteq \mathcal{V}(Ante_\mathcal{T}) \cup \mathcal{V}(Ante_\mathcal{G})$; also, if both $Ante_\mathcal{T}$ and $Ante_\mathcal{G}$ are non-empty, then $\mathcal{V}(Ante_\mathcal{T}) \cap \mathcal{V}(Ante_\mathcal{G}) \neq \emptyset$*

We generally write $(Ante_\mathcal{T}, Ante_\mathcal{G}, Con)$ as $\underline{Ante_\mathcal{T}} Ante_\mathcal{G} \Rightarrow Con$ where Table 2 follows this convention. We call $Ante_\mathcal{T}$ the terminological or T-Box antecedent pattern and $Ante_\mathcal{G}$ the assertional or A-Box antecedent pattern.

We now define three disjoint sets of $\mathcal{T}$-split rules which consist of only a T-Box graph pattern, both a T-Box and A-Box graph pattern and only an A-Box graph pattern:

Definition 3 (Rule-sets $\mathcal{R}_\mathcal{T}, \mathcal{R}_{\mathcal{TG}}, \mathcal{R}_\mathcal{G}$) *We define $\mathcal{R}_\mathcal{T}$ as the set of $\mathcal{T}$-split rules for which $Ante_\mathcal{T} \neq \emptyset$ and $Ante_\mathcal{G} = \emptyset$. We define $\mathcal{R}_{\mathcal{TG}}$ as the set of $\mathcal{T}$-split rules for which $Ante_\mathcal{T} \neq \emptyset$ and $Ante_\mathcal{G} \neq \emptyset$. We define $\mathcal{R}_\mathcal{G}$ as the set of $\mathcal{T}$-split rules for which $Ante_\mathcal{T} = \emptyset$ and $Ante_\mathcal{G} \neq \emptyset$.*

In Table 2, we categorise the $\mathcal{T}$-split rules into four rulesets: $\mathcal{R}0 \subset \mathcal{R}_\mathcal{T}$; $\mathcal{R}1 \subset \mathcal{R}_{\mathcal{TG}}$ where $|Ante_\mathcal{G}| = 1$; $\mathcal{R}2 \subset \mathcal{R}_{\mathcal{TG}}$ where $|Ante_\mathcal{G}| > 1$ and $\mathcal{R}0 \subset \mathcal{R}_\mathcal{G}$. We now introduce the notion of a $\mathcal{T}$-split inference rule application for a graph $\mathcal{G}$ w.r.t. a T-Box $\mathcal{T}$:

Definition 4 ($\mathcal{T}$-split inference rule application) *We define a* $\mathcal{T}$-split rule application *to be $T_r(\mathcal{T}, \mathcal{G})$ for $r = (Ante_\mathcal{T}, Ante_\mathcal{G}, Con)$ as follows:*

$$T_r(\mathcal{T}, \mathcal{G}) = \{\mu(Con) \mid \exists \mu \text{ such that } \mu(Ante_\mathcal{T}) \subseteq \mathcal{T} \text{ and } \mu(Ante_\mathcal{G}) \subseteq \mathcal{G}\}$$

Again, $T_{\mathcal{R}}(\mathcal{T},\mathcal{G}) = \bigcup_{r\in\mathcal{R}} T_r(\mathcal{T},\mathcal{G})$; also, given $\mathcal{T}$ as static, the exhaustive application of the $T_{\mathcal{R}}(\mathcal{T},\mathcal{G})$ up to the least fixpoint is called the *$\mathcal{T}$-split closure* of $\mathcal{G}$, denoted as $Cl_{\mathcal{R}}(\mathcal{T},\mathcal{G})$. Again we use abbreviations such as $T_R(\mathbb{T},\mathbb{KB})$ and $Cl_{\mathcal{R}}(\mathbb{T},\mathbb{KB})$, where $\mathbb{KB}$ should be interpreted as $\bigcup_{\mathcal{W}'\in\mathbb{KB}}\mathcal{W}'$ and $\mathbb{T}$ as $\bigcup_{\mathcal{T}_{\mathcal{W}'}\in\mathbb{T}}\mathcal{T}_{\mathcal{W}'}$.

Please note that since we enforce blank nodes in all positions mandated by the OWL abstract syntax for our rules, each instance of a given graph pattern $Ante_{\mathcal{T}}$ can only contain triples from one web graph $\mathcal{W}'$ where $\mathcal{T}_{\mathcal{W}'} \in \mathbb{T}$. Let $\mathcal{V}_{\mathcal{B}}(\mathcal{GP})$ be the set of all variables in a graph pattern $\mathcal{GP}$ which we restrict to only be instantiated by a blank node according to the abstract syntax. For all $Ante_{\mathcal{T}}$ in our rules where $|Ante_{\mathcal{T}}| > 1$ let $Ante^{-}_{\mathcal{T}}$ be any proper non-empty subset of $Ante_{\mathcal{T}}$; we can then say that $\mathcal{V}_{\mathcal{B}}(Ante^{-}_{\mathcal{T}}) \cap \mathcal{V}_{\mathcal{B}}(Ante_{\mathcal{T}} \setminus Ante^{-}_{\mathcal{T}}) \neq \emptyset$. In other words, since for every rule either (i) $Ante_{\mathcal{T}} = \emptyset$; or (ii) $Ante_{\mathcal{T}}$ consists of a single triple pattern; or otherwise (iii) no sub-pattern of $Ante_{\mathcal{T}}$ contains a unique set of blank-node enforced variables; then a given instance of $Ante_{\mathcal{T}}$ can only contain triples from one web-graph with unique blank nodes as is enforced by our knowledge-base. For our ruleset, we can then say that $T_R(\mathbb{T},\mathbb{KB}) = T_R(\bigcup_{\mathcal{T}_{\mathcal{W}'}\in\mathbb{T}}\mathcal{T}_{\mathcal{W}'},\mathbb{KB}) = \bigcup_{\mathcal{T}_{\mathcal{W}'}\in\mathbb{T}} T_R(\mathcal{T}_{\mathcal{W}'},\mathbb{KB})$. In other words, one web-graph cannot re-use structural statements in another web-graph to instantiate a T-Box pattern in our rule; this has bearing on our notion of authoritative reasoning which will be highlighted at the end of Section 3.4.

Further, a separate static T-Box within which inferences are not reflected has implications upon the completeness of reasoning w.r.t. the presented ruleset. Although, as presented in Section 3.2, we do not infer terminological statements and thus can support most inferences directly from our static T-Box, SAOR still does not fully support meta-modelling [33]: by separating the T-Box segment of the knowledge-base, we do not support all possible entailments from the simultaneous description of both a class (or property) and an indvidual. In other words, we do not fully support inferencing for meta-classes or meta-properties defined outside of the RDF(S)/OWL specification.

However, we do provide limited reasoning support for meta-modelling in the spirit of "punning" by conceptually separating the individual-, class- or property-meanings of a resource (c.f. [14]). More precisely, during reasoning we not only store the T-Box data in memory, but also store the data on-disk in the A-Box. Thus, we perform punning in one direction: viewing class and property descriptions which form our T-Box also as individuals. Interestingly, although we do not support terminological reasoning directly, we can through our limited punning perform reasoning for terminological data based on the RDFS descriptions provided for the RDFS and OWL specifications. For example, we would infer the following by storing the three input statements in both the T-Box and the A-Box:

```
rdfs:subClassOf rdfs:domain rdfs:Class; rdfs:range rdfs:Class .
ex:Class1 rdfs:subClassOf ex:Class2 .  ⇒
ex:Class1 a rdfs:Class .  ex:Class2 a rdfs:Class .
```

However, again our support for meta-modelling is limited; SAOR does not fully support so-called "non-standard usage" of RDF(S) and OWL: the use of properties and classes which make up the RDF(S) and OWL vocabularies in locations where they have not been intended, cf. [6, 34]. We adapt and refine the definition of non-standard vocabulary use for our purposes according to the parts of the RDF(S) and OWL vocabularies relevant for our inference ruleset:

Definition 5 (Non-Standard Vocabulary Usage) *An RDF triple t has* non-standard vocabulary usage *if one of the following conditions holds:*

1. *a property in $\mathcal{P}_{SAOR}$ appears in a position different from the predicate position.*
2. *a class in $\mathcal{C}_{SAOR}$ appears in a position different from the object position of an `rdf:type` triple.*

Continuing, we now introduce the following example wherein the first input statement is a case of non-standard usage with `rdfs:subClassOf`$\in \mathcal{P}_{SAOR}$ in the object position:[7]

```
ex:subClassOf rdfs:subPropertyOf rdfs:subClassOf .
ex:Class1 ex:subClassOf ex:Class2 .  ⇒
ex:Class1 rdfs:subClassOf ex:Class2 .
```

[7] A similar example from the Web can be found at `http://thesauri.cs.vu.nl/wordnet/rdfs/wordnet2b.owl`.

We can see that SAOR provides inference through `rdfs:subPropertyOf` as per usual; however, the inferred triple will not be reflected in the T-Box, thus we are incomplete and will not translate members of `ex:Class1` into `ex:Class2`. As such, non-standard usage may result in T-Box statements being produced which, according to our limited form of punning, will not be reflected in the T-Box and will lead to incomplete inference.

Indeed, there may be good reason for not fully supporting non-standard usage of the ontology vocabulary: non-standard use could have unpredictable results even under our simple rule-based entailment if we were to fully support meta-modelling. One may consider a finite combination of only four non-standard triples that, upon naive reasoning, would explode all web resources R by inferring $|R|^3$ triples, namely:

```
rdfs:subClassOf rdfs:subPropertyOf rdfs:Resource.
rdfs:subClassOf rdfs:subPropertyOf rdfs:subPropertyOf.
rdf:type rdfs:subPropertyOf rdfs:subClassOf.
rdfs:subClassOf rdf:type :SymmetricProperty.
```

The exhaustive application of standard RDFS inference rules plus inference rules for property symmetry together with the inference for class membership in `rdfs:Resource` for all collected resources in typical rulesets such as pD* lead to inference of any possible triple $(r_1\ r_2\ r_3)$ for arbitrary $r_1, r_2, r_3 \in R$.

Thus, although by maintaining a separate static T-Box we are incomplete w.r.t non-standard usage, we show that complete support of such usage of the RDFS/OWL vocabularies is undesirable for the Web.[8]

3.4 Authoritative Reasoning against Ontology Hijacking

During initial evaluation of a system which implements reasoning upon the above ruleset, we encountered a behaviour which we term "ontology hijacking", symptomised by a perplexing explosion of materialised statements. For example, we noticed that for a single `foaf:Person` membership assertion, SAOR inferred in the order of hundreds of materialised statements as opposed to the expected six. Such an explosion of statements is orthogonal to the aim of reduced materialised statements we have outlined for SAOR; thus, SAOR is designed to annul the diagnosed problem of ontology hijacking through anaylsis of the authority of web sources for T-Box data. Before formally defining ontology hijacking and our proposed solution, let us give some preliminary definitions:

Definition 6 (Authoritative Source) *A web-graph $\mathcal{W}$ from source (context) c speaks authoritatively about an RDF term n iff:*

1. $n \in \mathcal{B}$; *or*
2. $n \in \mathcal{U}$ *and c coincides with, or is redirected to by, the namespace*[9] *of n.*

Firstly, all graphs are authoritative for blank nodes defined in that graph (remember that according to the definition of our knowledge-base, all blank nodes are unique to a given graph). Secondly, we support namespace redirects so as to conform to best practices as currently adopted by web ontology publishers.[10]

For example, as taken from the Web:

- Source `http://usefulinc.com/ns/doap` is authoritative for all classes and properties which are within the `http://usefulinc.com/ns/doap` namespace; e.g., `http://usefulinc.com/ns/doap#Project`.
- Source `http://xmlns.com/foaf/spec/` is authoritative for all classes and properties which are within the `http://xmlns.com/foaf/0.1/` namespace; e.g., `http://xmlns.com/foaf/0.1/knows`; since the property `http://xmlns.com/foaf/0.1/knows` redirects to `http://xmlns.com/foaf/spec/`.

[8] In any case, as we will see in Section 3.4, our application of authoritative analysis would not allow such arbitrary third-party re-definition of core RDF(S)/OWL constructs.

[9] Here, slightly abusing XML terminology, by "namespace" of a URI we mean the prefix of the URI obtained from stripping off the final NCname.

[10] See Appendix A&B of `http://www.w3.org/TR/swbp-vocab-pub/`

We consider the authority of sources speaking about classes and properties in our T-Box to counteract ontology hijacking; ontology hijacking is the assertion of a set of non-authoritative T-Box statements such that could satisfy the T-Box antecedent pattern of a rule in $\mathcal{R}_{TG}$ (i.e., those rules with at least one terminological and at least one assertional triple pattern in the antecedent). Such third-party sources can then cause arbitrary inferences on membership assertions of classes or properties (contained in the A-Box) for which they speak non-authoritatively. We can say that only rules in $\mathcal{R}_{TG}$ are relevant to ontology hijacking since: (i) inferencing on $\mathcal{R}_G$, which does not contain any T-Box patterns, cannot be affected by non-authoritative T-Box statements; and (ii) the $\mathcal{R}_T$ ruleset does not contain any A-Box antecedent patterns and therefore, cannot directly hijack assertional data (i.e., in our scenario, the `:oneOf` construct can be viewed as directly asserting memberships, and is unable, according to our limited support, to directly redefine sets of individuals). We now define ontology hijacking:

Definition 7 (Ontology Hijacking) *Let $\mathcal{T}_\mathcal{W}$ be the T-Box extracted from a web-graph $\mathcal{W}$ and let $\widehat{sig}(\mathcal{W})$ be the set of classes and properties for which $\mathcal{W}$ speaks authoritatively; then if $Cl_{\mathcal{R}_{TG}}(\mathcal{T}_\mathcal{W}, \mathcal{G}) \neq \mathcal{G}$ for any $\mathcal{G}$ not mentioning any element of $\widehat{sig}(\mathcal{W})$, we say that web-graph $\mathcal{W}$ is performing* ontology hijacking.

In other words, ontology hijacking is the contribution of statements about classes and/or properties in a non-authoritative source such that reasoning on those classes and/or properties is affected. One particular method of ontology hijacking is defining new super-classes or properties of third-party classes or properties.

As a concrete example, if one were to publish today a description of a property in an ontology (in a location non-authoritative for `foaf:` but authoritative for `my:`), `my:name`, within which the following was stated: `foaf:name rdfs:subPropertyOf my:name .`, that person would be hijacking the `foaf:name` property and effecting the translation of all `foaf:name` statements in the web knowledge-base into `my:name` statements as well.

However, if the statement were instead `my:name rdfs:subPropertyOf foaf:name .`, this would not constitute a case of ontology hijacking but would be a valid example of translating from a local authoritative property into an external non-authoritative property.

Ontology hijacking is problematic in that it vastly increases the amount of statements that are materialised and can potentially harm inferencing on data contributed by other parties. With respect to materialisation, the former issue becomes prominent: members of classes/properties from popular/core ontologies get translated into a plethora of conceptual models described in obscure ontologies; we quantify the problem in Section 5. However, taking precautions against harmful ontology hijacking is growing more and more important as the Semantic Web features more and more attention; motivation for spamming and other malicious activity propagates amongst certain parties with ontology hijacking being a prospective avenue. With this in mind, we assign sole responsibility for classes and properties and reasoning upon their members to those who maintain the authoritative specification.

Related to the idea of ontology hijacking is the idea of "non-conservative extension" described in the Description Logics literature: cf. [13, 31, 27]. However, the notion of a "conservative extension" was defined with a slightly different objective in mind: according to the notion of deductively conservative extensions, a graph G_a is only considered malicious towards G_b if it causes additional inferences with respect to the intersection of the signature of the original G_b with the newly inferred statements. Returning to the former `my:name` example from above, defining a super-property of `foaf:name` would still constitute a conservative extension: the closure without the non-authoritative `foaf:name rdfs:subPropertyOf my:name .` statement is the same as the closure with the statement after all of the `my:name` membership assertions are removed. However, further stating that `my:name a :InverseFunctionalProperty.` would not satisfy a model conservative extension since members of `my:name` might then cause equalities in other remote ontologies as side-effects, independent from the newly defined signature. Summarising, we can state that every non-conservative extension (with respect to our notion of deductive closure) constitutes a case of ontology hijacking, but not vice versa; non-conservative extension can be considered "harmful" hijacking whereas the remainder of ontology hijacking cases can be considered "inflationary".

To negate ontology hijacking, we only allow inferences through *authoritative rule applications*, which we now define:

Definition 8 (Authoritative Rule Application) *Again let $\widehat{sig}(\mathcal{W})$ be the set of classes and properties for which $\mathcal{W}$ speaks authoritatively and let $\mathcal{T}_\mathcal{W}$ be the T-Box of $\mathcal{W}$. We define an* authoritative rule application *for a graph $\mathcal{G}$ w.r.t. the T-Box $\mathcal{T}_\mathcal{W}$ to be a $\mathcal{T}$-split rule application $T_r(\mathcal{T}_\mathcal{W}, \mathcal{G})$ where additionally, if both $Ante_\mathcal{T}$ and $Ante_\mathcal{G}$ are non-empty ($r \in \mathcal{R}_{\mathcal{TG}}$), then for the mapping μ of $T_r(\mathcal{T}_\mathcal{W}, \mathcal{G})$ there must exist a variable $v \in (\mathcal{V}(Ante_\mathcal{T}) \cap \mathcal{V}(Ante_\mathcal{G}))$ such that $\mu(v) \in \widehat{sig}(\mathcal{W})$. We denote an* authoritative rule application *by $T_{\widehat{r}}(\mathcal{T}_\mathcal{W}, \mathcal{G})$.*

In other words, an authoritative rule application will only occur if the rule consists of only assertional patterns ($\mathcal{R}_\mathcal{G}$); or the rules consists of only terminological patterns ($\mathcal{R}_\mathcal{T}$); or if in application of the rule, the terminological pattern instance is from a web-graph authoritative for at least one class or property in the assertional pattern instance. The $T_{\widehat{\mathcal{R}}}$ operator follows naturally as before for a set of authoritative rules $\widehat{\mathcal{R}}$, as does the notion of authoritative closure which we denote by $Cl_{\widehat{\mathcal{R}}}(\mathcal{T}_\mathcal{W}, \mathcal{G})$. We may also refer to, e.g., $T_{\widehat{\mathcal{R}}}(\mathbb{T}, \mathbb{KB})$ and $Cl_{\widehat{\mathcal{R}}}(\mathbb{T}, \mathbb{KB})$ as before for a $\mathcal{T}$-split rule application.

Table 2 identifies the authoritative restrictions we place on our rules wherein the underlined T-Box pattern is matched by a set of triples from a web-graph $\mathcal{W}$ iff $\mathcal{W}$ speaks authoritatively for at least one element matching a boldface variable in Table 2; i.e., again, for each rule, at least one of the classes or properties matched by the A-Box pattern of the antecedent must be authoritatively spoken for by an instance of the T-Box pattern. These restrictions only apply to $\mathcal{R}1$ and $\mathcal{R}2$ (which are both a subset of $\mathcal{R}_{\mathcal{TG}}$). Please note that, for example in rule **rdfp14b′** where there are no boldface variables, the variables enforced to be instantied by blank nodes will always be authoritatively spoken for: a web-graph is always authoritative for its blank nodes.

We now make the following proposition relating to the prevention of ontology-hijacking through authoritative rule application:

Proposition 1 *Given a T-Box $\mathcal{T}_\mathcal{W}$ extracted from a web-graph $\mathcal{W}$ and any graph $\mathcal{G}$ not mentioning any element of $\widehat{sig}(\mathcal{W})$, then $Cl_{\widehat{\mathcal{R}_{\mathcal{TG}}}}(\mathcal{T}_\mathcal{W}, \mathcal{G}) = \mathcal{G}$.*

Proof: Informally, our proposition is that the authoritative closure of a graph $\mathcal{G}$ w.r.t. some T-Box $\mathcal{T}_\mathcal{W}$ will not contain any inferences which constitute ontology hijacking, defined in terms of ruleset $\mathcal{R}_{\mathcal{TG}}$. Firstly, from Definition 3, for each rule $r \in \mathcal{R}_{\mathcal{TG}}$, $Ante_\mathcal{T} \neq \emptyset$ and $Ante_\mathcal{G} \neq \emptyset$. Therefore, from Definitions 4 & 8, for an authoritative rule application to occur for any such r, there must exist (i) a mapping μ such that $\mu(Ante_\mathcal{T},) \subseteq \mathcal{T}_\mathcal{W}$ and $\mu(Ante_\mathcal{G}) \subseteq \mathcal{G}$; and (ii) a variable $v \in (\mathcal{V}(Ante_\mathcal{T}) \cap \mathcal{V}(Ante_\mathcal{G}))$ such that $\mu(v) \in \widehat{sig}(\mathcal{W})$. However, since $\mathcal{G}$ does not mention any element of $\widehat{sig}(\mathcal{W})$, then there is no such mapping μ where $\mu(v) \in \widehat{sig}(\mathcal{W})$ for $v \in \mathcal{V}(Ante_\mathcal{G})$, and $\mu(Ante_\mathcal{G}) \subseteq \mathcal{G}$. Hence, for $r \in \mathcal{R}_{\mathcal{TG}}$, no such application $T_{\widehat{r}}(\mathcal{T}_\mathcal{W}, \mathcal{G})$ will occur; it then follows that $T_{\widehat{\mathcal{R}_{\mathcal{TG}}}}(\mathcal{T}_\mathcal{W}, \mathcal{G}) = \emptyset$ and $Cl_{\widehat{\mathcal{R}_{\mathcal{TG}}}}(\mathcal{T}_\mathcal{W}, \mathcal{G}) = \mathcal{G}$. □

The above proposition and proof holds for a given web-graph $\mathcal{W}$; however, given a set of web-graphs where an instance of $Ante_\mathcal{T}$ can consist of triples from more that one graph, it is possible for ontology hijacking to occur whereby some triples in the instance come from a non-authoritative graph and some from an authoritative graph. To illustrate we refer to Example 3.1, wherein (and without enforcing abstract syntax blank nodes) the second source could cause ontology hijacking by interfering with the authoritative definition of the class restriction in the first source as follows:

Example 3.2 :

```
# RULE (adapted so that ?C need not be a blank node)
?C :allValuesFrom ?D ; :onProperty ?P . ?x a ?C ; ?P ?y .  ⇒ ?y a ?D .
# FROM SOURCE <ex:>
ex:Person :onProperty ex:parent .
# FROM SOURCE <ex2:>
ex:Person :allValuesFrom ex2:Human .
# ASSERTIONAL
_:Jim a ex:Person ; ex:parent _:Jill .
⇒
_:Jill a ex2:Human .
```
◇

Here, the above inference is authoritative according to our definition since the instance of $Ante_{\mathcal{T}}$ (specifically the first statement from `source <ex:>`) speaks authoritatively for a class/property in the assertional data; however, the statement from `source <ex2:>` is causing inferences on assertional data not containing a class or property for which `source <ex2:>` is authoritative .

As previously discussed, for our ruleset, we enforce the OWL abstract syntax and thus we enforce that $\mu(Ante_{\mathcal{T}}) \subseteq \mathcal{T}_{\mathcal{W}'}$ where $\mathcal{T}_{\mathcal{W}'} \in \mathbb{T}$. However, where this condition does not hold (i.e., an instance of $Ante_{\mathcal{T}}$ can comprise of data from more than one graph), then an authoritative rule application should only occur if each web-graph contributing to an instance of $Ante_{\mathcal{T}}$ speaks authoritatively for at least one class/property in the $Ante_{\mathcal{G}}$ instance.

4 Reasoning Algorithm

In the following we first present observations on web data that influenced the design of the SAOR algorithm, then give an overview of the algorithm, and next discuss details of how we handle T-Box information, perform statement-wise reasoning, and deal with equality for individuals.

4.1 Characteristics of Web Data

Our algorithm is intended to operate over a web knowledge-base as retrieved by means of a web crawl; therefore, the design of our algorithm is motivated by observations on our web dataset:

1. Reasoning accesses a large slice of data in the index: we found that approximately 61% of statements in the 147m dataset and 90% in the 1.1b dataset produced inferred statements through authoritative reasoning.

2. Relative to assertional data, the volume of terminological data on the Web is small: <0.9% of the statements in the 1.1b dataset and <1.7% of statements in the 147m dataset were classifiable as SAOR T-Box statements[11].

3. The T-Box is the most frequently accessed segment of the knowledge-base for reasoning: although relatively small, all but the rules in $\mathcal{R}3$ require access to T-Box information.

Following from the first observation, we employ a file-scan batch-processing approach so as to enable sequential access over the data and avoid disk-lookups and dynamic data structures which would not perform well given high disk latency; also we avoid probing the same statements repeatedly for different rules at the low cost of scanning a given percentage of statements not useful for reasoning.

Following from the second and third observations, we optimise by placing T-Box data in a separate data structure accessible by the reasoning engine.

Currently, we hold all T-Box data in-memory, but the algorithm can be generalised to provide for a caching on-disk structure or a distributed in-memory structure as needs require.[12]

To be able to scale, we try to minimise the amount of main memory needed, given that main memory is relatively expensive and that disk-based algorithms are thus more economical [29]. Given high disk latency, we avoid using random-access on-disk data structures. In our previous work, a disk-based updateable random-access data structure (a B+-Tree) proved to be the bottleneck for reasoning due to a high volume of inserts, leading to frequent index reorganisations and hence inadequate performance. As a result, our algorithms are now build upon two disk-based primitives known to scale: file scanning and sorting.

4.2 Algorithm Overview

The SAOR algorithm performs a fixpoint computation by iteratively applying the rules in Table 2. Figure 1 outlines the architecture. The reasoning process can be roughly divided into the following steps:

[11] Includes some RDF collection fragments which may not be part of a class description

[12] We expect that a caching on-disk index would work well considering the distribution of membership assertions for classes and properties in web data; there would be a high hit-rate for the cache.

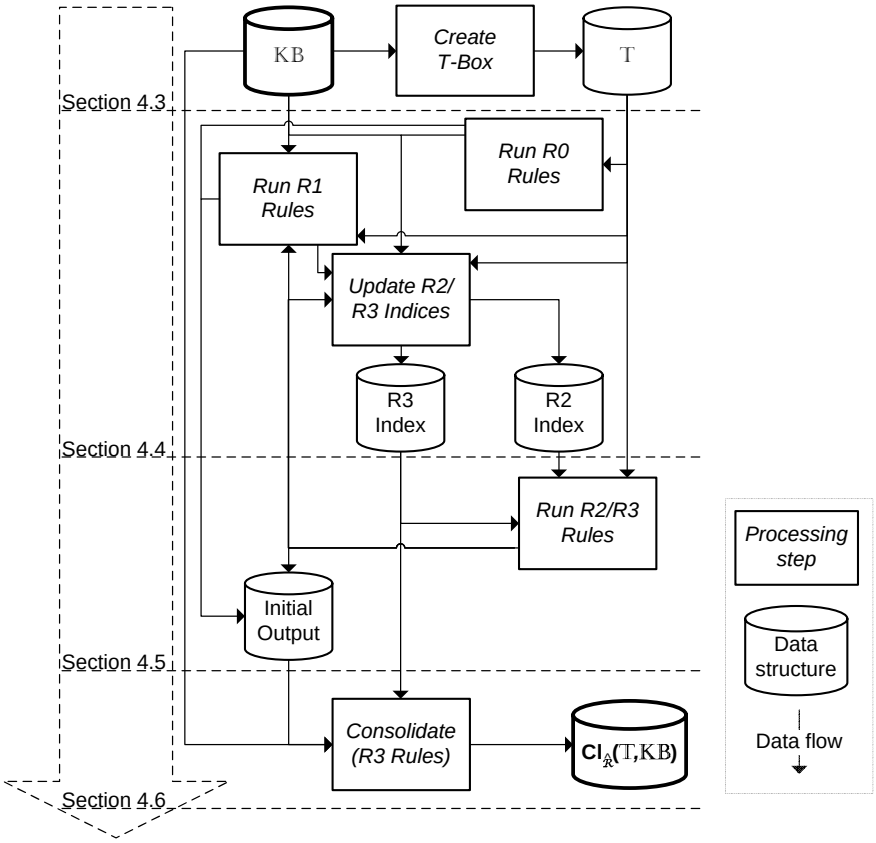

Figure 1: High-level architecture

1. Separate $\mathbb{T}$ from $\mathbb{KB}$, build in-memory representation $\mathbb{T}$, and apply ruleset $\mathcal{R}0$ (Section 4.3).
2. Perform reasoning over $\mathbb{KB}$ in a statement-wise manner (Section 4.4):
   - Execute rules with only a single A-Box triple pattern in the antecedent ($\mathcal{R}1$): join A-Box pattern with in-memory T-Box; recursively execute steps over inferred statements; write inferred RDF statements to output file.
   - Write on-disk files for computation of rules with multiple A-Box triple patterns in the antecedent ($\mathcal{R}2$); when a statement matches one of the A-Box triple patterns for these rules and the necessary T-Box join exists, the statement is written to the on-disk file for later rule computation.
   - Write on-disk equality file for rules which involve equality reasoning ($\mathcal{R}3$); `:sameAs` statements found during the scan are written to an on-disk file for later computation.
3. Execute ruleset $\mathcal{R}2 \cup \mathcal{R}3$: on-disk files containing partial A-Box antecedent matches for rules in $\mathcal{R}2$ and $\mathcal{R}3$ are sequentially analysed producing further inferred statements. Newly inferred statements are again subject to step 2 above; fresh statements can still be written to on-disk files and so the process is iterative until no new statements are found (Section 4.5).
4. Finally, consolidate source data along with inferred statements according to `:sameAs` computation ($\mathcal{R}3$) and write to final output (Section 4.6).

In the following sections, we discuss the individual components and processes in the architecture as highlighted, whereafter, in Section 4.7 we show how these elements are combined to achieve closure.

4.3 Handling Terminological Data

In the following, we describe how to separate the T-Box data and how to create the data structures for representing the T-Box.

T-Box data from RDFS and OWL specifications can be acquired either from conventional crawling techniques, or by accessing the locations pointed to by the dereferenced URIs of classes and properties in the data. We assume for brevity that all of the pertinent terminological data have already been collected and exist in the input data. If T-Box data are sourced separately via different means we can build an in-memory representation directly, without requiring the first scan of all input data.

We apply the following algorithm to create the T-Box in-memory representation, which we will analyse in the following sections:

1. FULL SCAN 1: separate T-Box information as described in Definition 1.
2. TBOX SCAN 1 & 2: reduce irrelevant RDF collection statements.
3. TBOX SCAN 3: perform authoritative analysis of the T-Box data and load in-memory representation.

4.3.1 Separating and Reducing T-Box Data

Firstly, we wish to separate all possible T-Box statements from the main bulk of data. $\mathcal{P}_{SAOR}$ and $\mathcal{C}_{SAOR}$ are stored in memory and then the data dump is scanned. Quadruples with property $\in \mathcal{P}_{SAOR} \cup \{$`rdf:first`, `rdf:rest` $\}$ or `rdf:type` statements with object $\in \mathcal{C}_{SAOR}$ (which, where applicable, abide by the OWL abstract syntax) are buffered to a T-Box data file.

However, the T-Box data file still contains a large amount of RDF collection statements (property $\in$ { `rdf:first, rdf:rest` }) which are not related to reasoning. SAOR is only interested in such statements wherein they form part of a `:unionOf`, `:intersectionOf` or `:oneOf` class description. Later when the T-Box is being loaded, these collection fragments are reconstructed in-memory and irrelevant collection fragments are discarded; to reduce the amount of memory required we can quickly discard irrelevant collection statements through two T-Box scans:

- scan the T-Box data and store contexts of statements where the property $\in$ { `:unionOf`, `:intersectionOf`, `:oneOf` }.
- scan the T-Box data again and remove statements for which both hold:
  - property $\in$ { `rdf:first, rdf:rest` }
  - the context does not appear in those stored from the previous scan.

These scans quickly remove irrelevant collection fragments where a `:unionOf`, `:intersectionOf`, `:oneOf` statement does not appear in the same source as the fragment (i.e., collections which cannot contribute to the T-Box pattern of one of our rules).

4.3.2 Authoritative Analysis

We next apply authoritative analysis to the T-Box and load the results into our in-memory representation; in other words, we build an *authoritative T-Box* which pre-computes authority of T-Box data. We denote our authoritative T-Box by $\widehat{\mathbb{T}}$, whereby $Cl_{\widehat{\mathcal{R}}}(\mathbb{T}, \mathbb{KB}) = Cl_{\mathcal{R}}(\widehat{\mathbb{T}}, \mathbb{KB})$; for each rule, $\widehat{\mathbb{T}}$ only contains T-Box pattern instances for $Ante_{\mathcal{T}}$ which can lead to an authoritative rule application.

Each statement read is initially matched without authoritative analysis against the patterns enumerated in Table 3. If a pattern is initially matched, the positions required to be authoritative, as identified in boldface, are checked. If one such authoritative check is satified, the pattern is loaded into the T-Box. Indeed the same statement may be matched by more than one T-Box pattern for different rules with different authoritative restrictions; for example the statement `foaf:name :equivalentProperty my:name .` retrieved from `my:` namespace matches the T-Box pattern of rules **rdfp13a′** & **rdfp13b′**, but only conforms to the authoritative restriction for rule **rdfp13b′**. Therefore, we only store the statement in such a fashion as to

apply to rule **rdfp13b′**; that is, the authoritative T-Box stores T-Box pattern instances separately for each rule, according to the authoritative restrictions for that rule.

Checking the authority of a source for a given namespace URI, as presented in Definition 6, may require a HTTP connection to the namespace URI so as to determine whether a redirect exists to the authoritative document (HTTP Response Code 303). Results of accessing URIs are cached once in-memory so as to avoid establishing repetitive connections. If the pattern is authoritatively matched, the statement is reflected in the in-memory T-Box. Alternatively, where available, a crawler can provide a set of redirect pairs which can be loaded into the system to avoid duplicating HTTP lookups; we presume for generality that such information is not provided.

	$\mathcal{R}0$	
rdfc0	?C :oneOf (?x$_1$ ... ?x$_n$) .	(?x$_1$ ... ?x$_n$) $\overset{\textbf{rdfc0}}{\rightarrow}$?C
	$\mathcal{R}1$	
rdfs2	**?P** rdfs:domain ?C .	**?P** $\overset{\textbf{rdfs2}}{\rightarrow}$?C
rdfs3′	**?P** rdfs:range ?C .	**?P** $\overset{\textbf{rdfs3′}}{\rightarrow}$?C
rdfs7′	**?P** rdfs:subPropertyOf ?Q .	**?P** $\overset{\textbf{rdfs7′}}{\rightarrow}$?Q
rdfs9	**?C** rdfs:subClassOf ?D .	**?C** $\overset{\textbf{rdfs9}}{\rightarrow}$?D
rdfp3′	**?P** a :SymmetricProperty .	**?P** $\overset{\textbf{rdfp3′}}{\rightarrow}$ True
rdfp8a′	**?P** :inverseOf ?Q .	**?P** $\overset{\textbf{rdfp8a′}}{\rightarrow}$?Q
rdfp8b′	?P :inverseOf **?Q** .	**?Q** $\overset{\textbf{rdfp8b′}}{\rightarrow}$?P
rdfp12a′	**?C** :equivalentClass ?D .	**?C** $\overset{\textbf{rdfp12a′}}{\rightarrow}$?D
rdfp12b′	?C :equivalentClass **?D** .	**?D** $\overset{\textbf{rdfp12b′}}{\rightarrow}$?C
rdfp13a′	**?P** :equivalentProperty ?Q .	**?P** $\overset{\textbf{rdfp13a′}}{\rightarrow}$?Q
rdfs13b′	?P :equivalentProperty **?Q** .	**?Q** $\overset{\textbf{rdfs13b′}}{\rightarrow}$?P
rdfp14a′	?C :hasValue ?y ; :onProperty **?P** .	**?P** $\overset{\textbf{rdfp14a′}}{\rightarrow}$ { ?C, ?y }
rdfp14b′	?C :hasValue ?y ; :onProperty ?P .	?C $\overset{\textbf{rdfp14b′}}{\rightarrow}$ { ?P, ?y }
rdfc1	?C :unionOf (?C$_1$...**?C**$_i$...?C$_n$) .	**?C**$_i$ $\overset{\textbf{rdfc1}}{\rightarrow}$?C
rdfc2	?C :minCardinality 1 ; :onProperty **?P** .	**?P** $\overset{\textbf{rdfc2}}{\rightarrow}$?C
rdfc3a	?C :intersectionOf (?C$_1$ ... ?C$_n$) .	?C $\overset{\textbf{rdfc3a}}{\rightarrow}$ { ?C$_1$, ..., ?C$_n$ }
rdfc3b	?C :intersectionOf (**?C**$_1$) .	**?C**$_1$ $\overset{\textbf{rdfc3b}}{\rightarrow}$?C
	$\mathcal{R}2$	
rdfp1′	**?P** a :FunctionalProperty .	**?P** $\overset{\textbf{rdfp1′}}{\rightarrow}$ True
rdfp2	**?P** a :InverseFunctionalProperty .	**?P** $\overset{\textbf{rdfp2}}{\rightarrow}$ True
rdfp4	**?P** a :TransitiveProperty .	**?P** $\overset{\textbf{rdfp4}}{\rightarrow}$ True
rdfp15′	?C :someValuesFrom **?D** ; :onProperty **?P** .	**?P** $\overset{\textbf{rdfp15′}}{\leftrightarrow}$ **?D** $\overset{\textbf{rdfp15′}}{\rightarrow}$?C
rdfp16′	?C :allValuesFrom ?D ; :onProperty ?P .	?P $\overset{\textbf{rdfp16′}}{\leftrightarrow}$?C $\overset{\textbf{rdfp16′}}{\rightarrow}$?D
rdfc3c	?C :intersectionOf (**?C**$_1$ **... ?C**$_n$) .	{ **?C**$_1$, **...**, **?C**$_n$ } $\overset{\textbf{rdfc3c}}{\rightarrow}$?C
rdfc4a	?C :cardinality 1 ; :onProperty ?P .	?C $\overset{\textbf{rdfc4a}}{\leftrightarrow}$?P
rdfc4b	?C :maxCardinality 1 ; :onProperty ?P .	?C $\overset{\textbf{rdfc4b}}{\leftrightarrow}$?P

Table 3: T-Box statements and how they are used to *wire* the concepts contained in the in-memory T-Box.

4.3.3 In-Memory T-Box

Before we proceed, we quickly discuss the storage of `:oneOf` constructs in the T-Box for rule **rdfc0**. Individuals `(?x`$_1$ ... `?x`$_n$`)` are stored with pointers to the one-of class `?C`. Before input data are read, these individuals are asserted to be of the `rdf:type` of their encompassing one-of class.

Besides the one-of support, for the in-memory T-Box we employ two separate hashtables, one for classes and another for properties, with RDF terms as key and a Java representation of the class or property as value. The representative Java objects contain labelled links to related objects as defined in Table 3. The property and class objects are designed to contain all of the information required for reasoning on a membership assertion of that property or class: that is, classes/properties satisfying the A-Box antecedent pattern of a rule are linked to the classes/properties appearing in the consequent of that rule, with the link labelled according to that rule. During reasoning, the class/property identifier used in the membership assertion is

sent to the corresponding hashtable and the returned internal object used for reasoning on that assertion. The objects contain the following:

- Property objects contain the property URI and references to objects representing domain classes (**rdfs2**), range classes (**rdfs3′**), super properties (**rdfs7′**), inverse properties (**rdfs8***) and equivalent properties (**rdfp13***). References are kept to restrictions where the property in question is the object of an `:onProperty` statement (**rdfp14a**, **rdfp16′**, **rdfc2**, **rdfc4***). Where applicable, if the property is part of a some-values-from restriction, a pointer is kept to the some-values-from class (**rdfp15′**). Boolean values are stored to indicate whether the property is functional (**rdfp1′**), inverse-functional (**rdfp2**), symmetric (**rdfp3′**) and/or transitive (**rdfp4**).
- Class objects contain the class URI and references to objects representing super classes (**rdfs9**), equivalent classes (**rdfp12***) and classes for which this class is a component of a union (**rdfc1**) or intersection (**rdfc3b/c**). On top of these core elements, different references are maintained for different types of class description:
  - intersection classes store references to their constituent class objects (**rdfc3a**)
  - restriction classes store a reference to the property the restriction applies to (**rdfp14b′**, **rdfp15′**, **rdfc2**, **rdfc4***) and also, if applicable to the type of restriction:
    - the values which the restriction property must have (**rdfp14b′**)
    - the class for which this class is a some-values-from restriction value (**rdfp15′**)

Figure 2 provides a UML-like representation of our T-Box, including multiplicities of the various links present between classes, properties and individuals labelled according to Table 3 for each rule.

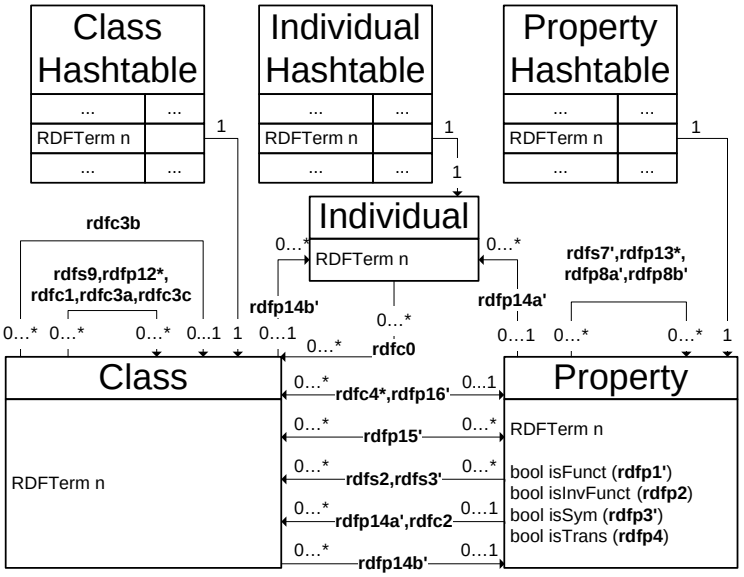

Figure 2: In-memory T-Box structure

The algorithm must also performs in-memory joining of collection segments according to `rdf:first` and `rdf:rest` statements found during the scan for the purposes of building union, intersection and enumeration class descriptions. Again, any remaining collections not relevant to the T-Box segment of the knowledge-base (i.e., not terminological collection statements) are discarded at the end of loading the input data; we also discard cyclic and branching lists as well as any lists not found to end with the `rdf:nil` construct.

We have now loaded the final T-Box for reasoning into memory; this T-Box will remain fixed throughout the whole reasoning process.

4.4 Initial Input Scan

Having loaded the terminological data, SAOR is now prepared for reasoning by statement-wise scan of the assertional data.

We provide the high-level flow for reasoning over an input statement in Function 1. The reasoning scan process can be described as recursive depth-first reasoning whereby each unique statement produced is again input immediately for reasoning. Statements produced thus far for the original input statement are kept in a set to provide uniqueness testing and avoid cycles; a uniquing function is also maintained for a common subject group in the data, ensuring that statements are only produced once for that statement group. Once all of the statements produced by a rule have been themselves recursively analysed, the reasoner moves on to analysing the proceeding rule and loops until no unique statements are inferred. The reasoner then processes the next input statement.

Function 1:ReasonStatement(s)

```
Input: statement s
Global: 𝕋, index, output
foreach rule r ∈ ℛ1 do
    fire rule r w.r.t s and 𝕋;
    foreach inferred statement: i do
        if i is unique then
            if i is valid RDF then
                write to output;
            ReasonStatement (i) ;
foreach rule r ∈ ℛ2 ∪ ℛ3 do
    if s relevant for r w.r.t. 𝕋 then
        write s to index_r ;
```

There are three disjoint categories of statements which require different handling: namely (i) `rdf:type` statements, (ii) `:sameAs` statements, (iii) all other statements. We assume disjointness between the statement categories: we do not allow any external extension of the core `rdf:type`/`:sameAs` semantics (non-standard use / non-authoritative extension). Further, the assertions about `rdf:type` in the RDFS specification define the `rdfs:domain` and `rdfs:range` of `rdf:type` as being `rdfs:Resource` and `rdfs:Class`; since we are not interested in inferring membership of such RDFS classes we do not subject `rdf:type` statements to property-based entailments. The only assertions about `:sameAs` from the OWL specification define domain and range as `:Thing` which we ignore by the same justification.

The `rdf:type` statements are subject to class-based entailment reasoning and require joins with class descriptions in the T-Box. The `:sameAs` statements are handled by ruleset $\mathcal{R}3$, which we discuss in Section 4.6. All other statements are subject to property-based entailments and thus requires joins with T-Box property descriptions.

Ruleset $\mathcal{R}2 \cup \mathcal{R}3$ cannot be computed solely on a statement-wise basis. Instead, for each rule, we assign an on-disk file (blocked and compressed to save disk space). Each file contains statements which may contribute to satisfying the antecedent of its pertinent rule. During the scan, if an A-Box statement satisfies the necessary T-Box join for a rule, it is written to the index for that rule. For example, when the statement

```
ex:me foaf:isPrimaryTopicOf ex:myHomepage .
```

is processed, the property object for `foaf:isPrimaryTopicOf` is retrieved from the T-Box property hashtable. The object states that this property is of type `:InverseFunctionalProperty` ($\overset{\textbf{rdfp2}}{\rightarrow}$ True). The rule cannot yet be fired as this statement alone does not satisfy the A-Box segment of the antecedent of **rdfp2** and the method is privy to only one A-Box statement at a time. When, later, the statement:

```
ex:me2 foaf:isPrimaryTopicOf ex:myHomepage .
```

is found, it also is written to the same file – the file now contains sufficient data to (although it cannot yet) fire the rule and infer:

```
ex:me :sameAs ex:me2 .
```

During the initial scan and inferencing, all files for ruleset $\mathcal{R}2 \cup \mathcal{R}3$ are filled with pertinent statements analogously to the example above. After the initial input statements have been exhausted, these files are analysed to infer, for example, the `:sameAs` statement above.

4.5 On-Disk A-Box Join Analysis

In this section, we discuss handling of the on-disk files containing A-Box statements for ruleset $\mathcal{R}2 \cup \mathcal{R}3$. We firstly give a general overview of the execution for each rule using an on-disk file and then look at the execution of each rule.

Table 4 presents the joins to be executed via the on-disk files for each rule: the key join variables, used for computing the join, are shown in boldface. In this table we refer to *SPOC and OPSC sorting order*: these can be intuitively interpreted as quads sorted according to subject, predicate, object, context (natural sorting order) and object, predicate, subject, context (inverse sorting order) respectively. For the internal index files, we use context to encode the sorting order of a statement and the iteration in which it was added; only joins with at least one new statement from the last iteration will infer novel output.

$\mathcal{R}2$		
rdfp1′	**?x** ?P ?y , ?z .	SPOC
rdfp2	?x ?P **?z** . ?y ?P **?z** .	OPSC
rdfp4	?x ?P **?y** . **?y** ?P ?z .	SPOC & OPSC
rdfp15′	?x ?P **?y** . **?y** a ?D .	SPOC / OPSC
rdfp16′	**?x** a ?C ; ?P ?y .	SPOC
rdfc3c	**?x** a $?C_1$, ..., $?C_n$.	SPOC
rdfc4a	**?x** a ?C ; ?P ?y , ?z .	SPOC
rdfc4b	**?x** a ?C ; ?P ?y , ?z .	SPOC

$\mathcal{R}3$		
rdfp7	?x :sameAs **?y** . **?y** :sameas ?z .	SPOC & OPSC
rdfp11′	**?x** :sameAs $?\_x$; ?P ?y .	SPOC
rdfp11″	**?y** :sameAs $?\_y$. ?x ?P **?y** .	SPOC / OPSC

Table 4: Table enumerating the A-Box joins to be computed using the on-disk files with key join position in boldface font and sorting order required for statements to compute join.

Again, an on-disk file is dedicated for each rule/join required. The joins to be computed are a simple "star shaped" join pattern or "one-hop" join pattern (which we reduce to a simple star shaped join computation by inverting one one or more patterns to inverse order). The statements in each file are initially sorted according to the key join variable. Thus, common bindings for the key join variable are grouped together and joins can be executed by means of sequential scan for common key join variable binding groups.

We now continue with a more detailed description of the process for each rule beginning with the more straightforward rules.

4.5.1 Functional Property Reasoning - Rule rdfp1′

From the initial input scan, we have a file containing only statements with functional properties in the predicate position (as described in Section 4.4). As can be seen from Table 4, the key join variable is in the subject position for all A-Box statements in the pattern. Thus, we can sort the file according to SPOC (natural) order. The result is a file where all statements are grouped according to a common subject, then predicate, then object. We can now scan this file, storing objects with a common subject-predicate. We can then fire the rule stating equivalence between these objects.

4.5.2 Inverse Functional Reasoning - Rule rdfp2

Reasoning on statements containing inverse functional properties is conducted analogously to functional property reasoning. However, the key join variable is now in the object position for all A-Box statements in the pattern. Thus, we instead sort the file according to OPSC (inverse) order and scan the file inferring equivalence between the subjects for a common object-predicate group.

4.5.3 Intersection Class Reasoning - Rule rdfc3c

The key join variable for rule **rdfc3c** is in the subject position for all A-Box triple patterns. Thus we can sort the file for the rule (filled with memberships assertions for classes which are part of some intersection) according to SPOC order. We can scan common subject-predicate (in any case, the predicates all have value `rdf:type`) groups storing the objects (all types for the subject resource which are part of an intersection). The containing intersection for each type can then be retrieved (through $\overset{\textbf{rdfc3c}}{\rightarrow}$) and the intersection checked to see if all of it's constituent types have been satisfied. If so, membership of the intersection is inferred.

4.5.4 All-Values-From Reasoning - Rule rdfp16′

Again, the key join variable for rule **rdfp16′** is in the subject position for all A-Box triple patterns and again we can sort the file according to SPOC order. For a common subject group, we store `rdf:type` values

and also all predicate/object edges for the given subject. For every member of an all-values-from restriction class (as is given by all of the `rdf:type` statements in the file according to the join with the T-Box on the `?C` position), we wish to infer that objects of the `:onProperty` value (as is given by all the non-`rdf:type` statements according to the T-Box join with `?P` – where `?P` is linked from `?C` with $\overset{\textbf{rdfp16}'}{\rightarrow}$) are of the all-values-from class. Therefore, for each restriction membership assertion, the objects of the corresponding `:onProperty`-value membership-assertions are inferred to be members of the all-values-from object class (`?D`).

4.5.5 Some-Values-From Reasoning - Rule rdfp15′

For some-values-from reasoning, the key join variable is in the subject position for `rdf:type` statements (all membership assertions of a some-values-from object class) but in the object position for the `:onProperty` value membership assertions. Thus, we order class membership assertions in the file according to natural SPOC order and property membership assertions according to inverse OPSC order. In doing so, we can scan common `?y` binding groups in the file, storing `rdf:type` values and also all predicate/subject edges. For every member of a some-values-from object class (as is given by all of the `rdf:type` statements in the file according to the join with the T-Box on the `?D` position), we infer that subjects of the `:onProperty`-value statements (as is given by all the non-`rdf:type` statements according to the T-Box join with `?P`) are members of the restriction class (`?C`).

4.5.6 Transitive Reasoning (Non-Symmetric) - Rule rdfp4

Transitive reasoning is perhaps the most challenging to compute: the output of rule **rdfp4** can again recursively act as input to the rule. For closure, recursive application of the rule must be conducted in order to traverse arbitrarily long transitive paths in the data.

Firstly, we will examine sorting order. The key join variable is in the subject position for one pattern and in the object position for the second pattern. However, both patterns are identical: a statement which matches one pattern will obviously match the second. Thus, every statement in the transitive reasoning file is duplicated with one version sorted in natural SPOC order, and another in inverse OPSC.

Take, for example, the following triples where `ex:comesBefore` is asserted in the T-Box as being of type `:TransitiveProperty`:

 # INPUT:

```
ex:a ex:comesBefore ex:b .
ex:b ex:comesBefore ex:c .
ex:c ex:comesBefore ex:d .
```

In order to compute the join, we must write the statements in both orders, using the context to mark which triples are in inverse order, and sort them accordingly (for this internal index, we temporarily relax the requirement that context is a URI).

 # SORTED FILE - ITERATION 1:[13]

```
ex:a ex:comesBefore ex:b _:spoc1 .
ex:b ex:comesBefore ex:a _:opsc1 .
ex:b ex:comesBefore ex:c _:spoc1 .
ex:c ex:comesBefore ex:b _:opsc1 .
ex:c ex:comesBefore ex:d _:spoc1 .
ex:d ex:comesBefore ex:c _:opsc1 .
```

The data, as above, can then be scanned and for each common join-binding/predicate group (e.g., `ex:b ex:comesBefore`), the subjects of statements in inverse order (e.g., `ex:a`) can be linked to the object of naturally ordered statements (e.g., `ex:c`) by the transitive property. However, such a scan will only compute a single one-hop join. From above, we only produce:

 # OUTPUT - ITERATION 1 / INPUT - ITERATION 2

```
ex:a ex:comesBefore ex:c .
ex:b ex:comesBefore ex:d .
```

[13] In N-Quads format: c.f. `http://sw.deri.org/2008/07/n-quads/`

We still not have not computed the valid statement `ex:a ex:comesBefore ex:d .` which requires a two hop join. Thus we must iteratively feedback the results from one scan as input for the next scan. The output from the first iteration, as above, is also reordered and sorted as before and merge-sorted into the main ***SORTED FILE***.

 # SORTED FILE - ITERATION 2:

```
ex:a ex:comesBefore ex:b _:spoc1 .
ex:a ex:comesBefore ex:c _:spoc2 .
ex:b ex:comesBefore ex:a _:opsc1 .
ex:b ex:comesBefore ex:c _:spoc1 .
ex:b ex:comesBefore ex:d _:spoc2 .
ex:c ex:comesBefore ex:a _:opsc2 .
ex:c ex:comesBefore ex:b _:opsc1 .
ex:c ex:comesBefore ex:d _:spoc1 .
ex:d ex:comesBefore ex:b _:opsc2 .
ex:d ex:comesBefore ex:c _:opsc1 .
```

The observant reader may already have noticed from above that we also mark the context with the iteration for which the statement was added. In every iteration, we only compute inferences which involve the delta from the last iteration; thus the process is comparable to semi-naïve evaluation. Only joins containing at least one newly added statement are used to infer new statements for output. Thus, from above, we avoid repeat inferences from ***ITERATION 1*** and instead infer:

 # OUTPUT - ITERATION 2:

```
ex:a ex:comesBefore ex:d .
```

A fixpoint is reached when no new statements are inferred. Thus we would require another iteration for the above example to ensure that no new statements are inferable. The number of iterations required is in $\mathcal{O}(\log n)$ according to the longest unclosed transitive path in the input data. Since the algorithm requires scanning of not only the delta, but also the entire data, performance using on-disk file scans alone would be sub-optimal. For example, if one considers that most of the statements constitute paths of, say $\leq$8 vertices, one path containing 128 vertices would require four more scans after the bulk of the paths have been closed.

With this in mind, we accelerate transitive closure by means of an in-memory transitivity index. For each transitive property found, we store sets of linked lists which represent the graph extracted for that property. From the example ***INPUT*** from above, we would store.

```
ex:comesBefore -- ex:a -> ex:b -> ex:c -> ex:d
```

From this in-memory linked list, we would then collapse all paths of length $\geq$2 (all paths of length 1 are input statements) and infer closure at once:

 # OUTPUT - ITERATION 1 / INPUT - ITERATION 2

```
ex:a ex:comesBefore ex:c .
ex:a ex:comesBefore ex:d .
ex:b ex:comesBefore ex:d .
```

Obviously, for scalability requirements, we do not expect the entire transitive body of statements to fit in-memory. Thus, before each iteration we calculate the in-memory capacity and only store a pre-determined number of properties and vertices. Once the in-memory transitive index is full, we infer the appropriate statements and continue by file-scan. The in-memory index is only used to store the delta for a given iteration (everything for the first iteration). Thus, we avoid excess iterations to compute closure of a small percentage of statements which form a long chain and greatly accelerate the fixpoint calculation.

4.5.7 Transitive Reasoning (Symmetric) - Rules rdfp3′/rdfp4

We use a separate on-disk file for membership assertions of properties which are both transitive *and* symmetric. A graph of symmetric properties is direction-less, thus the notion of direction as evident above though use of inverted ordered statements is unnecessary. Instead, all statements and their inverses (computed from symmetric rule **rdfp3′**) are written in natural SPOC order and direct paths are inferred between all objects in a common subject/predicate group. The in-memory index is again similar to above; however, we instead use a direction-less doubly-linked list.

4.6 Equality Reasoning

Thus far, we have not considered `:sameAs` entailment, which is supported in SAOR through rules in $\mathcal{R}3$. Prior to executing rules **rdfp11′** & **rdfp11″**, we must first perform symmetric transitive closure on the list of all `:sameAs` statements (rules **rdfp6′** & **rdfp7**). Thus, we use an on-disk file analogous to that described in Section 4.5.7.

However, for rules **rdfp6′** & **rdfp7**, we do not wish to experience an explosion of inferencing through long equivalence chains (lists of equivalent individuals where there exists a `:sameAs` path from each individual to every other individual). The closure of a symmetric transitive chain of n vertices results in $n(n-1)$ edges or statements (ignoring reflexive statements). For example, in [23] we found a chain of 85,803 equivalent individuals inferable from a web dataset.[14] Naïvely applying symmetric transitive reasoning as discussed in Section 4.5.7 would result in a closure of 7.362b `:sameAs` statements for this chain alone.

Similarly, `:sameAs` entailment, as according to rules **rdfp11′** & **rdfp11″**, duplicates data for all equivalent individuals which could result in a massive amount of duplicate data (particularly when considering uniqueness on a quad level: i.e., including duplicate triples from different sources). For example, if each of the 85,803 equivalent individuals had attached an average of 8 unique statements, then this could equate to 8*85,803*85,803 = 59b inferred statements.

Obviously, we must avoid the above scenarios, so we break from complete inference with respect to the rules in $\mathcal{R}3$. Instead, for each set of equivalent individuals, we chose a pivot identifier to use in rewriting the data. The pivot identifier is used to keep a consistent identifier for the set of equivalent individuals: the alphabetically highest pivot is chosen for convenience of computation. For alternative choices of pivot identifiers on web data see [23]. We use the pivot identifier to consolidate data by rewriting all occurrences of equivalent identifiers to the pivot identifier (effectively merging the equivalent set into one individual).

Thus, we do not derive the entire closure of `:sameAs` statements as indicated in rules **rdfp6′** & **rdfp7** but instead only derive an equivalence list which points from equivalent identifiers to their pivots. As highlighted, use of a pivot identifier is necessary to reduce the amount of output statements, effectively compressing equivalent resource descriptions: we hint here that a fully expanded view of the descriptions could instead be supported through backward-chaining over the semi-materialised data.

To achieve the pivot compressed inferences we use an on-disk file containing `:sameAs` statements. Take for example the following statements:

 # INPUT

```
ex:a :sameAs ex:b .
ex:b :sameAs ex:c .
ex:c :sameAs ex:d .
```

We only wish to infer the following output for the pivot identifier `ex:a`:

 # OUTPUT PIVOT EQUIVALENCES

```
ex:b :sameAs ex:a .
ex:c :sameAs ex:a .
ex:d :sameAs ex:a .
```

The process is the same as that for symmetric transitive reasoning as described before: however, we only close transitive paths to nodes with the highest alphabetical order. So, for example, if we have already materialised a path from `ex:d` to `ex:a` we ignore inferring a path from `ex:d` to `ex:b` as `ex:b` > `ex:a`.

To execute rules **rdfp11′** & **rdfp11″** and perform "consolidation" (rewriting of equivalent identifiers to their pivotal form), we perform a zig-zag join: we sequentially scan the `:sameAs` inference output as above and an appropriately sorted file of data, rewriting the latter data according to the `:sameAs` statements. For example, take the following statements to be consolidated:

 # UNCONSOLIDATED DATA

```
ex:a foaf:mbox <mail@example.org> .
...
ex:b foaf:mbox <mail@example.org> .
ex:b foaf:name "Joe Bloggs" .
...
```

[14] This is from incorrect use of the FOAF ontology by prominent exporters. We refer the interested reader to [23]

```
ex:d :sameAs ex:b .
...
ex:e foaf:knows ex:d .
```

The above statements are scanned sequentially with the closed `:sameAs` pivot output from above. For example, when the statement `ex:b foaf:mbox <mailto:mail@example.org> .` is first read from the unconsolidated data, the `:sameAs` index is scanned until `ex:b :sameAs ex:a .` is found (if `ex:b` is not found in the `:sameAs` file, the scan is paused when an element above the sorting order of `ex:b` is found). Then, `ex:b` is rewritten to `ex:a`.

 # PARTIALLY CONSOLIDATED DATA

```
ex:a foaf:mbox <mail@example.org> .
...
ex:a foaf:mbox <mail@example.org> .
ex:a foaf:name "Joe Bloggs" .
...
ex:a :sameAs ex:b .
...
ex:e foaf:knows ex:d .
```

We have now executed rule **rdfp11**$'$ and have the data partially consolidated as shown. However, the observant reader will notice that we have not consolidated the object of the last two statements. We must sort the data again according to inverse OPSC order and again sequentially scan both the partially consolidated data and the `:sameAs` pivot equivalences, this time rewriting `ex:b` and `ex:d` in the object position to `ex:a` and producing the final consolidated data. This equates to executing rule **rdfp11**$''$.

For the purposes of the on-disk files for computing rules requiring A-Box joins, we must consolidate the key join variable bindings according to the `:sameAs` statements found during reasoning. For example consider the following statements in the functional reasoning file:

```
ex:a ex:mother ex:m1 .
ex:b ex:mother ex:m2 .
```

Evidently, rewriting the key join position according to our example pivot file will lead to inference of:

```
ex:m1 :sameAs ex:m2 .
```

which we would otherwise miss. Thus, whenever the index of `:sameAs` statements is changed, for the purposes of closure it is necessary to attempt to rewrite all join index files according to the new `:sameAs` statements. Since we are, for the moment, only concerned with consolidating on the join position we need only apply one consolidation scan.

The final step in the SAOR reasoning process is to finalise consolidation of the initial input data and the newly inferred output statements produced by all rules from scanning and on-disk file analysis. Although we have provided exhaustive application of all inferencing rules, and we have the complete set of `:sameAs` statements, elements in the input and output files may not be in their equivalent pivotal form. Therefore, in order to ensure proper consolidation of all of the data according to the final set of `:sameAs` statements, we must firstly sort both input and inferred sets of data in SPOC order, consolidate subjects according to the pivot file as above; sort according to OPSC order and consolidate objects.

However, one may notice that `:sameAs` statements in the data become consolidated into reflexive statements: i.e., from the above example `ex:a :sameAs ex:a`. Thus, for the final output, we remove any `:sameAs` statements in the data and instead merge the statements contained in our final pivot `:sameAs` equivalence index, and their inverses, with the consolidated data. These statements retain the list of all possible identifiers for a consolidated entity in the final output.

4.7 Achieving Closure

We conclude this section by summarising the approach, detailing the overall fixpoint calculations (as such, putting the jigsaw together) and detailing how closure is achieved using the individual components. Along these lines, in Algorithm 2, we provide a summary of the steps seen so far and, in particular, show the fixpoint calculations involved for exhaustive application of ruleset $\mathcal{R}2 \cup \mathcal{R}3$; we compute one main fixpoint over all of the operations required, within which we also compute two local fixpoints.

Firstly, since all rules in $\mathcal{R}2$ are dependant on `:sameAs` equality, we perform `:sameAs` inferences first. Thus, we begin closure on $\mathcal{R}2 \cup \mathcal{R}3$ with a local equality fixpoint which (i) executes all rules which produce `:sameAs` inferences (**rdfp1′**,**rdfp2**,**rdfc4***); (ii) performs symmetric-transitive closure using pivots on all `:sameAs` inferences; (iii) rewrites **rdfp1′**, **rdfp2** and **rdfc4*** indexes according to `:sameAs` pivot equivalences and (iv) repeats until no new `:sameAs` statements are produced.

Next, we have a local transitive fixpoint for recursively computing transitive property reasoning: (i) the transitive index is rewritten according to the equivalences found through the above local fixpoint; (ii) a transitive closure iteration is run, output inferences are recursively fed back as input; (iii) ruleset $\mathcal{R}1$ is also recursively applied over output from previous step whereby the output from ruleset $\mathcal{R}1$ may also write new statements to any $\mathcal{R}2$ index. The local fixpoint is reached when no new transitive inferences are computed.

Finally, we conclude the main fixpoint by running the remaining rules: **rdfp15′**, **rdfp16′** and **rdfc3c**. For each rule, we rewrite the corresponding index according to the equivalences found from the first local fixpoint, run the inferencing over the index and send output for reasoning through ruleset $\mathcal{R}1$. Statements inferred directly from the rule index, or through subsequent application of ruleset $\mathcal{R}1$, may write new statements for $\mathcal{R}2$ indexes. This concludes one iteration of the main fixpoint, which is run until no new statements are inferred.

For each ruleset $\mathcal{R}0-3$, we now justify our algorithm in terms of our definition of closure with respect to our static T-Box. Firstly, closure is achieved immediately upon ruleset $\mathcal{R}0$, which requires only T-Box knowledge, from our static T-Box. Secondly, with respect to the given T-Box, every input statement is subject to reasoning according to ruleset $\mathcal{R}1$, as is every statement inferred from ruleset $\mathcal{R}0$, those recursively inferred from ruleset $\mathcal{R}1$ itself, and those recursively inferred from on-disk analysis for ruleset $\mathcal{R}1 \cup \mathcal{R}2$. Next, every input statement is subject to reasoning according to ruleset $\mathcal{R}2$ with respect to our T-Box; these again include all inferences from $\mathcal{R}0$, all statements inferred through $\mathcal{R}1$ alone, and all inferences from recursive application of ruleset $\mathcal{R}1 \cup \mathcal{R}2$.

Algorithm 2: SAOR reasoning algorithm

```
Input: 𝕂𝔹
Output: Cl_R̂(𝕋, 𝕂𝔹)
for scan 𝕂𝔹 (Section 4.3.1) do
  obtain candidate statements for 𝕋;
reduce 𝕋, derive T̂ (Sect. 4.3.2) ;
load T̂ in-memory (Sect. 4.3.3) ;
run R0 rules ;
for inferred statement i do
  if i is valid RDF then
    write to output;
  ReasonStatement (i) (Funct. 1) ;
for s ∈ 𝕂𝔹 (Sect. 4.4) do
  ReasonStatement (s) (Funct. 1) ;
// output contains R0 ∪ R1 inferences for initial input
// index contains initial statements relevant for R2
// sameas contains initial statements relevant for R3
repeat
  repeat
    for rule r ∈ {rdfp1′,rdfp2,rdfc4*} (Sect. 4.5) do
      if new_r is set then
        rewrite index_r w.r.t. sameas/ unset new_r;
      if index_r has changed or was rewritten then
        run r on index_r;
        write to sameas;
    if sameas has changed then
      run rules rdfp6′ and rdfp7 on sameas (Sect. 4.6) ;
      write sameas/ set all new;
  until fixpoint reached (no changes in previous iteration)
  ;
  if new_rdfp4 is set then
    rewrite index_rdfp4 w.r.t. sameas/ unset new_rdfp4;
  repeat
    run rule rdfp4 on index_rdfp4 (Sect. 4.5.6 and 4.5.7) ;
    for inferred statement i do
      write to index_rdfp4 ;
      if i is RDF then
        write to output;
      ReasonStatement (i) (Funct. 1) ;
  until fixpoint reached (no changes in previous iteration)
  ;
  for rule r ∈ {rdfp15′,rdfp16′,rdfc3c} (Sect. 4.5) do
    if new_r is set then
      rewrite index_r w.r.t. sameas/ unset new_r;
    if index_r has changed or was rewritten then
      run r on index_r;
      for inferred statement i do
        if i is RDF then
          write to output;
        ReasonStatement (i) (Funct. 1) ;
until fixpoint reached (no changes in previous iteration) ;
// output contains all inferences in non-pivotal form
for subject and object (Sect. 4.6) do
  for scan 𝕂𝔹 and output do
    rewrite according to sameas;
    write to Cl_R̂(𝕋, 𝕂𝔹);
  write sameas and sameas⁻ to Cl_R̂(𝕋, 𝕂𝔹);
```

Therefore, we can see that our algorithm applies exhaustive application of ruleset $\mathcal{R}0 \cup \mathcal{R}1 \cup \mathcal{R}2$ with respect to our T-Box, leaving only consideration of equality reasoning in ruleset $\mathcal{R}3$. Indeed, our algorithm is not complete with respect to ruleset $\mathcal{R}3$ since we choose pivot identifiers for representing equivalent individuals as justified in Section 4.6. However, we still provide a form of "pivotal closure" whereby backward-chaining support of rules **rdfp11′** and **rdfp11″** over the output of our algorithm would provide a view of closure as defined; i.e., our output contains all of the possible inferences according to our notion of closure, but with equivalent individuals compressed in pivotal form.

Firstly, for rules **rdfp6′** and **rdfp7**, all statements where p = `:sameAs` from the original input or as produced by $\mathcal{R}0 \cup \mathcal{R}1 \cup \mathcal{R}2$ undergo on-disk symmetric-transitive closure in pivotal form. Since both rules only produce more `:sameAs` statements, and according to the standard usage restriction of our closure, they are not applicable to reasoning under $\mathcal{R}0 \cup \mathcal{R}1 \cup \mathcal{R}2$. Secondly, we loosely apply rules **rdfp11′** and **rdfp11″** such as to provide closure with respect to joins in ruleset $\mathcal{R}2$; i.e., all possible joins are computed with respect to the given `:sameAs` statements. Equivalence is clearly not important to $\mathcal{R}0$ since we strictly do not allow `:sameAs` statements to affect our T-Box; $\mathcal{R}1$ inferences do not require joins and, although the statements produced will not be in pivotal form, they will be output and rewritten later; inferences from $\mathcal{R}2$ will be produced as discussed, also possibly in non-pivotal form. In the final consolidation step, we then rewrite all statements to their pivotal form and provide incoming and outgoing `:sameAs` relations between pivot identifiers and their non-pivot equivalent identifiers. This constitutes our output, which we call *pivotal authoritative closure*.

5 Evaluation and Discussion

We now provide evaluation of the SAOR methodology firstly with quantitative analysis of the importance of authoritative reasoning, and secondly we provide performance measurements and discussion along with insights into the fecundity of each rule w.r.t. reasoning over web data. All experiments are run on one machine with a single Opteron 2.2 GHz CPU and 4 GB of main memory. We provide evaluation on two datasets: we provide complete evaluation for a dataset of 147m statements collected from 665k sources and scale-up experiments running scan-reasoning (rules in $\mathcal{R}0 \cup \mathcal{R}1$) on a dataset of 1.1b statements collected from 6.5m sources; both datasets are from web-crawls using MultiCrawler [21].

We create a unique set of blank nodes for each graph $\mathcal{W}' \in M(\mathcal{S}_w)$ using a function on c and the original blank node label which ensures a one-to-one mapping from the original blank node labels and uniqueness of the blank nodes for a given context c.

To show the effects of ontology hijacking we constructed two T-Boxes with and without authoritative analysis for each dataset. We then ran reasoning on single membership assertions for the top five classes and properties found natively in each dataset. Table 5 summarises the results. Taking `foaf:Person` as an example, with an authoritative T-Box, six statements are output for every input `rdf:type foaf:Person` statement in both datasets. With the non-authoritative T-Box, 388 and 4,631 statements are output for every such input statement for the smaller and larger datasets respectively. Considering that there are 3.25m and 63.33m such statements in the respective datasets, overall output for `rdf:type foaf:Person` input statements alone approach 1.26b and 293b statements for non-authoritative reasoning respectively. With authoritative reasoning we only produce 19.5m and 379.6m statements, a respective saving of 65x and 772x on output statement size.[15]

It should be noted that reasoning on a membership assertion of the top level class (`:Thing`/`rdfs:Resource`) is very large for both the 147m (234 inferences) and the 1.1b dataset (4251 inferences). For example, in both datasets, there are many `:unionOf` class descriptions with `:Thing` as a member;[16] for the 1.1b dataset, many inferences on the top level classes stem from, for example, the OWL W3C Test Repository[17]. Of course we do not see such documents as being malicious in any way, but clearly they would cause inflationary inferences when naïvely considered as part of web knowledge-base.

Next, we present some metrics regarding the first step of reasoning: the separation and in-memory construction of the T-Box. For the 1.1b dataset, the initial scan of all data found 9,683,009 T-Box statements (0.9%). Reducing the T-Box by removing collection statements as described in Section 4.3.1 dropped a further 1,091,698 (11% of total) collection statements leaving 733,734 such statements in the T-Box (67% collection statements dropped) and 8,591,311 (89%) total. Table 6 shows, for membership assertions of each class and property in $\mathcal{C}_{SAOR}$ and $\mathcal{P}_{SAOR}$, the result of applying authoritative analysis. Of the 33,157 unique namespaces probed, 769 (2.3%) had a redirect, 4068 (12.3%) connected but had no redirect and 28,320

[15] For example, the document retrievable from `http://pike.kw.nl/files/documents/pietzwart/RDF/PietZwart200602.owl` defines super-classes/-properties for all of the FOAF vocabulary.

[16] Fifty-five such `:unionOf` class descriptions can be found in `http://lsdis.cs.uga.edu/~oldham/ontology/wsag/wsag.owl`; 34 are in `http://colab.cim3.net/file/work/SICoP/ontac/reference/ProtegeOntologies/COSMO-Versions/TopLevel06.owl`.

[17] `http://www.w3.org/2002/03owlt/`

147m Dataset					
C	$\lvert Cl_{\mathcal{R}1}(\widehat{\mathbb{T}}, \{m(C)\})\rvert$	$\lvert Cl_{\mathcal{R}1}(\mathbb{T}, \{m(C)\})\rvert$	n	$n\lvert Cl_{\mathcal{R}1}(\widehat{\mathbb{T}}, \{m(C)\})\rvert$	$n\lvert Cl_{\mathcal{R}1}(\mathbb{T}, \{m(C)\})\rvert$
rss:item	0	356	3,558,055	0	1,266,667,580
foaf:Person	6	388	3,252,404	19,514,424	1,261,932,752
rdf:Seq	2	243	1,934,852	3,869,704	470,169,036
foaf:Document	1	354	1,750,365	1,750,365	619,629,210
wordnet:Person	0	236	1,475,378	0	348,189,208
TOTAL	9	1,577	11,971,054	25,134,493	3,966,587,786
P	$\lvert Cl_{\mathcal{R}1}(\{\widehat{\mathbb{T}}, m(P)\})\rvert$	$\lvert Cl_{\mathcal{R}1}(\mathbb{T}, \{m(P)\})\rvert$	n	$n\lvert Cl_{\mathcal{R}1}(\{\widehat{\mathbb{T}}, m(P)\})\rvert$	$n\lvert Cl_{\mathcal{R}1}(\mathbb{T}, \{m(P)\})\rvert$
dc:title*	0	14	5,503,170	0	77,044,380
dc:date*	0	377	5,172,458	0	1,950,016,666
foaf:name*	3	418	4,631,614	13,894,842	1,936,014,652
foaf:nick*	0	390	4,416,760	0	1,722,536,400
rss:link*	1	377	4,073,739	4,073,739	1,535,799,603
TOTAL	4	1,576	23,797,741	17,968,581	7,221,411,701
1.1b Dataset					
C	$\lvert Cl_{\mathcal{R}1}(\widehat{\mathbb{T}}, \{m(C)\})\rvert$	$\lvert Cl_{\mathcal{R}1}(\mathbb{T}, \{m(C)\})\rvert$	n	$n\lvert Cl_{\mathcal{R}1}(\widehat{\mathbb{T}}, \{m(C)\})\rvert$	$n\lvert Cl_{\mathcal{R}1}(\mathbb{T}, \{m(C)\})\rvert$
foaf:Person	6	4,631	63,271,689	379,630,134	293,011,191,759
foaf:Document	1	4,523	6,092,322	6,092,322	27,555,572,406
rss:item	0	4,528	5,745,216	0	26,014,338,048
oboInOwl:DbXref	0	0	2,911,976	0	0
rdf:Seq	2	4,285	2,781,994	5,563,988	11,920,844,290
TOTAL	9	17,967	80,803,197	391,286,444	358,501,946,503
P	$\lvert Cl_{\mathcal{R}1}(\widehat{\mathbb{T}}, \{m(P)\})\rvert$	$\lvert Cl_{\mathcal{R}1}(\mathbb{T}, \{m(P)\})\rvert$	n	$n\lvert Cl_{\mathcal{R}1}(\widehat{\mathbb{T}}, \{m(P)\})\rvert$	$n\lvert Cl_{\mathcal{R}1}(\mathbb{T}, \{m(P)\})\rvert$
rdfs:seeAlso	2	8,647	113,760,738	227,521,476	983,689,101,486
foaf:knows	14	9,269	77,335,237	1,082,693,318	716,820,311,753
dc:title*	0	4,621	71,321,437	0	329,576,360,377
foaf:nick*	0	4,635	65,855,264	0	305,239,148,640
foaf:weblog	7	9,286	55,079,875	385,559,125	511,471,719,250
TOTAL	23	36,458	383,352,551	1,695,773,919	2,846,796,641,506

Table 5: Comparison of authoritative and non-authoritative reasoning for the number of unique inferred RDF statements produced (w.r.t. ruleset $\mathcal{R}1$) over the five most frequently occurring classes and properties in both input datasets. '*' indicates a datatype property where the object of $m(P)$ is a literal. The amount of statements produced for authoritative reasoning for a single membership assertion of the class or property is denoted by $\left|Cl_{\mathcal{R}1}(\widehat{\mathbb{T}}, \{m(C)\})\right|$ and $\left|Cl_{\mathcal{R}1}(\widehat{\mathbb{T}}, \{m(P)\})\right|$ respectively. Non-authoritative counts are given by $|Cl_{\mathcal{R}1}(\mathbb{T}, \{m(C)\})|$ and $|Cl_{\mathcal{R}1}(\mathbb{T}, \{m(P)\})|$. n is the number of membership assertions for the class C or property P in the given dataset.

Property	AuthSub	AuthObj	AuthBoth	AuthNone	Total	Drop
rdfs:subClassOf	25,076	**583,399**	1,595,850	**1,762,414**	3,966,739	2,345,813
:onProperty	1,041,873	-	97,921	-	1,139,843	-
:someValuesFrom	681,968	-	217,478	-	899,446	-
rdf:first	273,805	-	392,707	-	666,512	-
rdf:rest	249,541	-	416,946	-	666,487	-
:equivalentClass	574	189,912	162,886	**3,198**	356,570	3,198
:intersectionOf	-	-	216,035	-	216,035	-
rdfs:domain	5,693	**7,788**	66,338	**79,748**	159,567	87,536
rdfs:range	32,338	**4,340**	37,529	**75,338**	149,545	79,678
:hasValue	9,903	0	82,853	0	92,756	-
:allValuesFrom	51,988	-	22,145	-	74,133	-
rdfs:subPropertyOf	3,365	**147**	22,481	**26,742**	52,734	26,888
:maxCardinality	26,963	-	-	-	26,963	-
:inverseOf	75	52	6,397	**18,363**	24,887	18,363
:cardinality	20,006	-	-	-	20,006	-
:unionOf	-	-	21,671	-	21,671	-
:minCardinality	15,187	-	-	-	15,187	-
:oneOf	-	-	6,171	-	6,171	-
:equivalentProperty	105	24	187	**696**	1,012	696
Class						
:FunctionalProperty	9,616	-	-	**18,111**	27,727	18,111
:InverseFunctionalProperty	872	-	-	**3,080**	3,952	3,080
:TransitiveProperty	807	-	-	**1,994**	2,801	1,994
:SymmetricProperty	265	-	-	**351**	616	351
OVERALL	2,450,020	785,661	3,365,595	1,990,035	8,591,311	2,585,708

Table 6: Authoritative analysis of T-Box statements in 1.1b dataset for each primitive where dropped statements are highlighted in bold

(85.4%) did not connect at all. In total, 14,227,116 authority checks were performed. Of these, 6,690,704 (47%) were negative and 7,536,412 (53%) were positive. Of the positive, 4,236,393 (56%) were blank-nodes, 2,327,945 (31%) were a direct match between namespace and source and 972,074 (13%) had a redirect from the namespace to the source. In total, 2,585,708 (30%) statements were dropped as they could not contribute to a valid authoritative inference. The entire process of separating, analysing and loading the T-Box into memory took 6.47 hours: the most costly operation here is the large amount of HTTP lookups required for authoritative analysis, with many connections unsuccessful after our five second timeout. The process required ∼3.5G of Java heap-space and ∼10M of stack space.

For the 147m dataset, 2,649,532 (1.7%) T-Box statements were separated from the data, which was reduced to 1,609,958 (61%) after reducing the amount of irrelevant collection statements; a further 536,564 (33%) statements were dropped as they could not contribute to a valid authoritative inference leaving 1,073,394 T-Box statements (41% of original). Loading the T-Box into memory took approximately 1.04 hours.

We proceed by evaluating the application of reasoning over all rules on the 147m dataset with respect to throughtput of statements written and read.

Figure 3 shows performance for reaching an overall fixpoint for application of all rules. Clearly, the performance plateaus after 79 mins. At this point the input statements have been exhausted, with rules in $\mathcal{R}0$ and $\mathcal{R}1$ having been applied to the input data and statements written to the on-disk files for $\mathcal{R}2$ and $\mathcal{R}3$. SAOR now switches over to calculating a fixpoint over the on-disk computed $\mathcal{R}2$ and $\mathcal{R}3$ rules, the results of which become the new input for $\mathcal{R}1$ and further recursive input to the $\mathcal{R}2$ and $\mathcal{R}3$ files.

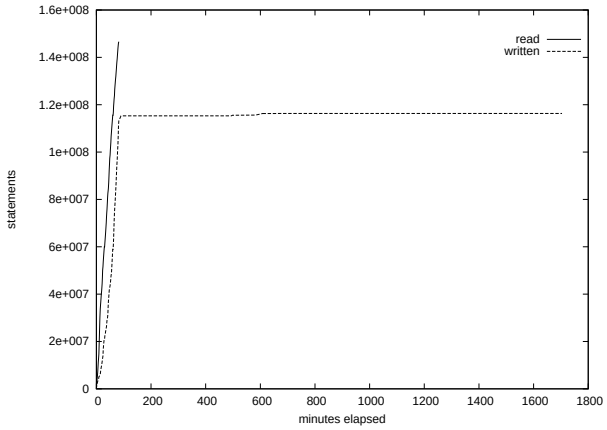

Figure 3: Performance of applying entire ruleset on the 147m statements dataset (without final consolidation step)

Figure 4 shows performance specifically for achieving closure on the on-disk $\mathcal{R}2$ and $\mathcal{R}3$ rules. There are three pronounced steps in the output of statements. The first one shown at (a) is due to inferencing of `:sameAs` statements from rule **rdfp2** (`:InverseFunctionalProperty` - 2.1m inferences). Also part of the first step are `:sameAs` inferences from rules **rdfp1′** (`:FunctionalProperty` - 31k inferences) and rules **rdfc4*** (`:cardinality`/`:maxCardinality` - 449 inferences). For the first plateau shown at (b), the `:sameAs` equality file is closed for the first time and a local fixpoint is being calculated to derive the initial `:sameAs` statements for future rules; also during the plateau at (b), the second iteration for the `:sameAs` fixpoint (which, for the first time, consolidates the key join variables in files for rules **rdfp2**, **rdfp1′**, **rdfc4a**, **rdfc4b** according to all `:sameAs` statements produced thus far) produces 1,018 new such statements, with subsequent iterations producing 145, 2, and 0 new statements respectively.

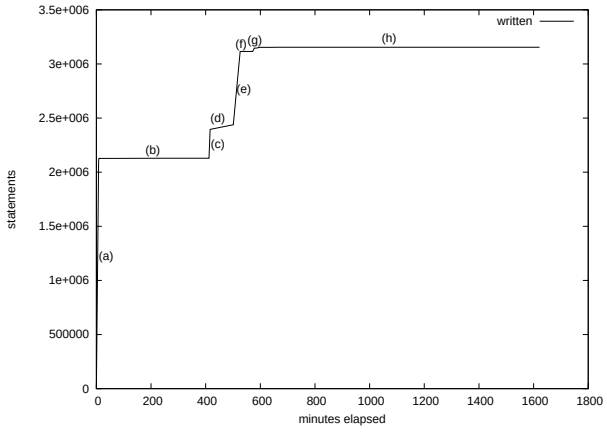

Figure 4: Performance of inferencing over $\mathcal{R}2$ and $\mathcal{R}3$ on-disk indexes for the 147m statements dataset (without final consolidation)

The second pronounced step at (c) is attributable to 265k transitive inferences, followed by 1.7k symmetric-transitive inferences. The proceeding slope at (d) is caused by inferences on **rdfc3c** (`:intersectionOf` - 265 inferences) and **rdfp15′** (`:someValuesFrom` - 36k inferences), with rule **rdfp16′**

(:allValuesFrom - 678k inferences) producing the final significant step at (e). The first complete iteration of the overall fixpoint calculation is now complete.

Since the first local :sameAs fixpoint, 22k mostly rdf:type statements have been written back to the cardinality rule files, 4 statements to the :InverseFunctionalProperty file and 14 to the :FunctionalProperty file. Thus, the :sameAs fixpoint is re-executed at (f), with no new statements found. The final, minor, staggered step at (g) occurs after the second :sameAs fixpoint when, most notably, rule **rdfp4** (:TransitiveProperty) produces 24k inferences, rule **rdfc3c** (:intersectionOf) produces 6.7k inferences, and rule **rdfp16′** (:allValuesFrom) produces 7.3k new statements.

The final, extended plateau at (h) is caused by rules which produce/consume rdf:type statements. In particular, the fixpoint encounters :allValuesFrom inferencing producing a minor contribution of statements (≤ 2) which lead to an update and re-execution of :allValuesFrom inferencing and :intersectionOf reasoning. In particular, :allValuesFrom required 66 recursive iterations to reach a fixpoint. We identified the problematic data as follows:

```
@prefix veml:  <http://www.icsi.berkeley.edu/~snarayan/VEML.owl#>
@prefix verl:  <http://www.icsi.berkeley.edu/~snarayan/VERL.owl#>
@prefix data:  <http://www.icsi.berkeley.edu/~snarayan/meeting01.owl#>
...

# FROM veml: (T-BOX)
veml:sceneEvents rdfs:range veml:EventList .
veml:EventList rdfs:subClassOf _:r1 ; rdfs:subClassOf _:r2 .
_:r1 :allValuesFrom verl:Event ; :onProperty rdf:first .
_:r2 :allValuesFrom veml:EventList ; :onProperty rdf:rest .

# FROM data: (A-BOX)
data:scene veml:sceneEvents ( data:1 , ..., data:65 ) .

# EXAMPLE COLLECTION SNIPPET
_:cN rdf:first data:N ; rdf:rest _:cN+1 .
```

From the above data, each iteration of :allValuesFrom reasoning and subsequent subclass reasoning produced:

```
# INPUT TO ALL-VALUES-FROM, ITERATION 0
# FROM INPUT
(_:c1..._:c65) rdf:first (data:1 ...  data:65) .
# FROM RANGE
_:c1 a veml:EventList .

# OUTPUT ALL-VALUES-FROM, ITERATION N
_:dataN a verl:Event .
_:cN+1 a veml:EventList .

# FROM SUBCLASS ON ABOVE
# ADDED TO ALL-VALUES-FROM, ITERATION N+1
_:cN+1 rdf:type _:r1 ;  rdf:type _:r2 .
```

In particular, a small contribution of input statements requires a merge-sort and re-scan of the file in question. This could indeed be solved by implementing binary-search lookup functionality over the sorted files for small input from a previous round; however, this would break with our initial aim of performing reasoning using only the primitives of file-scanning and multi-way merge-sort.

Finally in the reasoning process, we must perform consolidation of the input data and the output inferred statements according to the :sameAs index produced in the previous step. The first step involves sorting the input and inferred data according to natural SPOC order; the process took 6.4 hours and rewrote 35.4m statements into pivotal form. The second step involves subsequent sorting of the data according to inverse OPSC order; the process took 8.2 hours and rewrote 8.5m statements. The expense of these steps is primarily attributable to applying multi-way merge-sorting over all data in both sorting orders.

Although the degradation of performance related to the on-disk fixpoint computation of ruleset $\mathcal{R}2 \cup \mathcal{R}3$ is significant, if one is prepared to trade completeness (as we define it) for computational efficiency, the fixpoint calculation can be restrained to only perform a small, known amount of iterations (e.g., inferencing of the majority of statements in Figure 4 takes place over approx. 3 hours). Only minute amounts of inferred statements are produced in latter iterations of the fixpoint.

Further still, most inferences are produced after the initial scan which takes approx. 79 minutes. Thus, even after application of only $\mathcal{R}0$ and $\mathcal{R}1$ rules, the majority of inferencing has been conducted. This simpler more practical reasoning subset exhibits linear scale, as is visible for the first stage of Figure 3 prior to the on-disk computations. Along these lines, we present in Figure 5 the performance of applying rules $\mathcal{R}0$ and $\mathcal{R}1$ to the 1.1b statement dataset, in one scan, with respect to the T-Box derived from that dataset as described above. In particular, we refer to the linear trend present; upon inspection, one can see that minor slow-down in the rate of statements read is attributable to an increased throughput in terms of output statements (disk write operations).

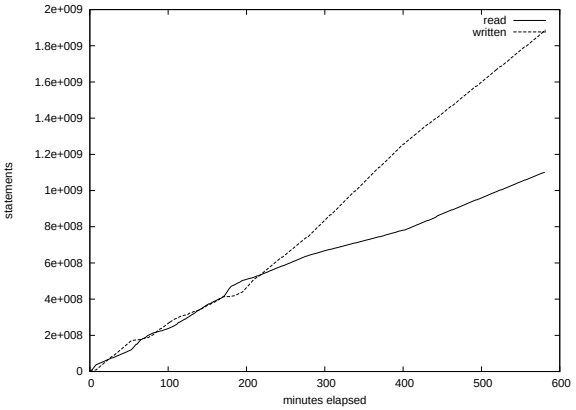

Figure 5: Performance of applying ruleset $\mathcal{R}0 \cup \mathcal{R}1$ on the 1.1b dataset

Finally, Table 7 lists the number of times each rule was fired for reasoning on the 1.1b dataset, reasoning using only $\mathcal{R}0 \cup \mathcal{R}1$ on the 147m dataset and also of applying all rules to the 147m dataset. Again, from both Figure 3 and Table 7 we can deduce that the bulk of current web reasoning is covered by those rules ($\mathcal{R}0 \cup \mathcal{R}1$) which exhibit linear scale.

6 Related Work

OWL reasoning, specifically query answering over OWL Full, is not tackled by typical DL Reasoners; such as FaCT++ [45], RACER [19] or Pellet [40]; which focus on complex reasoning tasks such as subsumption checking and provable completeness of reasoning. Likewise, KAON2 [32], which reports better results on query answering, is limited to OWL-DL expressivity due to completeness requirements. Despite being able to deal with complex ontologies in a complete manner, these systems are not tailored for the particular challenges of processing large amounts of RDF data and particularly large A-Boxes.

Systems such as TRIPLE [39], JESS[18], or Jena[19] support rule representable RDFS or OWL fragments as we do, but only work in-memory whereas our framework is focused on conducting scalable reasoning using persistent storage.

The OWLIM [28] family of systems allows reasoning over a version of pD* using the TRREE: Triple Reasoning and Rule Entailment Engine. Besides the in-memory version SwiftOWLIM, which uses TRREE, there is also a version offering query-processing over a persistent image of the repository, BigOWLIM, which comes closest technically to our approach. In evaluation on 2 x Dual-Core 2GHz machines with 16GB of RAM, BigOWLIM is claimed to index over 1 bn triples from the LUBM benchmark [17] in just under 70 hours [1]; however, this figure includes indexing of the data for query-answering, and is not directly comparable with our results, and in any case, our reasoning approach strictly focuses on sensible reasoning for web data.

Some existing systems already implement a separation of T-Box and A-Box for scalable reasoning, where in particular, assertional data are stored in some RDBMS; e.g. DLDB [35], Minerva [48] and OntoDB [25]. Similar to our approach of reasoning over web data, [36] demonstrates reasoning over 166m triples using the DLDB system. Also like us, (and as we had previously introduced in [23]) they internally choose pivot identifiers to represent equivalent sets of individuals. However, they use the notion of perspectives to support

[18] `http://herzberg.ca.sandia.gov/`
[19] `http://jena.sourceforge.net/`

Rule	1.1b - $\mathcal{R}0-1$	147M - $\mathcal{R}0-1$	147M - $\mathcal{R}0-3$
$\mathcal{R}0$			
rdfc0	35,157	6,084	6,084
$\mathcal{R}1$			
rdfs2	591,304,476	30,203,111	30,462,570
rdfs3′	596,661,696	31,789,905	32,048,477
rdfs7′	156,744,587	27,723,256	27,882,492
rdfs9	1,164,619,890	64,869,593	65,455,001
rdfp3′	562,426	483,204	483,204
rdfp8a′	231,661,554	9,404,319	9,556,544
rdfp8b′	231,658,162	9,404,111	9,556,336
rdfp12a′	8,153,304	23,869	38,060
rdfp12b′	57,116	17,769	25,362
rdfp13a′	5,667,464	11,478	11,478
rdfp13b′	6,642	4,350	4,350
rdfp14a′	98,601	39,422	39,902
rdfp14b′	104,780	43,886	44,390
rdfc1	15,198,615	1,492,395	1,595,293
rdfc2	584,913	337,141	337,279
rdfc3a	115,416	3,075	17,224
rdfc3b	54	8	8
$\mathcal{R}2$			
rdfp1′	-	-	31,174
rdfp2	-	-	2,097,007
rdfp4	-	-	291,048
rdfp15′	-	-	42,098
rdfp16′	-	-	685,738
rdfc3c	-	-	6,976
rdfc4a	-	-	211
rdfc4b	-	-	246

Table 7: Count of number of statements inferred for applying the given ruleset on the given dataset.

inferencing based on T-Box data; in their experiment they manually selected nine T-Box perspectives, unlike our approach that deals with arbitrary T-Box data from the Web. Their evaluation was performed on a workstation with dual 64-bit CPUs and 10GB main memory on which they loaded 760k documents / 166m triples (14% larger than our 147m statement dataset) in about 350 hrs; however, unlike our evaluation, the total time taken includes indexing for query-answering.

In a similar approach to our authoritative analysis, [8] introduced restrictions for accepting sub-class and equivalent-class statements from third-party sources; they follow similar arguments to that made in this paper. However, their notion of what we call authoritativeness is based on hostnames and does not consider redirects; we argue that in both cases, e.g., use of PURL services[20] is not properly supported: (i) all documents using the same service (and having the same namespace hostname) would be 'authoritative' for each other, (ii) the document cannot be served directly by the namespace location, but only through a redirect. Indeed, further work presented in [7] introduced the notion of an *authoritative description* which is very similar to ours. In any case, we provide much more extensive treatment of the issue, supporting a much more varied range of RDF(S)/OWL constructs.

One promising alternative to authoritative reasoning for the Web is the notion of "context-dependant" or "quarantined reasoning" introduced in [11], whereby inference results are only considered valid within the given context of a document. As opposed to our approach whereby we construct one authoritative model for

[20] `http://purl.org/`

all web data, their approach uses a unique model for each document, based on implicit and explicit imports of the document; thus, they would infer statements within the local context which we would consider to be non-authoritative. However, they would miss inferences which can only be conducted by considering a merge of documents, such as transitive closure or equality inferences based on inverse-functional properties over multiple documents. Their evaluation was completed on three machines with quad-core 2.33GHz and 8GB main memory; they claimed to be able to load, on average, 40 documents per second.

7 Conclusion and Future Work

We have presented SAOR: a system for performing reasoning over web data based on primitives known to scale: file-scan and sorting. We maintain a separate optimised T-Box index for our reasoning procedure. To keep the resulting knowledge-base manageable, both in size and quality, we made the following modifications to traditional reasoning procedures:

- only consider a positive fragment of OWL reasoning;
- analyse the authority of sources to counter ontology hijacking;
- use pivot identifiers instead of full materialisation of equality.

We show in our evaluation that naïve inferencing over web data leads to an explosion of materialised statements and show how to prevent this explosion through analysis of the authority of data sources. We also present metrics relating to the most productive rules with regards inferencing on the Web.

Although SAOR is currently not optimised for reaching full closure, we show that our system is suitable for optimised computation of the approximate closure of a web knowledge-base w.r.t. the most commonly used RDF(S) and OWL constructs. In our evaluation, we showed that the bulk of inferencing on web data can be completed with two scans of an unsorted web-crawl.

Future work includes investigating possible distribution methods: indeed, by limiting our tool-box to file scans and sorts, our system can be implemented on multiple machines, as-is, according to known distribution methods for our foundational operations.

References

[1] Bigowlim: System doc., Oct. 2006. `http://www.ontotext.com/owlim/big/BigOWLIMSysDoc.pdf`.

[2] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference. W3C Recommendation, Feb. 2004. `http://www.w3.org/TR/owl-ref/`.

[3] S. Bechhofer and R. Volz. Patching syntax in owl ontologies. In *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 668–682. Springer, November 2004.

[4] D. Brickley and R. Guha. Rdf vocabulary description language 1.0: Rdf schema. W3C Recommendation, Feb. 2004. `http://www.w3.org/TR/rdf-schema/`.

[5] D. Brickley and L. Miller. FOAF Vocabulary Specification 0.91, Nov. 2007. `http://xmlns.com/foaf/spec/`.

[6] J. d. Bruijn and S. Heymans. Logical foundations of (e)RDF(S): Complexity and reasoning. In *6th International Semantic Web Conference*, number 4825 in LNCS, pages 86–99, Busan, Korea, Nov 2007.

[7] G. Cheng, W. Ge, H. Wu, and Y. Qu. Searching semantic web objects based on class hierarchies. In *Proceedings of Linked Data on the Web Workshop*, 2008.

[8] G. Cheng and Y. Qu. Term dependence on the semantic web. In *International Semantic Web Conference*, pages 665–680, oct 2008.

[9] J. de Bruijn. *Semantic Web Language Layering with Ontologies, Rules, and Meta-Modeling*. PhD thesis, University of Innsbruck, 2008.

[10] J. de Bruijn, A. Polleres, R. Lara, and D. Fensel. OWL$^-$. Final draft d20.1v0.2, WSML, 2005.

[11] R. Delbru, A. Polleres, G. Tummarello, and S. Decker. Context dependent reasoning for semantic documents in Sindice. In *Proceedings of the 4th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2008)*, October 2008.

[12] D. Fensel and F. van Harmelen. Unifying reasoning and search to web scale. *IEEE Internet Computing*, 11(2):96, 94–95, 2007.

[13] S. Ghilardi, C. Lutz, and F. Wolter. Did i damage my ontology? a case for conservative extensions in description logics. In *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning*, pages 187–197, June 2006.

[14] B. C. Grau, I. Horrocks, B. Parsia, P. Patel-Schneider, and U. Sattler. Next steps for OWL. In *OWL: Experiences and Directions Workshop*, Nov. 2006.

[15] B. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *13th International Conference on World Wide Web*, 2004.

[16] R. V. Guha, R. McCool, and R. Fikes. Contexts for the semantic web. In *Third International Semantic Web Conference*, pages 32–46, November 2004.

[17] Y. Guo, Z. Pan, and J. Heflin. Lubm: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, 3(2-3):158–182, 2005.

[18] C. Gutiérrez, C. Hurtado, and A. O. Mendelzon. Foundations of Semantic Web Databases. In *23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Paris*, June 2004.

[19] V. Haarslev and R. Möller. Racer: A core inference engine for the semantic web. In *International Workshop on Evaluation of Ontology-based Tools*, 2003.

[20] A. Harth and S. Decker. Optimized index structures for querying rdf from the web. In *3rd Latin American Web Congress*, pages 71–80. IEEE Press, 2005.

[21] A. Harth, J. Umbrich, and S. Decker. Multicrawler: A pipelined architecture for crawling and indexing semantic web data. In *5th International Semantic Web Conference*, pages 258–271, 2006.

[22] P. Hayes. RDF Semantics. W3C Recommendation, Feb. 2004. `http://www.w3.org/TR/rdf-mt/`.

[23] A. Hogan, A. Harth, and S. Decker. Performing object consolidation on the semantic web data graph. In *1st I3 Workshop: Identity, Identifiers, Identification Workshop*, 2007.

[24] A. Hogan, A. Harth, and A. Polleres. SAOR: Authoritative Reasoning for the Web. In *Proceedings of the 3rd Asian Semantic Web Conference (ASWC 2008)*, Bankok, Thailand, Dec. 2008.

[25] D. Hondjack, G. Pierra, and L. Bellatreche. Ontodb: An ontology-based database for data intensive applications. In *Proceedings of the 12th International Conference on Database Systems for Advanced Applications*, pages 497–508, April 2007.

[26] I. Horrocks and P. F. Patel-Schneider. Reducing owl entailment to description logic satisfiability. *Journal of Web Semamtics*, 1(4):345–357, 2004.

[27] E. Jiménez-Ruiz, B. C. Grau, U. Sattler, T. Schneider, and R. B. Llavori. Safe and economic re-use of ontologies: A logic-based methodology and tool support. In *Proceedings of the 21st International Workshop on Description Logics (DL2008)*, May 2008.

[28] A. Kiryakov, D. Ognyanov, and D. Manov. Owlim - a pragmatic semantic repository for owl. In *Web Information Systems Engineering Workshops*, LNCS, pages 182–192, New York, USA, Nov 2005.

[29] D. Kunkle and G. Cooperman. Solving rubik's cube: disk is the new ram. *Communications of the ACM*, 51(4):31–33, 2008.

[30] J. W. Lloyd. *Foundations of Logic Programming (2nd edition)*. Springer-Verlag, 1987.

[31] C. Lutz, D. Walther, and F. Wolter. Conservative extensions in expressive description logics. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 453–458, January 2007.

[32] B. Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Forschungszentrum Informatik, Karlsruhe, Germany, 2006.

[33] B. Motik. On the properties of metamodeling in owl. *Journal of Logic and Computation*, 17(4):617–637, 2007.

[34] S. Muñoz, J. Pérez, and C. Gutiérrez. Minimal deductive systems for RDF. In *ESWC*, pages 53–67, 2007.

[35] Z. Pan and J. Heflin. Dldb: Extending relational databases to support semantic web queries. In *PSSS1 - Practical and Scalable Semantic Systems, Proceedings of the First International Workshop on Practical and Scalable Semantic Systems*, October 2003.

[36] Z. Pan, A. Qasem, S. Kanitkar, F. Prabhakar, and J. Heflin. Hawkeye: A practical large scale demonstration of semantic web integration. In *OTM Workshops (2)*, volume 4806 of *Lecture Notes in Computer Science*, pages 1115–1124. Springer, November 2007.

[37] P. F. Patel-Schneider and I. Horrocks. Owl web ontology language semantics and abstract syntax section 4. mapping to rdf graphs. W3C Recommendation, Feb. 2004. `http://www.w3.org/TR/owl-semantics/mapping.html`.

[38] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Recommendation, Jan. 2008. `http://www.w3.org/TR/rdf-sparql-query/`.

[39] M. Sintek and S. Decker. Triple - a query, inference, and transformation language for the semantic web. In *1st International Semantic Web Conference*, pages 364–378, 2002.

[40] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.

[41] M. K. Smith, C. Welty, and D. L. McGuinness. OWL Web Ontology Language Guide. W3C Recommendation, Feb. 2004. `http://www.w3.org/TR/owl-guide/`.

[42] H. J. ter Horst. Combining rdf and part of owl with rules: Semantics, decidability, complexity. In *4th International Semantic Web Conference*, pages 668–684, 2005.

[43] H. J. ter Horst. Completeness, decidability and complexity of entailment for rdf schema ans a semantic extension involving the owl vocabulary. *Journal of Web Semantics*, 3:79–115, 2005.

[44] Y. Theoharis, V. Christophides, and G. Karvounarakis. Benchmarking database representations of rdf/s stores. In *Proceedings of the Fourth International Semantic Web Conference*, pages 685–701, November 2005.

[45] D. Tsarkov and I. Horrocks. Fact++ description logic reasoner: System description. In *International Joint Conf. on Automated Reasoning*, pages 292–297, 2006.

[46] T. D. Wang, B. Parsia, and J. A. Hendler. A survey of the web ontology landscape. In *Proceedings of the 5th International Semantic Web Conference (ISWC 2006)*, pages 682–694, Athens, GA, USA, Nov. 2006.

[47] Z. Wu, G. Eadon, S. Das, E. I. Chong, V. Kolovski, M. Annamalai, and J. Srinivasan. Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules in Oracle. In *24th International Conference on Data Engineering*. IEEE, 2008. To appear.

[48] J. Zhou, L. Ma, Q. Liu, L. Zhang, Y. Yu, and Y. Pan. Minerva: A scalable owl ontology storage and inference system. In *Proceedings of The First Asian Semantic Web Conference (ASWC)*, pages 429–443, September 2006.