Robust and Scalable Linked Data Reasoning Incorporating Provenance and Trust Annotations

Piero A. Bonatti [a], Aidan Hogan [b], Axel Polleres [b], Luigi Sauro [a]

[a] *Università di Napoli "Federico II", Napoli, Italy*
[b] *Digital Enterprise Research Institute, National University of Ireland, Galway*

Abstract

In this paper, we leverage annotated logic programs for tracking indicators of provenance and trust during reasoning, specifically focussing on the use-case of applying a scalable subset of OWL 2 RL/RDF rules over static corpora of arbitrary Linked Data (Web data). Our annotations encode three facets of information: (i) *blacklist*: a (possibly manually generated) boolean annotation which indicates that the referent data are known to be harmful and should be ignored during reasoning; (ii) *ranking*: a numeric value derived by a PageRank-inspired technique—adapted for Linked Data—which determines the centrality of certain data artefacts (such as RDF documents and statements); (iii) *authority*: a boolean value which uses Linked Data principles to *conservatively* determine whether or not some terminological information can be trusted. We formalise a logical framework which annotates inferences with the *strength* of derivation along these dimensions of trust and provenance; we formally demonstrate some desirable properties of the deployment of annotated logic programming in our setting, which guarantees (i) a unique minimal model (least fixpoint); (ii) monotonicity; (iii) finitariness; and (iv) finally decidability. In so doing, we also give some formal results which reveal strategies for scalable and efficient implementation of various reasoning tasks one might consider. Thereafter, we discuss scalable and distributed implementation strategies for applying our ranking and reasoning methods over a cluster of commodity hardware; throughout, we provide evaluation of our methods over 1 billion Linked Data quadruples crawled from approximately 4 million individual Web documents, empirically demonstrating the scalability of our approach, and how our annotation values help ensure a more robust form of reasoning. We finally sketch, discuss and evaluate a use-case for a simple repair of inconsistencies detectable within OWL 2 RL/RDF constraint rules using ranking annotations to detect and defeat the "marginal view", and in so doing, infer an empirical "consistency threshold" for the Web of Data in our setting.

Key words: annotated programs; linked data; web reasoning; scalable reasoning; distributed reasoning; authoritative reasoning; owl 2 rl; provenance; pagerank; inconsistency; repair

1. Introduction

The Semantic Web is no longer purely academic: in particular, the Linking Open Data project has fostered a rich lode of openly available structured data on the Web, commonly dubbed the "Web of Data" [53]. Based on the Resource Description Framework (RDF), Linked Data emphasises four simple principles: (i) use URIs as names for things (and not just documents); (ii) make those URIs dereferenceable via HTTP; (iii) return useful and relevant RDF content upon lookup of those URIs; (iv) include links to other datasets. Since its inception four years ago, the Linked Data community has overseen exports from corporate bodies (e.g., the BBC, New York Times, Freebase, Linked Clinical Trials [`linkedct.org`], Thompson Reuters [`opencalais.com`] etc.), governmental bod-

Email addresses: `bonatti@na.infn.it` (Piero A. Bonatti), `aidan.hogan@deri.org` (Aidan Hogan), `axel.polleres@deri.org` (Axel Polleres), `sauro@na.infn.it` (Luigi Sauro).

ies (e.g., UK Government [data.gov.uk], US Government [data.gov], US National Science Foundation, etc.), community driven efforts (e.g., Wikipedia-based exports [dbpedia.org], GeoNames, etc.), social-networking sites (e.g., MySpace [dbtune.org], flickr, Twitter [semantictweet.com], etc.) and scientific communities (e.g., DBLP, PubMed, UniProt), amongst others. [1]

Interspersed with these voluminous exports of assertional (or instance) data are lightweight schemata/ontologies defined in RDFS/OWL—which we will collectively call *vocabularies*—comprising the terminological data which describe classes and properties and provide a formalisation of the domain of discourse. The assertional data describe *things* by assigning datatype (string) values for named attributes (datatype properties), named relations to other things (object properties), and named classifications (classes). The terminological data describe these classes and properties, with RDFS and OWL providing the formal mechanisms to render the semantics of these terms—in particular, their interrelation and prescribed intended usage. Importantly, best-practices encourage: (i) re-use of class and property terms by independent and remote data publishers; (ii) inter-vocabulary extension of classes and properties; (iii) dereferenceability of classes and properties, returning a formal RDFS/OWL description thereupon.

Applications are slowly emerging which leverage this rich vein of structured Web data; however, thus far (with of course a few exceptions) applications have been slow to leverage the underlying terminological data for *reasoning*. Loosely speaking, reasoning utilises the formal underlying semantics of the terms in the data to enable the derivation of new knowledge. With respect to Linked Data, sometimes there is only sparse re-use of (terminological and/or assertional) terms between sources, and so reasoning can be used to better integrate the data in a merged corpus as follows: (i) infer and support the semantics of ground equality (owl:sameAs relations between individuals) to unite knowledge fractured by insufficient URI re-use across the Web; (ii) leverage terminological knowledge to infer new assertional knowledge, possibly across vocabularies and even assertional documents; (iii) detect inconsistencies whereby one or more parties may provide conflicting data—herein, we focus on the latter two reasoning tasks.

[1] See http://richard.cyganiak.de/2007/10/lod/ for a comprehensive graph of such datasets and their inter-linkage; however—and as the Linked Open Numbers project (see [64, Figure 1]) has aptly evinced, and indeed as we will see ourselves in later evaluation—not all of this current Web of Data is entirely "compelling".

However, reasoning over (even subsets) of the Web of Data poses two major challenges, with serious implications for reasoning: (i) *scalability*, where one can expect Linked Data corpora containing in the order of billions or tens of billions of statements (for the moment); (ii) *tolerance to noise and inconsistency*, whereby data published on the Web are afflicted with naïve errors and disagreement, arbitrary redefinitions of classes/properties, etc. Traditional reasoning approaches are not well positioned to tackle such challenges, where the main body of literature in the area is focussed on considerations such as expressiveness, soundness, completeness and polynomial-time tractability [2], and makes some basic assumptions about the underlying data quality; for example, tableaux-based algorithms struggle with large bodies of assertional knowledge, and are tied by the principle of *ex falso quodlibet*—from contradiction follows anything—and thus struggle when reasoning over possibly inconsistent data. [3]

In previous works, we presented our Scalable Authoritative OWL Reasoner (SAOR) which applies a subset of OWL 2 RL/RDF rules over arbitrary Linked Data crawls: in particular, we abandon completeness in favour of conducting "sensible" inferencing which we argue to be suitable for the Linked Data use-case [36,38]. [4] We have previously demonstrated distributed reasoning over ∼1b Linked Data triples [38], and have also presented some preliminary formalisations of what we call "authoritative reasoning", which considers the source of terminological data during reasoning [36]. The SAOR system is actively used for materialising inferences in the Semantic Web Search Engine (SWSE) [37] which offers search and browsing over Linked Data. [5]

In this paper, we look to reformalise how the SAOR engine incorporates the notions of provenance and trust which form an integral part of its tolerance to noise, where we see the use of annotated logic programs [42] as a natural fit. We thus derive a formal logical framework for annotated reasoning in our setting, which encodes three indicators of provenance and trust and which thus allows us to apply robust materialisation over large, static, Linked Data corpora.

[2] In our scenario with assertional data from the Web in the order of billions of statements, even quadratic complexity is prohibitively expensive.

[3] There is ongoing research devoted to *paraconsistent logics* which tackle this issue—e.g., see a recent proposal for OWL 2 [51]—but the *efficiency* of such approaches is still an open question.

[4] For a high-level discussion on the general infeasibility of completeness for scenarios such as ours, see [31].

[5] http://swse.deri.org/

Further, in previous work we noted that many inconsistencies arise on the Web as the result of incompatible naming of resources, or naïve publishing errors [35]: herein, we look at a secondary use-case for our annotations which leverages OWL 2 RL/RDF *constraint rules* to detect inconsistencies, where subsequently we perform a simple "repair" of the Web knowledge-base using our annotations—particularly ranks—to identify and defeat the "marginal view" present in the inconsistency.

More specifically, we:
- introduce some necessary preliminaries (Section 2);
- discuss our proposed annotation values to represent provenance and trust—*blacklisting*, *authority*, and *ranking*—giving concrete instantiations for each (Section 3);
- describe a formal framework for annotated programs which incorporate the above three dimensions of provenance and trust, including formal discussion of *constraint rules* (Section 4);
- describe our experimental setup and our 1 billion triple Linked Data corpus (Section 5);
- describe our distributed (i) implementation and evaluation of links-based ranking (Section 6), (ii) annotated reasoning for a subset of OWL 2 RL/RDF rules (Section 7), and (iii) our inconsistency detection and repair use-case (Section 8);
- discuss issues relating to scalability and expressiveness (Section 9), render related work in the field (Section 10), and conclude (Section 11).

2. Preliminaries

In this section, we provide some necessary preliminaries relating to (i) RDF (Section 2.1); (ii) Linked Data principles and data sources (Section 2.2); (iii) rules and atoms (Section 2.3); (iv) generalised annotated programs (Section 2.4) (v) terminological data given by RDFS/OWL (Section 2.5); and (vi) OWL 2 RL/RDF rules (Section 2.6). We attempt to preserve notation and terminology as prevalent in the literature.

2.1. *RDF*

We briefly give some necessary notation relating to RDF constants and RDF triples; cf. [30].

RDF Constant Given the set of URI references $\mathbf{U}$, the set of blank nodes $\mathbf{B}$, [6] and the set of literals $\mathbf{L}$, the set of *RDF constants* is denoted by $\mathbf{C} \coloneqq \mathbf{U} \cup \mathbf{B} \cup \mathbf{L}$. We also define the set of variables $\mathbf{V}$ which range over $\mathbf{C}$.

Herein, we use CURIEs [5] to denote URIs. Following Turtle syntax [2], use `a` as a convenient shortcut for `rdf:type`. We denote variables with a '?' prefix.

RDF Triple A triple $t \coloneqq (s, p, o) \in (\mathbf{U} \cup \mathbf{B}) \times \mathbf{U} \times \mathbf{C}$ is called an *RDF triple*, where *s* is called subject, *p* predicate, and *o* object. A triple $t \coloneqq (s, p, o) \in \mathbf{G}, \mathbf{G} \coloneqq \mathbf{C} \times \mathbf{C} \times \mathbf{C}$ is called a *generalised triple* [25], which allows any RDF constant in any triple position: henceforth, we assume generalised triples unless explicitly stated otherwise. We call a finite set of triples $G \subset \mathbf{G}$ a *graph*.

2.2. *Linked Data principles, Data Sources and Quadruples*

In order to cope with the unique challenges of handling diverse and unverified Web data, many of our components and algorithms require inclusion of a notion of provenance: consideration of the source of RDF data found on the Web. Tightly related to such notions are the best practices of Linked Data [3], which give clear guidelines for publishing RDF on the Web. We briefly discuss Linked Data principles and notions relating to provenance. [7]

Linked Data Principles The four best practices of Linked Data are as follows [3]:
- (**LDP1**) use URIs to name things;
- (**LDP2**) use HTTP URIs so that those names can be looked up;
- (**LDP3**) provide useful structured information when a look-up on a URI is made – loosely, called *dereferencing*;
- (**LDP4**) include links using external URIs.

Data Source We define the *http-download* function $\mathsf{get} : \mathbf{U} \to 2^{\mathbf{G}}$ as the mapping from a URI to an RDF graph (set of facts) it may provide by means of a given HTTP lookup [21] which directly returns status code `200 OK` and data in a suitable RDF format; this function

[6] We interpret blank-nodes as skolem constants, as opposed to existential variables. Also, we rewrite blank-node labels to ensure uniqueness per document, as prescribed in [30].

[7] Note that in a practical sense, all HTTP-level functions $\{\mathsf{get}, \mathsf{redir}, \mathsf{redirs}, \mathsf{deref}\}$ are set at the time of the crawl, and are bounded by the knowledge of our crawl.

also performs a rewriting of blank-node labels (based on the input URI) to ensure uniqueness when merging RDF graphs [30]. We define the set of *data sources* $\mathbf{S} \subset \mathbf{U}$ as the set of URIs $\mathbf{S} := \{s \in \mathbf{U} \mid \mathsf{get}(s) \neq \emptyset\}$.

RDF Triple in Context/RDF Quadruple An ordered pair (t, c) with a triple $t = (s, p, o)$, $c \in \mathbf{S}$ and $t \in \mathsf{get}(c)$ is called a *triple in context* c. We may also refer to (s, p, o, c) as an *RDF quadruple* or quad q with context c.

HTTP Redirects/Dereferencing A URI may provide a HTTP redirect to another URI using a `30x` response code [21]; we denote this function as $\mathsf{redir} : \mathbf{U} \to \mathbf{U}$ which may map a URI to itself in the case of failure (e.g., where no redirect exists)—note that we do not need to distinguish between the different `30x` redirection schemes, and that this function would implicitly involve, e.g., stripping the fragment identifier of a URI [4]. We denote the fixpoint of redir as redirs, denoting traversal of a number of redirects (a limit may be set on this traversal to avoid cycles and artificially long redirect paths). We define *dereferencing* as the function $\mathsf{deref} := \mathsf{get} \circ \mathsf{redirs}$ which maps a URI to an RDF graph retrieved with status code `200 OK` *after* following redirects, or which maps a URI to the empty set in the case of failure.

2.3. *Atoms and Rules*

In this section, we briefly introduce some notation as familiar from the field of Logic Programming [48], which acts as a generalisation of the aforementioned RDF notation.

Atom Atoms are of the form $p(e_1, \ldots, e_n)$ where $e_1, \ldots, e_n$ are terms (like Datalog, function symbols are disallowed) and where p is a predicate of arity n—we denote the set of all such atoms by $\mathbf{Atoms}$. This is a generalisation of RDF triples, for which we employ a *ternary predicate* T where our atoms are of the form $T(s, p, o)$—for brevity, we commonly omit the ternary predicate and simply write (s, p, o). An RDF atom of this form is synonymous with a generalised triple pattern where variables of the set $\mathbf{V}$ are allowed in any position). A *ground atom*—or simply a *fact*—is one which does not contain variables (e.g., a generalised triple); we denote the set of all facts by $\mathbf{Facts}$—a generalisation of $\mathbf{G}$. A (Herbrand) *interpretation* I is a finite subset of $\mathbf{Facts}$—a generalisation of a graph.

Letting A and B be two atoms, we say that A *subsumes* B—denoted $A \triangleright B$—if there exists a substitution θ of variables such that $A\theta = B$ (applying θ to the variables of A yields B); we may also say that B is an *instance* of A; if B is ground, we say that it is a *ground instance*. Similarly, if we have a substitution θ such that $A\theta = B\theta$, we say that θ is a *unifier* of A and B; we denote by $\mathrm{mgu}(A, B)$ the *most general unifier* of A and B which provides the "minimal" variable substitution (up to variable renaming) required to unify A and B.

Rule A *rule* R is given as follows:

$$H \leftarrow B_1, \ldots, B_n (n \geq 0)$$

where $H, B_1, \ldots, B_n$ are atoms, H is called the *head* (conclusion/consequent) and $B_1, \ldots, B_n$ the *body* (premise/antecedent). We use $\mathsf{Head}(R)$ to denote the head H of R and $\mathsf{Body}(R)$ to denote the body $B_1, \ldots, B_n$ of R. Our rules are *range restricted* – or *safe* [60]: like Datalog, the variables appearing in the head of each rule must also appear in the body.

The set of all rules that can be defined over atoms using an (arbitrary but fixed) infinite supply of variables will be denoted by $\mathbf{Rules}$. A rule with an empty body is considered a fact; a rule with a non-empty body is called a *proper-rule*. We call a finite set of such rules a *program* P.

Like before, a ground rule is one without variables. We denote with $\mathsf{Ground}(R)$ the set of ground instantiations of a rule R and with $\mathsf{Ground}(P)$ the ground instantiations of all rules occurring in a program P.

Immediate Consequence Operator We give the (classical) immediate consequence operator $\mathfrak{C}_P$ of a program P under interpretation I as:

$$\mathfrak{C}_P : \quad 2^{\mathbf{Facts}} \to 2^{\mathbf{Facts}}$$
$$I \mapsto \{\mathsf{Head}(R)\theta \mid R \in P \text{ and } \exists I' \subseteq I \text{ s.t. } \theta = \mathrm{mgu}(\mathsf{Body}(R), I')\}$$

Intuitively, the immediate consequence operator maps from a set of facts I to the set of facts it *directly* entails with respect to the program P—note that $\mathfrak{C}_P(I)$ will retain the facts in P since facts are rules with empty bodies and thus unify with any interpretation, and note that for our purposes $\mathfrak{C}_P$ is *monotonic*—the addition of facts and rules to a program can only lead to a superset of consequences.

Since our rules are a syntactic subset of Datalog, $\mathfrak{C}_P$ has a *least fixpoint*—denoted $\mathrm{lfp}(\mathfrak{C}_P)$—whereby further application of $\mathfrak{C}_P$ will not yield any changes, and which can be calculated in a bottom-up fashion, starting from the empty interpretation Δ and applying iteratively $\mathfrak{C}_P$ [66] (here, convention assumes that P contains the set of input facts as well as proper rules). Define the iterations of $\mathfrak{C}_P$ as follows: $\mathfrak{C}_P \uparrow 0 := \Delta$; for

all ordinals α, $\mathfrak{C}_P \uparrow (\alpha+1) \coloneqq \mathfrak{C}_P(\mathfrak{C}_P \uparrow \alpha)$; since our rules are Datalog, there exists an α such that $\mathrm{lfp}(\mathfrak{C}_P) = \mathfrak{C}_P \uparrow \alpha$ for $\alpha < \omega$, where ω denotes the least infinite ordinal—i.e., the immediate consequence operator will reach a fixpoint in countable steps [61], thereby giving all ground consequences of the program. We call $\mathrm{lfp}(\mathfrak{C}_P)$ *the least model*, which is given the more succinct notation $\mathrm{lm}(P)$.

2.4. *Generalised annotated programs*

In *generalised annotated programs* [42] the set of truth values is generalised to an arbitrary upper semilattice $\mathcal{T}$, that may represent—say—fuzzy values, inconsistencies, validity intervals (i.e. time), or a confidence value, to name but a few.

(Generalised) Annotated rules Annotated rules are expressions like

$$H{:}\rho \leftarrow B_1{:}\mu_1, \ldots, B_n{:}\mu_n$$

where each μ_i can be either an element of $\mathcal{T}$ or a variable ranging over $\mathcal{T}$; ρ can be a function $f(\mu_1, \ldots, \mu_n)$ over $\mathcal{T}$. Programs, $\mathsf{Ground}(R)$ and $\mathsf{Ground}(P)$ are defined analogously to non-annotated programs.

Example: Consider the following simple example of a (generalised) annotated rule where $\mathcal{T}$ corresponds to a set of confidence values in the interval $[0, 1]$ of real numbers:

$$Father(?x){:}(0.5 \times \mu) \leftarrow Parent(?x){:}\mu\,. \quad (1)$$

This rule intuitively states that something is a $Father$ with (at least) half of the confidence for which it is a $Parent$.[8] ◇

Restricted interpretations So-called *restricted* interpretations map each ground atom to a member of $\mathcal{T}$. A restricted interpretation I satisfies $A{:}\mu$ (in symbols, $I \models A{:}\mu$) iff $I(A) \geq_{\mathcal{T}} \mu$, where $\geq_{\mathcal{T}}$ is $\mathcal{T}$'s ordering. Now I satisfies a rule like the above iff either I satisfies the head or I does not satisfy some of the annotated atoms in the body.

Example: Take the annotated rule from Equation 1, and let's say that we have a restricted interpretation I which satisfies $Parent(sam){:}0.6$. Now, for I to satisfy the given rule, it must also satisfy $Father(sam){:}0.3$ such that $I(Father(sam)) \geq 0.3$. ◇

[8] This example can be trivially converted to RDF by instead considering the ternary atoms (`?x`, `a`, `ex:Parent`) and (`?x`, `a`, `ex:Father`).

Restricted immediate consequences In the generalised annotation framework, the restricted immediate consequence operator of an annotated program P is defined as follows, where σ ranges over substitutions:

$$\mathfrak{R}_P(I)(A) \coloneqq \mathrm{lub}\big\{\rho \mid (A{:}\rho \leftarrow B_1{:}\mu_1, \ldots, B_n{:}\mu_n)\sigma \in \mathsf{Ground}(P) \text{ and } I \models (B_i{:}\mu_i)\sigma \text{ for } (1 \leq i \leq n)\big\}\,.$$

Example: Take a program P which comprises of the annotated rule shown in Equation 1 and the annotated fact $Parent(sam){:}0.6$. Then, $\mathfrak{R}_P(I)(Father(sam)) = 0.3$. Instead, let's say that P also contains $Father(sam){:}0.5$. Then $\mathfrak{R}_P(I)(Father(sam)) = \mathrm{lub}\{0.5, 0.3\} = 0.5$. ◇

Importantly, various properties of $\mathfrak{R}_P$ have been formally demonstrated for generalised annotated programs in [42], where we will reuse these results later for our own (more specialised) annotation framework in Section 4; for example, $\mathfrak{R}_P$ has been shown to be monotonic, but not always continuous [42].

2.5. *Terminological data: RDFS/OWL*

As previously described, RDFS/OWL allow for providing terminological data which constitute definitions of classes and properties used in the data. A detailed discussion of RDFS/OWL is out of scope, but the distinction of terminological and assertional data—which we now describe—is important for our purposes. First, we require some preliminaries.[9]

Meta-class We consider a *meta-class* as a class specifically of classes or properties; i.e., the members of a meta-class are themselves either classes or properties. Herein, *we restrict our notion of meta-classes to the set defined in RDF(S) and OWL specifications*, where examples include `rdf:Property`, `rdfs:Class`, `owl:Class`, `owl:Restriction`, `owl:DatatypeProperty`, `owl:FunctionalProperty`, etc. Note that `rdfs:Resource`, `rdfs:Literal`, e.g., are not meta-classes, since their members need not be classes or properties.

[9] As we are dealing with Web data, we refer to the OWL 2 Full language and the OWL 2 RDF-based semantics [18] unless explicitly stated otherwise. Note that a clean and intuitive definition of terminological data is somewhat difficult for RDFS and particularly OWL Full. We instead rely on a 'shibboleth' approach which identifies markers for what we consider to be RDFS/OWL terminological data.

Meta-property A *meta-property* is one which has a meta-class as its domain. Again, *we restrict our notion of meta-properties to the set defined in RDF(S) and OWL specifications*, where examples include `rdfs:domain`, `rdfs:subClassOf`, `owl:hasKey`, `owl:inverseOf`, `owl:oneOf`, `owl:onProperty`, `owl:unionOf`, etc. Note that `rdf:type`, `owl:sameAs`, `rdfs:label`, e.g., do *not* have a meta-class as domain, and are not considered meta-properties.

Terminological data We define the set of *terminological triples* as the union of the following sets of triples:

(i) triples with `rdf:type` as predicate and a meta-class as object;
(ii) triples with a meta-property as predicate;
(iii) triples forming a *valid* RDF list whose head is the object of a meta-property (e.g., a list used for `owl:unionOf`, `owl:intersectionOf`, etc.);
(iv) triples which contribute to an all-disjoint-classes or all-disjoint-properties axiom. [10]

Note that the last category of terminological data is only required for special consistency-checking rules called constraints: i.e., rules which check for logical contradictions in the data. For brevity, we leave this last category of terminological data implicit in the rest of the paper, where `owl:AllDisjointClasses` and `owl:AllDisjointProperties` can be thought of as "honorary meta-classes" included in category 1, `owl:members` can be thought of as an "honorary meta-property" included in category 2, and the respective RDF lists included in category 3.

Finally, we do not consider triples involving "user-defined" meta-classes or meta-properties as contributing to the terminology, where in the following example, the first triple is considered terminological, but the second triple is not. [11]

(`ex:inSubFamily`, `rdfs:subPropertyOf`, `rdfs:subClassOf`)
(`ex:Bos`, `ex:inSubFamily`, `ex:Bovinae`)

T-split rule A *T-split rule* R is given as follows:

$$H \leftarrow A_1, \ldots, A_n, T_1, \ldots, T_m \quad (n, m \geq 0) \qquad (2)$$

where the T_i, $0 \leq i \leq m$ atoms in the body (T-atoms) are all those that can *only* have terminological ground instances, whereas the A_i, $1 \leq i \leq n$ atoms (A-atoms), can have arbitrary ground instances. We use $\mathsf{TBody}(R)$ and $\mathsf{ABody}(R)$ to respectively denote the set of T-atoms and the set of A-atoms in the body of R. Herein, we presume that the T-atoms and A-atoms of our rules can be distinguished and referenced as defined above.

Example: Let R_{EX} denote the following rule

(`?x`, `a`, `?c2`) ← (`?c1`, `rdfs:subClassOf`, `?c2`), (`?x`, `a`, `?c1`)

When writing T-split rules, we denote $\mathsf{TBody}(R_{EX})$ by underlining: the underlined T-atom can only be bound by a triple with the meta-property `rdfs:subClassOf` as RDF predicate, and thus can only be bound by a terminological triple. The second atom in the body can be bound by assertional or terminological triples, and so is considered an A-atom. ◇

T-ground rule A *T-ground rule* is a set of *rule instances* for the T-split rule R given by grounding $\mathsf{TBody}(R)$. We denote the set of such rules for a program P and a set of facts I as $\mathsf{Ground}^T(P, I)$, defined as:

$$\mathsf{Ground}^T(P, I) := \big\{\mathsf{Head}(R)\theta \leftarrow \mathsf{ABody}(R)\theta \mid R \in P \text{ and } \exists I' \subseteq I \text{ s.t. } \theta = \mathrm{mgu}(\mathsf{TBody}(R), I')\big\}.$$

The result is a set of rules whose T-atoms are grounded by the terminological data in I.

Example: Consider the T-split rule R_{EX} from the previous example. Now let $I_{EX} := \{$ (`foaf:Person`, `rdfs:subClassOf`, `foaf:Agent`), (`foaf:Agent`, `rdfs:subClassOf`, `dc:Agent`) $\}$. Here, $\mathsf{Ground}^T(\{R_{EX}\}, I_{EX}) = \{$ (`?x`, `a`, `foaf:Agent`) ← (`?x`, `a`, `?foaf:Person`); (`?x`, `a`, `dc:Agent`) ← (`?x`, `a`, `?foaf:Agent`) $\}$. ◇

T-split program and least fixpoint Herein, we give an overview of the computation of the *T-split least fixpoint* for a program P, which is broken up into two parts: (i) the terminological least fixpoint, and (ii) the assertional least fixpoint. Let $P^F := \{R \in P \mid \mathsf{Body}(R) = \emptyset\}$ be the set of facts in P, [12] let $P^{T\emptyset} := \{R \in P \mid \mathsf{TBody}(R) \neq \emptyset, \mathsf{ABody}(R) = \emptyset\}$, let $P^{\emptyset A} := \{R \in P \mid \mathsf{TBody}(R) = \emptyset, \mathsf{ABody}(R) \neq \emptyset\}$, and let $P^{TA} := \{R \in P \mid \mathsf{TBody}(R) \neq \emptyset, \mathsf{ABody}(R) \neq \emptyset\}$. Clearly, $P = P^F \cup P^{T\emptyset} \cup P^{\emptyset A} \cup P^{TA}$. Now,

[10] That is, triples with `rdf:type` as predicate and `owl:AllDisjointClasses` or `owl:AllDisjointProperties` as object, triples whose predicate is `owl:members` and whose subject unifies with the previous category of triples, and triples forming a valid RDF list whose head unifies with the object of such an `owl:members` triple.

[11] In particular, we require a data-independent method for distinguishing terminological data from purely assertional data, such that we only allow those meta-classes/-properties which are known a-priori.

[12] Of course, P^F can refer to axiomatic facts and/or the initial facts given by an input knowledge-base.

let $TP := P^F \cup P^T$ denote the initial (terminological) program containing ground facts and T-atom only rules, and let $\mathrm{lm}(TP)$ denote the least model for the terminological program. Now, let $AP := \mathrm{lm}(TP) \cup P^{\emptyset A} \cup \mathsf{Ground}^T(P^{TA}, \mathrm{lm}(TP))$ denote the second (assertional) program containing all available facts and rules with empty or grounded T-atoms. Now, we can give the *least model of the T-split program* P as $\mathrm{lm}(AP)$ for AP derived from P as above—we more generally denote this by $\mathrm{lm}^T(P)$.

In [38], we showed that the T-split least fixpoint is complete wrt. the standard variant (given that our rules are monotonic) and that the T-split least fixpoint is complete with respect to the standard fixpoint if rules requiring assertional knowledge do not infer unique terminological knowledge required by the T-split program (i.e., the assertional program AP does not generate new terminological facts not available to the initial program TP).

2.6. *OWL 2 RL/RDF rules*

OWL 2 RL/RDF [25] rules are a partial axiomatisation of the OWL 2 RDF-Based Semantics which is applicable for arbitrary RDF graphs, and thus is compatible with RDF Semantics [30]. The atoms of these rules comprise primarily of ternary predicates encoding generalised RDF triples; some rules have a special head denoted `false` which indicate that an instance of the body is inconsistent. All such rules can be considered T-split where we use the aforementioned criteria for characterising terminological data and subsequently T-atoms.

As we will further discuss in Section 7, full materialisation wrt. the entire set of OWL 2 RL/RDF is infeasible in our use-case; in particular, given a large A-Box of arbitrary content, we wish to select a subset of the OWL 2 RL/RDF profile which is linear with respect to that A-Box; thus, we select a subset $\mathcal{O}2\mathcal{R}^-$ of OWL 2 RL/RDF rules where $|\mathsf{ABody}(R)| \leq 1$ for all $R \in \mathcal{O}2\mathcal{R}^-$ [36]—we provide the full ruleset in Appendix A. Besides ensuring that the growth of assertional inferences is linear wrt. the A-Box—and as we will see in later sections—our linear profile allows for near-trivial distribution of reasoning, as well as various other optimisation techniques (see [38,32]).

With respect to OWL 2 RL/RDF, in [32] we showed that the T-split least fixpoint is complete assuming (i) no *non-standard usage*, whereby `rdf:type`, `rdf:first`, `rdf:rest` and the RDFS/OWL meta-properties do not appear other than as a predicate in the data, and RDFS/OWL meta-classes do not appear in a position other than as the value for `rdf:type`; *and* (ii) that `owl:sameAs` does not affect constants in the terminology. [13]

3. Annotation values

Naïvely conducting materialisation wrt. non-arbitrary rules over arbitrary, non-verified data merged from millions of sources crawled from the Web broaches many obvious dangers. In this section, we discuss the annotation values we have chosen to represent the various dimensions of provenance and trust we use for reasoning over Linked Data, which have been informed by our past experiences in reasoning over arbitrary Linked Data.

3.1. *Blacklisting*

Despite our efforts to create algorithms which automatically detect and mitigate noise in the input data, it may often be desirable to *blacklist* input data based on some criteria: for example, data from a certain domain may be considered likely to be spam, or certain triple patterns may constitute common publishing errors which hinder the reasoning process. We currently do not require the blacklisting function, and thus consider all triples to be *not blacklisted*. However, such an annotation has obvious uses for bypassing noise which cannot otherwise be automatically detected.

One particular use-case we have in mind for including the blacklisting annotation relates to the publication of void values for inverse-functional properties: the Friend Of A Friend (FOAF) [14] vocabulary offers classes and properties for describing information about people, organisations, documents, and so forth: it is currently one of the most widely instantiated vocabularies in Linked Data [37, Appendix A]. FOAF includes a number of inverse-functional-properties which allow for identifying (esp.) people in the absence of agreement upon URIs, e.g.: `foaf:homepage`, `foaf:mbox`, `foaf:mbox_sha1sum`. However, FOAF exporters on the Web commonly do not respect the inverse-functional semantics of these properties; one particularly pathogenic case we encountered was exporters producing empty

[13] Note that the OWL 2 RL/RDF eq-rep-* rules can cause incompleteness by condition (ii), but herein, we do not support these rules. Further, note that in other profiles (such as RDFS and pD*) axiomatic triples may be considered non-standard—however, none of the OWL 2 RL/RDF axiomatic triples (see Table A.1) are non-standard.

[14] `http://xmlns.com/foaf/0.1/`

strings or values such as '`mailto:`' for `foaf:mbox` when users omitted specifying their email. Similarly, we encountered many corresponding erroneous values for the `foaf:mbox_sha1sum` property—representing a SHA1 encoded email value—referring to the SHA1 hashes of '`mailto:`' and the empty string [34]. These values—-caused by naïve publishing errors—lead to quadratic spurious inferences equating all users who omitted an email to each other. Although for reasons of efficiency we currently do not support the relevant OWL 2 RL/RDF rules which support the semantics of inverse-functional properties, the blacklisting annotation could be used to negate the effects of such pathogenic values—essentially, it serves as a pragmatic *last resort*.

3.2. *Authoritative analysis*

In our initial works on SAOR [36]—a pragmatic reasoner for Linked Data—we encountered a puzzling deluge of inferences which we did not initially expect. We found that remote documents sometimes cross-define terms resident in popular vocabularies, changing the inferences *authoritatively* mandated for those terms. For example, we found one document [15] which defines `owl:Thing` to be a member of 55 union classes—thus, materialisation wrt. OWL 2 RL/RDF rule cls-uni [25, Table 6] over any member of `owl:Thing` would infer 55 additional memberships for these obsolete and obscure union classes. We found another document [16] which defines nine *properties* as the domain of `rdf:type`—again, anything defined to be a member of any class would be inferred to be a member of these nine *properties*. Even aside from "cross-defining" core terms, popular vocabularies such as FOAF were also affected [36]—such practice lead to the materialisation of an impractical bulk of arguably irrelevant data (which would subsequently burden the consumer application).

To counter-act remote contributions about the semantics of terms, we introduced a more conservative form of reasoning called *authoritative reasoning* [36] which critically examines the source of terminological knowledge. We now re-introduce the concept of authoritative reasoning from [36], herein providing more detailed formalisms and updated discussion.

Our authoritative reasoning methods are based on the intuition that a publisher instantiating a vocabulary's term (class/property) thereby accepts the inferencing mandated by that vocabulary and recursively referenced vocabularies for that term. Thus, once a publisher instantiates a class or property from a vocabulary, only that vocabulary and its references should influence what inferences are possible through that instantiation.

Firstly, we must define the relationship between a class/property term and a vocabulary, and give the notion of *term-level authority*. We view a term as an RDF constant, and a vocabulary as a Web document. From Section 2.2, we recall the get mapping from a URI (a Web location) to an RDF graph it may provide by means of a given HTTP lookup and the redirs mapping for following the HTTP redirects of a URI. Also, let $\mathsf{bnodes}(G)$ denote the set of blank nodes appearing in the RDF graph G. Now, we denote a mapping from a source URI to the set of terms it speaks authoritatively for as follows: [17]

$$\begin{aligned}&\mathsf{auth} : \mathbf{S} \to 2^{\mathbf{C}}\\ &\qquad s \mapsto \{c \in \mathbf{U} \mid \mathsf{redirs}(c) = s\} \cup \mathsf{bnodes}(\mathsf{get}(s))\end{aligned}$$

where a Web source is authoritative for URIs which dereference to it and the blank nodes it contains; e.g., the FOAF vocabulary is authoritative for terms in its namespace since it follows best-practices and makes its class/property URIs dereference to an RDF/XML document defining the terms. Note that no document is authoritative for literals.

To negate the effects of non-authoritative terminological axioms on reasoning over Web data (as exemplified above), we add an extra condition to the T-grounding of a rule: in particular, we only require amendment to rules where both $\mathsf{TBody}(R) \neq \emptyset$ and $\mathsf{ABody}(R) \neq \emptyset$.

Authoritative T-ground rule Let $\mathsf{vars}^{TA}(R) \subset \mathbf{V}$ denote the set of variables appearing in both $\mathsf{TBody}(R)$ and $\mathsf{ABody}(R)$. Now, we define the set of *authoritative rule instances* for a program P, RDF graph G, and source s as: [18]

$$\begin{aligned}\widehat{\mathsf{Ground}}^{T}(P, G, s) \coloneqq \{&\mathsf{Head}(R)\theta \leftarrow \mathsf{ABody}(R)\theta \mid\\ &R \in P\\ &\text{and } \exists G' \subseteq G \text{ s.t. } \theta = \mathrm{mgu}(\mathsf{TBody}(R), G')\\ &\text{and if } \mathsf{TBody}(R) \neq \emptyset \wedge \mathsf{ABody}(R) \neq \emptyset\\ &\text{then } \exists v \in \mathsf{vars}^{TA}(R) \text{ s.t. } \theta(v) \in \mathsf{auth}(s)\}\end{aligned}$$

[15] `http://lsdis.cs.uga.edu/~oldham/ontology/wsag/wsag.owl`

[16] `http://www.eiao.net/rdf/1.0`

[17] Even pre-dating Linked Data, dereferenceable vocabulary terms were encouraged; cf. `http://www.w3.org/TR/2006/WD-swbp-vocab-pub-20060314/`

[18] Here we favour RDF graph notation as authority applies only in the context of Linked Data (but could be trivially generalised through the auth function).

where authoritative rule instances are synonymous with *authoritatively T-ground rules* and where the notion of authoritative rule instances for a program follows naturally. The additional condition for authoritativeness states that if a rule contains both T-atoms and A-atoms in the body ($\mathsf{ABody}(R) \neq \emptyset \wedge \mathsf{TBody}(R) \neq \emptyset$), then the unifier must substitute at least one variable appearing in both $\mathsf{ABody}(R)$ and $\mathsf{TBody}(R)$ (a variable from the set $\mathsf{vars}^{TA}(R)$) for an authoritative term from source s (a constant from the set $\mathsf{auth}(s)$) for the resulting T-ground rule to be authoritative. This implies that the source s must speak authoritatively for at least one term that will appear in the body of each proper T-ground rule which its terminology generates, and so cannot create new assertional rules which could apply over arbitrary assertional data not mentioning any of its terms. We illustrate this with an example.

Example: Take the T-split rule R_{EX} as before where $\mathsf{vars}^{TA}(R_{EX}) = \{\texttt{?c1}\}$ representing the set of variables in both $\mathsf{TBody}(R_{EX})$ and $\mathsf{ABody}(R_{EX})$. Let I_{EX} be the graph from source s, where now for each substitution θ, there must exist $v \in \mathsf{vars}^{TA}(R_{EX})$ such that s speaks authoritatively for $\theta(v)$. In this case,

- s must speak authoritatively for the URI `foaf:Person`—for which `?c1` is substituted—for the T-ground rule (`?x`, `a`, `foaf:Agent`) ← (`?x`, `a`, `?foaf:Person`) to be authoritative,
- analogously, s must speak authoritatively for the URI `foaf:Agent`—again for which `?c1` is substituted—for the T-ground rule (`?x`, `a`, `dc:Agent`) ← (`?x`, `a`, `foaf:Agent`) to be authoritative.

In other words, the source s serving the T-facts in I_{EX} must be the FOAF vocabulary for the above rules to authoritative. ◇

For reference, we highlight variables in $\mathsf{vars}^{TA}(R)$ with boldface in Table A.4.

(It is worth noting that for rules where $\mathsf{ABody}(R)$ and $\mathsf{TBody}(R)$ are both non-empty, authoritative instantiation of the rule will only consider unifiers for $\mathsf{TBody}(R)$ which come from one source: however, in practice for OWL 2 RL/RDF this is not so restrictive: although $\mathsf{TBody}(R)$ may contain multiple atoms, in such rules $\mathsf{TBody}(R)$ usually refers to an atomic axiom which requires multiple triples to represent—indeed, the OWL 2 Structural Specification [19] enforces usage of blank-nodes and cardinalities on such constructs to ensure that the constituent triples of the multi-triple axiom appear in one source. To take an example, for the T-atoms (`?x`, `owl:hasValue`, `?y`), (`?x`, `owl:onProperty`, `?p`), we would expect `?x` to be ground by a blank-node skolem and thus expect the instance to come from one graph.)

[19] http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/

3.3. *Links-based ranking*

There is a long history of links-based analysis over Web data—and in particular over hypertext documents—where links are seen as a positive vote for the relevance or importance of a given document. Seminal works exploiting the link structure of the Web for ranking documents include HITS [44] and PageRank [8]. Various approaches (e.g., [1,16,33,23,15,29]) look at incorporating links-based analysis techniques for ranking RDF data, with various end-goals in mind, most commonly, prioritisation of informational artefacts in user result-views; however, such analyses have been applied to other use-cases, including work by Guéret et al. [28] which uses betweenness centrality measures to identify potentially weak points in the Web of Data in terms of maintaining connectedness integrity.

Herein, we employ links-based analysis with the underlying premise that higher ranked sources contribute more "trustworthy" data: in our case, we would expect a correlation between the (Eigenvector) centrality of a source in the graph, and the quality of data that it provides. Inspired in particular by the work of Harth et al. [29] on applying PageRank to RDF, we implement a two-step process: (i) we create the graph of links between sources, and apply a standard PageRank calculation over said graph to derive source ranks; (ii) we propagate source ranks to the triples they contain using a simple summation aggregation. We now discuss these two steps in more detail.

3.3.1. *Creating and ranking the source graph*

Creating the graph of interlinking Linked Data sources is non-trivial, in that the notion of a hyperlink does not directly exist. Thus, we must extract a graph sympathetic to Linked Data principles and current publishing patterns.

Recalling the Linked Data principles enumerated in Section 2.2, according to **LDP4**, links should be specified simply by using external URI names in the data. These URI names should dereference to an RDF description of themselves according to **LDP2** and **LDP3** respectively. Let $D := (V, E)$ represent a simple directed graph where $V \subset \mathbf{S}$ is a set of sources (vertices), and $E \subset \mathbf{S} \times \mathbf{S}$ is a set of pairs of vertices (edges). Letting $s_i, s_j \in V$ be two vertices, then $(s_i, s_j) \in E$ iff $s_i \neq s_j$ and there exists some $u \in \mathbf{U}$ such that $\mathsf{redirs}(u) = s_j$

and u appears in some triple $t \in \mathsf{get}(s_i)$: i.e., an edge extends from s_i to s_j iff the RDF graph returned by s_i mentions a URI which redirects to s_j.

Now, let $E(s)$ denote the set of direct successors of s (outlinks), let $E_\emptyset$ denote the set of vertices with no outlinks (dangling nodes), and let $E^-(s)$ denote the set of direct predecessors of s (inlinks). The PageRank of a vertex s_i in the directed graph $D := (V, E)$ is then given as follows [8]:

$$r(s_i) := \frac{1-d}{|V|} + d \sum_{s_\emptyset \in E_\emptyset} \frac{r(s_\emptyset)}{|V|} + d \sum_{s_j \in E^-(s_i)} \frac{r(s_j)}{|E(s_j)|}$$

where d is a damping constant (usually set to 0.85) which helps ensure convergence in the following iterative calculation, and where the middle component splits the ranks of dangling nodes evenly across all other nodes. Note also that the first and second components are independent of i, and constitute the minimum possible rank of all nodes (ensures that ranks do not need to be normalised during iterative calculation).

Now let $w := \frac{1-d}{|V|}$ represent the weight of a universal (weak link) given by all non-dangling nodes to all other nodes—dangling nodes split their vote evenly and thus don't require a weak link; we can use a weighted adjacency matrix M as follows to encode the graph $D := (V, E)$:

$$m_{i,j} := \begin{cases} \frac{d}{|E(s_j)|} + w, & \text{if } (s_j, s_i) \in E \\ \frac{1}{|V|}, & \text{if } s_j \in E_\emptyset \\ w, & \text{otherwise} \end{cases}$$

where this stochastic matrix can be thought of as a Markov chain (dubbed the random-surfer model). The ranks of all sources can be expressed algebraically as the principal eigenvector of M, which in turn can be estimated using the power iteration method up until some termination criteria (fixed number of iterations, convergence measures, etc.) is reached. We refer the interested reader to [8] for more detail.

3.3.2. *Calculating triple ranks*

Based on the rank values for the data sources, we now calculate the ranks for individual triples. We use a simple model for ranking triples, based on the intuition that triples appearing in highly ranked sources should benefit from that rank, and that each additional source stating a triple should increase the rank of the triple. [20] Thus, for calculating the rank of a triple t, we use the summation of the ranks of the sources it appears in as follows:

$$r(t) := \sum_{s_t \in \{s \in \mathbf{S} | t \in \mathsf{get}(s)\}} r(s_t)$$

4. The logical framework

In this section, we look at incorporating the above three dimensions of trust and provenance—which we will herein refer to as annotation properties—into an annotated logic programming framework which tracks this information during reasoning, and determines the annotations of inferences based on the annotations of the rule and the relevant instances, where the resultant values of the annotation properties can be viewed as denoting the strength of a derivation. We propose and formalise a general annotation framework, introduce some high-level tasks it enables, and discuss issues relating to scalability in our scenario.

We begin in Section 4.1 by formalising *annotation functions* which abstract the mechanisms used for annotating facts and rules. In Section 4.2, we propose an annotated program framework and associated semantics based on the previous work of Kifer et al. [42] (introduced previously in Section 2.4). In Section 4.3, we introduce some high-level reasoning tasks that this framework enables; in Section 4.4 we look at how each task scales in the general case, and in Section 4.5 we focus on the scalability of the task required for our use-case over our selected annotation properties. In Section 4.6, we briefly discuss some alternative semantics that one might consider for our framework. We wrap up in Section 4.7 by introducing *annotated constraint rules* which allow for detecting and labelling inconsistencies in our use-case.

4.1. *Abstracting annotation functions*

The first step towards a formal semantics of annotated logic programs consists in generalising the annotations (such as blacklisting, authoritativeness, and ranking) recalled in the previous sections. The annotation domains

[20] Note that one could imagine a spamming scheme where a large number of spurious low-ranked documents repeatedly make the same assertions to create a set of highly-ranked triples. In future, we may revise this algorithm to take into account some limiting function derived from PLD-level analysis.

are abstracted by an arbitrary finite set of ordered domains $D_1, \ldots, D_z$:

Definition 1 *An* annotation domain *is a Cartesian product* $\mathbf{D} := \times_{i=1}^{z} D_i$ *where each* D_i *is totally ordered by a relation* $\leq_i$*. Each* D_i *has a* $\leq_i$*-maximal element* $\top_i$*. Define a partial order* $\leq$ *on* $\mathbf{D}$ *as the direct product of the orderings* $\leq_i$*, that is* $\langle d_1, \ldots, d_z\rangle \leq \langle d'_1, \ldots, d'_z\rangle$ *if for all* $1 \leq i \leq z$, $d_i \leq_i d'_i$*. When* $\langle d_1, \ldots, d_z\rangle < \langle d'_1, \ldots, d'_z\rangle$ *we say that* $\langle d'_1, \ldots, d'_z\rangle$ dominates $\langle d_1, \ldots, d_z\rangle$.

We denote with $\mathrm{lub}(\mathbf{D}')$ *and* $\mathrm{glb}(\mathbf{D}')$ *respectively the least upper bound and the greatest lower bound of a subset* $\mathbf{D}' \subseteq \mathbf{D}$.

In the examples introduced so far, based on blacklisting, authoritativeness, and ranking, $z = 3$ and $D_1 = \{\mathbf{b}, \mathbf{nb}\}$ (**b**=blacklisted, **nb**=non-blacklisted), $D_2 = \{\mathbf{na}, \mathbf{a}\}$ (**a**=authoritative, **na**=non-authoritative), $D_3 = \mathbb{R}$. Moreover, $\mathbf{b} \leq_\mathbf{1} \mathbf{nb}$, $\mathbf{na} \leq_\mathbf{2} \mathbf{a}$, and $x \leq_\mathbf{3} y$ iff $x \leq y$.

Now there are several kinds of annotations for the facts and rules mentioned so far:

(i) Some—like blacklisting—depend on structural properties of facts (e.g., e-mail addresses set to empty strings). Such annotations can be modelled as functions $f : \mathbf{Facts} \to D_i$, for some i.

(ii) Some depend only on the source context, like page ranking; all of the facts in $\mathsf{get}(s)$ inherit the ranking assigned to the source s. Such annotations can be modelled as functions $f : \mathbf{S} \to D_i$, for some i.[21]

(iii) Some—like authoritativeness—apply to profile rules, that are sound (meta)axiomatisations of OWL 2/RDF-based semantics. In this case, the quality and reliability of the rules as such is not questioned. In fact, the ranking applies (indirectly) to the T-atoms unified with the body and their semantic consequences (i.e., to specific rule applications). Accordingly, different rule instances are given different rankings based on the provenance of the facts unified with the rule body and the corresponding values of the variables. Roughly speaking, such annotations can be modelled as $f : \mathbf{Rules} \times \mathbf{S} \to D_i$ where **Rules** typically corresponds to the $\mathcal{O}2\mathcal{R}^-$ ruleset as discussed in Section 2.6.

The first two kinds of annotation functions can be generalised by abstract annotation functions

$$\alpha_i : \mathbf{Facts} \times \mathbf{S} \to D_i\,.$$

[21] Strictly speaking, ranking also depends also on the interlinkage occurring between sources; these details are left implicit in our framework.

We assume that $\alpha_i(F, s)$ is defined for all $F \in \mathsf{get}(s)$. *Furthermore we assume without loss of generality that for some index* z' $(0 \leq z' \leq z)$*,* $D_1, \ldots, D_{z'}$ *are associated to annotation functions of this type.*

Annotations of type 3 can be formalised as

$$\alpha_i : \mathbf{Rules} \times (\mathbf{S} \to 2^{\mathsf{Facts}}) \to D_i\,.$$

We assume that such α_i are defined over $\mathcal{O}2\mathcal{R}^- \times \mathsf{get}$. *Moreover, we assume (without loss of generality) that* $D_{z'+1}, \ldots, D_z$ *are associated to annotation functions of this type.*

Now given the annotation functions $\alpha_1, \ldots, \alpha_z$, the corresponding *annotated program* is:

$$\begin{aligned}
\mathsf{AnnP}(\mathcal{O}2\mathcal{R}^-, \mathsf{get}) := \bigcup_{s\in\mathbf{S}} \big\{ & F{:}\langle d_1, \ldots, d_z\rangle \mid F \in \mathsf{get}(s),\\
& d_i := \alpha_i(F, s) \quad (1 \leq i \leq z')\\
& d_i := \top_i \qquad\quad (z' < i \leq z) \big\}\\
\cup\\
\big\{ & R{:}\langle d_1, \ldots, d_z\rangle \mid R \in \mathsf{Ground}^T(\mathcal{O}2\mathcal{R}^-, \mathsf{get}),\\
& d_i := \top_i \qquad\quad (1 \leq i \leq z')\\
& d_i := \alpha_i(R, \mathsf{get}) \quad (z' < i \leq z) \big\}\,.
\end{aligned}$$

The formal semantics of $\mathsf{AnnP}(\mathcal{O}2\mathcal{R}^-, \mathsf{get})$ will be specified in the next section.

4.2. *Annotated program semantics*

According to the nature of $\mathsf{AnnP}(\mathcal{O}2\mathcal{R}^-, \mathsf{get})$, we define our annotated programs as follows:

Definition 2 (Programs) *A* program P *is a finite set of annotated rules*

$$H \leftarrow B_1, \ldots, B_m : \mathbf{d} \qquad (m \geq 0)$$

where $H, B_1, \ldots, B_m$ *are logical atoms and* $\mathbf{d} \in \mathbf{D}$*. When* $m = 0$*, a rule is called a fact and denoted by* $H{:}\mathbf{d}$ *(omitting the arrow).*

In the above definition, we abstract away the details of rule distribution across multiple sources; provenance is only reflected in rule annotations. In this section, we need no a-priori restriction on the set of predicates (although we may use the RDF ternary predicate in examples).

Note that our notion of an annotated program is a specialisation of the generalised annotated programs introduced in Section 2.4, where our notion of an annotated rule (or fact) comprises of a classical rule (or fact) and a ground annotation value.

Now we can define the models of our programs. Following the examples introduced in the previous sections,

the semantics of an atom A is a set of annotations, covering the possible ways of deriving A. Roughly speaking, the annotations of A include the minimum rankings of the facts and rule used to derive A.

Definition 3 (Interpretations) *Let $\mathcal{B}_P$ be the Herbrand base of a program P. An* interpretation *is a mapping $I : \mathcal{B}_P \to 2^{\mathbf{D}}$ that associates each fact $F \in \mathcal{B}_P$ with a set of possible annotations.*

Given a ground rule $R \coloneqq F \leftarrow B_1, \ldots, B_m : \mathbf{d}$ an interpretation I satisfies R if for all $\mathbf{d}_i \in I(B_i)$ $(1 \leq i \leq m)$, $\mathrm{glb}(\{\mathbf{d}_1, \ldots, \mathbf{d}_m, \mathbf{d}\}) \in I(F)$.

More generally, I satisfies a (possibly non-ground) rule R (in symbols, $I \models R$) iff I satisfies all of the ground rules in $\mathsf{Ground}(R)$. Accordingly, I is a model of a program P ($I \models P$) iff for all $R \in P$, $I \models R$. Finally, we say that the fact $F{:}\mathbf{d}$ is a logical consequence of P iff for all interpretation I, $I \models P$ implies $I \models F{:}\mathbf{d}$.

Example: Consider the following program, where annotations come from our use-case domain:

$$\begin{aligned} A &: \langle \mathbf{b}, \mathbf{a}, 0.4 \rangle \\ B &: \langle \mathbf{nb}, \mathbf{a}, 0.7 \rangle \\ C &: \langle \mathbf{nb}, \mathbf{a}, 0.6 \rangle \\ A \leftarrow B, C &: \langle \mathbf{nb}, \mathbf{na}, 1 \rangle \end{aligned}$$

The atom A can be derived directly by the fact $A{:}\langle \mathbf{b}, \mathbf{a}, 0.4 \rangle$ or using the rule $A \leftarrow B, C{:}\langle \mathbf{nb}, \mathbf{na}, 1 \rangle$ and the facts $B{:}\langle \mathbf{nb}, \mathbf{a}, 0.7 \rangle$ and $C{:}\langle \mathbf{nb}, \mathbf{a}, 0.6 \rangle$. In each derivation, the annotation assigned to A gathers—component by component—the minimal values used during the derivation. In particular, during the second derivation, no blacklisted information is used, a non-authoritative rule is applied, and the rank value never falls below 0.6. It is easy to see that, according to Definition 3, both $A{:}\langle \mathbf{b}, \mathbf{a}, 0.4 \rangle$ and $A{:}\langle \mathbf{nb}, \mathbf{na}, 0.6 \rangle$ are logical consequences of P. ◇

The semantics in Definition 3 enjoys the same good properties as standard logic programming semantics. In particular, every given program P has one minimal model which contains exactly the logical consequences of P, and can be characterised as the least fixed point of an *immediate consequence* operator $\mathfrak{T}_P$. To see this, we need a suitable ordering over interpretations:

$$I \preceq I' \text{ iff } \text{for all } F \in \mathcal{B}_P, I(F) \subseteq I'(F)\,.$$

The partial order $\preceq$ induces a complete lattice in the set of all interpretations. Given a set of interpretations $\mathcal{I}$ the least upper bound $\sqcup\mathcal{I}$ and the greatest lower bound $\sqcap\mathcal{I}$ satisfy $\sqcup\mathcal{I}(F) = \bigcup_{I\in\mathcal{I}} I(F)$ and $\sqcap\mathcal{I}(F) = \bigcap_{I\in\mathcal{I}} I(F)$, for all $F \in \mathcal{B}_P$. The bottom interpretation Δ maps each $F \in \mathcal{B}_P$ to $\emptyset$.

The immediate consequence operator is a mapping over interpretations such that for all facts $F \in \mathcal{B}_P$:

$$\mathfrak{T}_P(I)(F) \coloneqq \bigcup_{F \leftarrow B_1,\ldots,B_m : \mathbf{d} \in \mathsf{Ground}(P)} \left\{ \mathrm{glb}(\{\mathbf{d}_1, \ldots, \mathbf{d}_m, \mathbf{d}\}) \mid \underset{1\leq i\leq m}{\forall} \mathbf{d}_i \in I(B_i) \right\}$$

Theorem 1 *For all programs P and interpretations I,*

(i) I is a model of P iff $\mathfrak{T}_P(I) \preceq I$;

(ii) $\mathfrak{T}_P$ is monotone, i.e., $I \preceq I'$ implies $\mathfrak{T}_P(I) \preceq \mathfrak{T}_P(I')$.

Proof: (Sketch) Our framework can be regarded as a special case of the general theory of annotated programs developed in [42]. In that framework, our rules can be reformulated as

$$H{:}f(X_1, \ldots, X_m, \mathbf{d}) \leftarrow B_1{:}X_1, \ldots, B_m{:}X_m$$

where each X_i is a variable ranging over $2^{\mathbf{D}}$ and

$$\begin{aligned} &f(X_1, \ldots, X_m, \mathbf{d}) \coloneqq \\ &\{\mathrm{glb}(\{\mathbf{d}_1, \ldots, \mathbf{d}_m, \mathbf{d}\}) \mid \mathbf{d}_i \in X_i \ (1 \leq i \leq m)\}\,. \end{aligned} \tag{3}$$

The upper semilattice of truth values $\mathcal{T}$ [42, Sec. 2] can be set to the complete lattice $\langle 2^{\mathbf{D}}, \subseteq, \cup, \cap\rangle$. Then our semantics corresponds to the *restricted* semantics defined in [42] and our operator $\mathfrak{T}_P$ corresponds to the operator $\mathfrak{R}_P$ which has been proven to satisfy the two statements. □

Corollary 2 *For all programs P,*

(i) P has one minimal model that equals the least fixed point of $\mathfrak{T}_P$, $\mathrm{lfp}(\mathfrak{T}_P)$;

(ii) for all $F \in \mathcal{B}_P$, $\mathbf{d} \in \mathrm{lfp}(\mathfrak{T}_P)(F)$ iff $P \models F{:}\mathbf{d}$.

Another standard consequence of Theorem 1 is that $\mathrm{lfp}(\mathfrak{T}_P)$ can be calculated in a bottom-up fashion, starting from the empty interpretation Δ and applying iteratively $\mathfrak{T}_P$. Define the iterations of $\mathfrak{T}_P$ in the usual way: $\mathfrak{T}_P \uparrow 0 \coloneqq \Delta$; for all ordinals α, $\mathfrak{T}_P \uparrow (\alpha + 1) \coloneqq \mathfrak{T}_P(\mathfrak{T}_P \uparrow \alpha)$; if α is a limit ordinal, let $\mathfrak{T}_P \uparrow \alpha \coloneqq \sqcup_{\beta<\alpha}\mathfrak{T}_P \uparrow \beta$. Now, it follows from Theorem 1 that there exists an α such that $\mathrm{lfp}(\mathfrak{T}_P) = \mathfrak{T}_P \uparrow \alpha$. To ensure that the logical consequences of P can be effectively computed, it should also be proven that $\alpha \leq \omega$, which is usually proven by showing that $\mathfrak{T}_P$ is continuous. In [42, Ex. 3], it is shown that these properties do not hold for general annotated programs—even if the program is Datalog (i.e., terms can only be constants)—because it is possible to infer infinite sequences $F{:}\mathbf{d}_1, F{:}\mathbf{d}_2, \ldots, F{:}\mathbf{d}_i, \ldots$ such that the sequence of labels $\mathbf{d}_1, \mathbf{d}_2, \ldots$ converges to a limit $\mathbf{d}^\infty$, but $F{:}\mathbf{d}^\infty \notin \mathfrak{R}_P \uparrow \omega$. We demonstrate this now by adapting Example 3 from [42].

Example: Consider a simple *generalised annotated program* P, with truth values $\mathcal{T}$ in the interval $[0,1]$ of real numbers, and three rules as follows:

$$A{:}0 \leftarrow$$
$$A{:}\tfrac{1+\alpha}{2} \leftarrow A{:}\alpha$$
$$B{:}1 \leftarrow A{:}1$$

By the restricted semantics of generalised annotated programs, $A{:}1 \in \mathfrak{R}_P \uparrow \omega$. However, since the third rule is discontinuous, $B{:}1 \notin \mathfrak{R}_P \uparrow \omega$ and so we see that $\mathfrak{R}_P \uparrow \omega \neq \mathrm{lfp}(\mathfrak{R}_P)$; note that $B{:}1 \in \mathfrak{R}_P \uparrow (\omega+1)$. $\diamond$

Then, in order to prove that our $\mathfrak{T}_P$ is continuous, we have to rely on the specific properties of the function $f(.)$ defined in (3).

Lemma 3 *Let $\mathbf{D}$ be a z-dimensional annotation domain, P a program and F a fact. The number of possible annotations $\mathbf{d}$ such that $P \models F{:}\mathbf{d}$ is bounded by $|P|^z$, where $|P|$ is the cardinality of P.*

Proof: Let D_i^P, for $1 \leq i \leq z$, be the set of all values occurring as the i-th component in some annotation in P and $\mathbf{D}^P := \times_{i=1}^{z} D_i^P$. Clearly, for all $i = 1, \ldots, z$, $|D_i^P| \leq |P|$, therefore the cardinality of $\mathbf{D}^P$ is at most $|P|^z$. We are only left to show that the annotations occurring in $\mathfrak{T}_P \uparrow \alpha$ are all members of $\mathbf{D}^P$. Note that if $\{\mathbf{d}_1, \ldots, \mathbf{d}_m, \mathbf{d}\} \subseteq \mathbf{D}^P$, then also $\mathrm{glb}\{\mathbf{d}_1, \ldots, \mathbf{d}_m, \mathbf{d}\} \in \mathbf{D}^P$. Then, by a straightforward induction on α, it follows that for all α, if $F{:}\mathbf{d} \in \mathfrak{T}_P \uparrow \alpha$, then $\mathbf{d} \in \mathbf{D}^P$. $\square$

In other words, since the glb function cannot introduce new elements from the component sets of the annotation domain, the application of $\mathfrak{T}_P$ can only create labels from the set of tuples which are combinations of existing domain elements in P; thus the set of all labels is bounded by $|P|^z$.

Next, in order to demonstrate that $\mathfrak{T}_P$ is continuous we must introduce the notion of a *chain of interpretations*: a sequence $\{I_\beta\}_{\beta \leq \alpha}$ such that for all $\beta < \gamma$, $I_\beta \preceq I_\gamma$.

Theorem 4 *For all programs P, $\mathfrak{T}_P$ is continuous: that is, for all chains $\mathcal{I} := \{I_\beta\}_{\beta \leq \alpha}$, $\mathfrak{T}_P(\sqcup\mathcal{I}) = \sqcup\{\mathfrak{T}_P(I) \mid I \in \mathcal{I}\}$.*

Proof: The $\supseteq$ inclusion is trivial. For the $\subseteq$ inclusion, assume that $\mathbf{d} \in \mathfrak{T}_P(\sqcup\mathcal{I})(F)$. By definition, there exists a rule $F \leftarrow B_1, \ldots, B_m : \mathbf{d}'$ in $\mathsf{Ground}(P)$ and some $\mathbf{d}_1, \ldots, \mathbf{d}_m$ such that $\mathbf{d} = \mathrm{glb}(\mathbf{d}', \mathbf{d}_1, \ldots, \mathbf{d}_m)$ and for all $1 \leq j \leq m$, $\mathbf{d}_j \in \sqcup\mathcal{I}(F)$. Therefore, for all $1 \leq j \leq m$ there exists a $\beta_j \leq \alpha$ such that $\mathbf{d}_j \in I_{\beta_j}$. Let β be the maximum value of $\mathbf{d}_1, \ldots, \mathbf{d}_m$; since $\mathcal{I}$ is a chain, $\mathbf{d}_j \in I_\beta(F)$, for all $1 \leq j \leq m$. Therefore, $\mathbf{d}$ is in $\mathfrak{T}_P(I_\beta)(F)$ and hence in $\sqcup\{\mathfrak{T}_P(I) \mid I \in \mathcal{I}\}(F)$. $\square$

Corollary 5 *The interpretation $\mathfrak{T}_P \uparrow \omega$ is the least fixed point of $\mathfrak{T}_P$, $\mathrm{lfp}(\mathfrak{T}_P)$, and hence it is the minimal model of P.*

The logical consequences of our programs satisfy another important property.

Lemma 6 *A fact $F : \mathbf{d}$ is a ground logical consequence of P iff for some (finite) $i < \omega$, $\mathbf{d} \in \mathfrak{T}_P \uparrow i(F)$.*

Proof: Due to corollaries 2 and 5, $P \models F{:}\mathbf{d}$ iff $\mathbf{d} \in \mathfrak{T}_P \uparrow \omega(F)$. By definition, $\mathfrak{T}_P \uparrow \omega(F) = \bigcup_{i<\omega} \mathfrak{T}_P \uparrow i(F)$, therefore $P \models F{:}\mathbf{d}$ iff for some finite $i < \omega$, $\mathbf{d} \in \mathfrak{T}_P \uparrow i(F)$. $\square$

In the jargon of classical logic, this means that our framework is *finitary*. As an immediate consequence of this lemma, the logical consequences of P are semidecidable. Note that for generalised annotated programs, even if $\mathfrak{R}_P$ is continuous, it may not be finitary.

Example: Consider the previous example (as borrowed from [42, Ex. 3]), but drop the last (discontinuous) rule:

$$A{:}0 \leftarrow$$
$$A{:}\tfrac{1+\alpha}{2} \leftarrow A{:}\alpha$$

This program is continuous with respect to the restricted semantics of general annotated programs, therefore $\mathfrak{R}_P \uparrow \omega = \mathrm{lfp}(\mathfrak{R}_P)$. However, although $A{:}1 \in \mathfrak{R}_P \uparrow \omega$, it is not finitary because for all $i < \omega$, $A{:}1 \notin \mathfrak{R}_P \uparrow i$. $\diamond$

Moreover, if P is Datalog, then the least fixed point of $\mathfrak{T}_P$ is reached after a finite number of iterations:

Lemma 7 *If P is Datalog, then there exists $i < \omega$ such that $\mathrm{lfp}(\mathfrak{T}_P) = \mathfrak{T}_P \uparrow i$.*

Proof: Due to corollary 2 and Lemma 3, for each $F \in \mathcal{B}_P$, the set $\mathrm{lfp}(\mathfrak{T}_P)(F)$ is finite. Moreover, when P is Datalog, the Herbrand base $\mathcal{B}_P$ is finite as well. Thus, the thesis immediately follows from the monotonicity of $\mathfrak{T}_P$. $\square$

It follows that the least model of P—denoted $\mathrm{lm}(P)$—can be finitely represented, and that the logical consequences of P are decidable.

4.3. *Annotated reasoning tasks*

Once the semantics of annotated programs has been defined, it is possible to consider several types of high-level reasoning tasks which, roughly speaking, refine the set of ground logical consequences according to optimality and threshold conditions.

Plain: Returns all of the ground logical consequences of P. Formally:

$$\mathsf{plain}(P) := \{F{:}\mathbf{d} \mid F \in \mathcal{B}_P \text{ and } P \models F{:}\mathbf{d}\}\,.$$

Optimal: Only the *non-dominated* elements of $\mathsf{plain}(P)$ are returned. Intuitively, an answer—say, $F{:}\langle \mathbf{nb}, \mathbf{nb}, 0.5\rangle$—can be ignored if a stronger evidence can be derived; for example $F{:}\langle \mathbf{nb}, \mathbf{a}, 0.6\rangle$. Formally, for all sets S of annotated rules and facts, let:

$$\max(S) \coloneqq \{R{:}\mathbf{d} \in S \mid \text{for all } R{:}\mathbf{d}' \in S,\ \mathbf{d} \not< \mathbf{d}'\}$$

and define $\mathsf{opt}(P) \coloneqq \max(\mathsf{plain}(P))$.

Above Threshold (Optimal): Refines $\mathsf{opt}(P)$ by selecting the consequences that are above a given threshold. Formally, given a threshold vector $\mathbf{t} \in \mathbf{D}$, for all sets S of annotated rules and facts let

$$S_{\geq \mathbf{t}} \coloneqq \{R{:}\mathbf{d} \in S \mid \mathbf{t} \leq \mathbf{d}\}$$

and define $\mathsf{opt}_{\mathbf{t}}(P) \coloneqq \mathsf{opt}(P)_{\geq \mathbf{t}}$.

Above Threshold (Classical): Returns the ground atoms that have some annotation above a given threshold $\mathbf{t}$. Annotations are not included in the answer. Formally, define:

$$\mathsf{above}_{\mathbf{t}}(P) \coloneqq \{F \in \mathcal{B}_P \mid \exists \mathbf{d} \geq \mathbf{t} \text{ s.t. } P \models F{:}\mathbf{d}\}\,.$$

All tasks except plain, make it possible to optimise the input program by dropping some rules that do not contribute to the answers. For example, $\mathsf{opt}(P)$ does not depend on the dominated elements of P, which can thus be discarded:

Theorem 8 $\mathsf{opt}(P) = \mathsf{opt}(\max(P))$.

Proof: Clearly, $\max(P) \subseteq P$ and both $\max$ and plain are monotonic wrt. set inclusion. Therefore, $\mathsf{plain}(\max(P))$ is contained in $\mathsf{plain}(P)$ and $\max(\mathsf{plain}(\max(P)))$ is contained in $\max(\mathsf{plain}(P))$, i.e., $\mathsf{opt}(\max(P)) \subseteq \mathsf{opt}(P)$.

For the opposite inclusion, we first prove by induction on natural numbers that for all $i \geq 0$ and $F \in \mathcal{B}_P$, if $\mathbf{d} \in \mathfrak{T}_P \uparrow i(F)$, then there exists an annotation $\mathbf{d}' \geq \mathbf{d}$ such that $\mathbf{d}' \in \mathfrak{T}_{\max(P)} \uparrow i(F)$.

The assertion is vacuously true for $i = 0$. Assume that $\mathbf{d} \in \mathfrak{T}_P \uparrow (i+1)(F)$, with $i \geq 0$, we have that for some rule $F \leftarrow B_1, \ldots, B_m : \overline{\mathbf{d}}$ in $\mathsf{Ground}(P)$ and some $\mathbf{d}_1, \ldots, \mathbf{d}_m$:

$$\mathbf{d} = \mathrm{glb}(\{\mathbf{d}_1, \ldots, \mathbf{d}_m, \overline{\mathbf{d}}\})$$

and for all $1 \leq j \leq m$, $\mathbf{d}_j \in \mathfrak{T}_P \uparrow i(B_j)$. By definition of $\max(P)$, there exists a $F \leftarrow B_1, \ldots, B_m : \widehat{\mathbf{d}}$ in $\mathsf{Ground}(\max(P))$ with $\widehat{\mathbf{d}} \geq \overline{\mathbf{d}}$. Moreover, by induction hypothesis, for all $1 \leq j \leq m$, there exists a $\mathbf{d}'_j \geq \mathbf{d}_j$ such that $\mathbf{d}'_j \in \mathfrak{T}_{\max(P)} \uparrow i(B_j)$. Thus if we set $\mathbf{d}' \coloneqq \mathrm{glb}(\{\mathbf{d}'_1, \ldots, \mathbf{d}'_m, \widehat{\mathbf{d}}\})$, we have $\mathbf{d}' \geq \mathbf{d}$ and $\mathbf{d}' \in \mathfrak{T}_{\max(P)} \uparrow (i+1)(F)$.

Now assume that $F{:}\mathbf{d} \in \mathsf{opt}(P)$. In particular $F{:}\mathbf{d}$ is a logical consequence of P: therefore, by Lemma 6 and the previous statement, we have that there exists an annotation $F{:}\mathbf{d}' \in \mathsf{plain}(\max(P))$ with $\mathbf{d} \leq \mathbf{d}'$. However, since $\mathsf{opt}(\max(P)) \subseteq \mathsf{opt}(P)$, $\mathsf{plain}(\max(P))$ cannot contain facts that improve $F{:}\mathbf{d}$, therefore $\mathbf{d}' = \mathbf{d}$ and $F{:}\mathbf{d} \in \mathsf{opt}(\max(P))$. □

Similarly, if a minimal threshold is specified, then the program can be filtered by dropping all of the rules that are not above threshold:

Theorem 9 $\mathsf{opt}_{\mathbf{t}}(P) = \mathsf{opt}(P_{\geq \mathbf{t}})$.

Proof: Since by definition we know that $\mathsf{opt}_{\mathbf{t}}(P) = \max(\mathsf{plain}(P))_{\geq \mathbf{t}}$ and $\max(\mathsf{plain}(P))_{\geq \mathbf{t}} = \max(\mathsf{plain}(P)_{\geq \mathbf{t}})$, it suffices to show that $\mathsf{plain}(P)_{\geq \mathbf{t}} = \mathsf{plain}(P_{\geq \mathbf{t}})$. By Lemma 6, $F{:}\mathbf{d} \in \mathsf{plain}(P)_{\geq \mathbf{t}}$ implies that for some i, $\mathbf{d} \in \mathfrak{T}_P \uparrow i(F)$ and $\mathbf{d} \geq \mathbf{t}$. Analogously to Theorem 8, it can be proven by induction on natural numbers that for all $i \geq 0$ and $F \in \mathcal{B}_P$, if $\mathbf{d} \in \mathfrak{T}_P \uparrow i(F)$ and $\mathbf{d} \geq \mathbf{t}$, then $\mathbf{d} \in \mathfrak{T}_{P_{\geq \mathbf{t}}} \uparrow i(F)$. Therefore, $\mathbf{d} \in \mathfrak{T}_{P_{\geq \mathbf{t}}} \uparrow i(F)$ and hence $F{:}\mathbf{d} \in \mathsf{plain}(P_{\geq \mathbf{t}})$.

Again it is easy to prove by induction that if $F{:}\mathbf{d} \in \mathsf{plain}(P_{\geq \mathbf{t}})$, then $\mathbf{d} \geq \mathbf{t}$. Thus, since plain is monotone wrt. set inclusion and $P_{\geq \mathbf{t}} \subseteq P$, $\mathsf{plain}(P_{\geq \mathbf{t}}) \subseteq \mathsf{plain}(P)$. But $\mathsf{plain}(P_{\geq \mathbf{t}})$ contains only annotations that dominate $\mathbf{t}$, hence $\mathsf{plain}(P_{\geq \mathbf{t}}) \subseteq \mathsf{plain}(P)_{\geq \mathbf{t}}$. □

4.4. *Scalability issues: general case*

We see that some of the reasoning tasks outlined above are amenable to straightforward optimisations which allow for pruning dominated facts and rules. However, in our application domain programs contain typically $\backsim 10^9$ facts coming from millions of RDF sources [36,38]. With programs of this size, it is necessary to carefully take into account the number of facts a task has to manipulate. In general a polynomial bound with respect to the cardinality of P is not a sufficient evidence of feasibility; even a quadratic exponent may be too high. Accordingly—and as discussed in Section 2—the OWL 2 RL/RDF ruleset has been restricted to the $\mathcal{O}2\mathcal{R}^-$ fragment where the number of logical consequences of a program grows linearly with the number of assertional facts [32]. In order to achieve the same bound for our reasoning tasks over the same fragment of OWL 2, for each $F \in \mathcal{B}_P$, the number of derived consequences $F{:}\mathbf{d}$ should be constant wrt. the cardinality of P (equivalently, the number of different annotations associated to each proper rule or fact should be constant wrt. $|P|$). In this section, we now give a detailed assessment of the four reasoning tasks with respect to this requirement.

From Lemma 3, we know that the cardinality of

$\mathsf{plain}(P)$ is bounded by $|P|^z$; we now use an example to demonstrate that this bound is tight.

Example: Consider a z-dimensional $\mathbf{D}$ where each component i may assume an integer value from 1 to n. Let P be the following propositional program consisting of all rules of the following form:

$$\begin{array}{rll} A_1 & : \langle m_1, n, \ldots, n\rangle & (1 \leq m_1 \leq n) \\ A_i \leftarrow A_{i-1} & : \langle n, \ldots, m_i, \ldots, n\rangle & (1 \leq m_i \leq n) \\ \ldots & & (\ 2 \leq i \leq z) \end{array}$$

where, intuitively, m_i assigns all possible values to each component i. Now, there are n facts which have every possible value for the first annotation component and the value n for all other components. Thereafter, for each of the remaining $z-1$ annotation components, there are n annotated rules which have every possible value for the given annotation component, and the value n for all other components. Altogether, the cardinality of P is nz. The set of annotations that can be derived for A_z is exactly $\mathbf{D}$, therefore its cardinality is n^z which grows as $\Theta(|P|^z)$. When $z \geq 2$, the number of labels associated to A_z alone exceeds the desired linear bound on materialisations.

To illustrate this, let's instantiate P for $n = 2$ and $z = 3$:

$$\begin{array}{c} A_1 : \langle 1,2,2\rangle\ ,\ A_1 : \langle 2,2,2\rangle\ , \\ A_2 \leftarrow A_1 : \langle 2,1,2\rangle\ ,\ A_2 \leftarrow A_1 : \langle 2,2,2\rangle\ , \\ A_3 \leftarrow A_2 : \langle 2,2,1\rangle\ ,\ A_3 \leftarrow A_2 : \langle 2,2,2\rangle\ . \end{array}$$

Here, $|P| = 2 * 3 = 6$. By $\mathsf{plain}(P)$, we will get:

$$\begin{array}{c} A_1 : \langle 1,2,2\rangle\ ,\ A_1 : \langle 2,2,2\rangle\ , \\ A_2 : \langle 1,1,2\rangle\ ,\ A_2 : \langle 1,2,2\rangle\ ,\ A_2 : \langle 2,1,2\rangle\ ,\ A_2 : \langle 2,2,2\rangle\ , \\ A_3 : \langle 1,1,1\rangle\ ,\ A_3 : \langle 1,1,2\rangle\ ,\ A_3 : \langle 1,2,1\rangle\ ,\ A_3 : \langle 1,2,2\rangle\ , \\ A_3 : \langle 2,1,1\rangle\ ,\ A_3 : \langle 2,1,2\rangle\ ,\ A_3 : \langle 2,2,1\rangle\ ,\ A_3 : \langle 2,2,2\rangle\ . \end{array}$$

Where A_3 is associated with $2^3 = 8$ annotations. $\diamond$

Such examples naturally preclude the reasoning task $\mathsf{plain}(.)$ for our scenario, making $\mathsf{opt}(.)$ a potentially appealing alternative. Unfortunately, even if Theorem 8 enables some optimisation, in general $\mathsf{opt}(P)$ is itself not linear wrt. $|P|$. This can be seen with a simple example which uses RDF rules and two rational-valued annotation properties:

Example: Consider a program containing all facts and rules of the form:

$$\begin{array}{c} (\texttt{ex:Foo},\ \texttt{ex:spam},\ \texttt{ex:Bar}) : \langle k, \frac{1}{k}\rangle \\ \ldots \\ (\texttt{?x},\ \texttt{rdf:\_}k,\ \texttt{?y}) \leftarrow (\texttt{?y},\ \texttt{ex:spam},\ \texttt{?x}) : \langle n, n\rangle \\ \ldots \end{array}$$

such that n is a (constant) positive integer and $1 \leq k \leq n$. By increasing n, P grows as $\Theta(2n)$, whereas (as per the previous example) $|\mathsf{plain}(P)|$ grows as $\Theta(n^2)$, with n facts of the form ($\texttt{ex:Foo}$, $\texttt{ex:spam\_}k$, $\texttt{ex:Bar}$) being associated with n annotations of the form $\langle k, \frac{1}{k}\rangle$—thus, $|\mathsf{plain}(P)|$ grows quadratically with $|P|$. Now, for all such consequences relative to the same fact $F{:}\langle h, \frac{1}{h}\rangle$ and $F{:}\langle j, \frac{1}{j}\rangle$, if $h < j$, then $\frac{1}{j} < \frac{1}{h}$ and vice versa. This implies that all of the derived consequences are optimal—that is, $\mathsf{plain}(P) = \mathsf{opt}(P)$. Consequently, $|\mathsf{opt}(P)|$ grows quadratically with $|P|$, too.

To illustrate this, let's instantiate P for $n = 3$:

$$\begin{array}{c} (\texttt{ex:Foo},\ \texttt{ex:spam},\ \texttt{ex:Bar}) : \langle 1,1\rangle\ , \\ (\texttt{ex:Foo},\ \texttt{ex:spam},\ \texttt{ex:Bar}) : \langle 2, \frac{1}{2}\rangle\ , \\ (\texttt{ex:Foo},\ \texttt{ex:spam},\ \texttt{ex:Bar}) : \langle 3, \frac{1}{3}\rangle\ , \\ (\texttt{?x},\ \texttt{rdf:\_1},\ \texttt{?y}) \leftarrow (\texttt{?y},\ \texttt{ex:spam},\ \texttt{?x}) : \langle 3,3\rangle\ , \\ (\texttt{?x},\ \texttt{rdf:\_2},\ \texttt{?y}) \leftarrow (\texttt{?y},\ \texttt{ex:spam},\ \texttt{?x}) : \langle 3,3\rangle\ , \\ (\texttt{?x},\ \texttt{rdf:\_3},\ \texttt{?y}) \leftarrow (\texttt{?y},\ \texttt{ex:spam},\ \texttt{?x}) : \langle 3,3\rangle\ . \end{array}$$

Here, $|P| = 2^3 = 6$. Now, $\mathsf{opt}(P)$—or, equivalently, $\mathsf{plain}(P)$—will give $3^2 = 9$ annotated facts of the form:

$$\begin{array}{l} (\texttt{ex:Bar},\ \texttt{rdf:\_}k,\ \texttt{ex:Foo}) : \langle 1,1\rangle\ , \\ (\texttt{ex:Bar},\ \texttt{rdf:\_}k,\ \texttt{ex:Foo}) : \langle 2, \frac{1}{2}\rangle\ , \\ (\texttt{ex:Bar},\ \texttt{rdf:\_}k,\ \texttt{ex:Foo}) : \langle 3, \frac{1}{3}\rangle\ . \end{array} \quad \Big(1 \leq k \leq 3\Big)$$

Here, all 9 facts are considered optimal (again, we do not assume a lexicographical order). $\diamond$

In the above example, the annotations associated to each atom grow linearly with $|P|$. We can generalise this result and prove that the annotations of a given atom may grow as $|P|^{\frac{z}{2}}$ by slightly modifying the example for plain materialisation.

Example: Assume that the dimension z is even, and that a component may assume any value from the set of rationals in the interval $[0, n]$; now, consider the program P containing all such facts and rules:

$$\begin{array}{rll} A_1 & : \langle r_1, \frac{1}{r_1}, n \ldots, n\rangle & (1 \leq r_1 \leq n) \\ A_i \leftarrow A_{i-1} & : \langle n, \ldots, n, r_{2i-1}, \frac{1}{r_{2i-1}}, n, \ldots, n\rangle & (1 \leq r_{2i-1} \leq n) \\ \ldots & & (2 \leq i \leq \frac{z}{2}) \end{array}$$

where r_{2i-1} assigns the $(2i-1)$th (odd) component all possible values between 1 and n inclusive, and $\frac{1}{r_{2i-1}}$ assigns the $2i$th (even) component all possible values between 1 and $\frac{1}{n}$ inclusive. Note that the cardinality of P is $\frac{nz}{2}$, containing n facts and $\frac{n(z-2)}{2}$ proper rules. Now, given two consequences $A{:}\mathbf{d}_1, A{:}\mathbf{d}_2 \in \mathsf{plain}(P)$ sharing the same atom A, there exist two distinct integers $j, k \leq n$, $j \neq k$ and a pair of contiguous components $i, i+1 \leq z$ such that $\mathbf{d}_1 = \langle \ldots, j, \frac{1}{j}, \ldots\rangle$ and $\mathbf{d}_2 = \langle \ldots, k, \frac{1}{k}, \ldots\rangle$. Therefore, all of the facts in $\mathsf{plain}(P)$ are optimal, and the number of annotations for $A_{\frac{z}{2}}$ is

$n^{\frac{z}{2}}$.

To illustrate this, let's instantiate P for $n = 2$ and $z = 6$:

$$A_1 : \langle 1,1,2,2,2,2\rangle \ , \ A_1 : \langle 2,\tfrac{1}{2},2,2,2,2\rangle \ ,$$
$$A_2 \leftarrow A_1 : \langle 2,2,1,1,2,2\rangle \ , \ A_2 \leftarrow A_1 : \langle 2,2,2,\tfrac{1}{2},2,2\rangle \ ,$$
$$A_3 \leftarrow A_2 : \langle 2,2,2,2,1,1\rangle \ , \ A_3 \leftarrow A_2 : \langle 2,2,2,2,2,\tfrac{1}{2}\rangle \ .$$

Here, $|P| = \frac{2\times 6}{2} = 6$. By $\mathsf{opt}(P)$—or, equivalently, by $\mathsf{plain}(P)$—we will get:

$$\begin{array}{l} A_1 : \langle 1,1,2,2,2,2\rangle \ , A_1 : \langle 2,\tfrac{1}{2},2,2,2,2\rangle \ , \\ A_2 : \langle 1,1,1,1,2,2\rangle \ , A_2 : \langle 1,1,2,\tfrac{1}{2},2,2\rangle \ , \\ A_2 : \langle 2,\tfrac{1}{2},1,1,2,2\rangle \ , A_2 : \langle 2,\tfrac{1}{2},2,\tfrac{1}{2},2,2\rangle \ , \\ A_3 : \langle 1,1,1,1,1,1\rangle \ , A_3 : \langle 1,1,1,1,2,\tfrac{1}{2}\rangle \ , \\ A_3 : \langle 1,1,2,\tfrac{1}{2},1,1\rangle \ , A_3 : \langle 1,1,2,\tfrac{1}{2},2,\tfrac{1}{2}\rangle \ , \\ A_3 : \langle 2,\tfrac{1}{2},1,1,1,1\rangle \ , A_3 : \langle 2,\tfrac{1}{2},1,1,2,\tfrac{1}{2}\rangle \ , \\ A_3 : \langle 2,\tfrac{1}{2},2,\tfrac{1}{2},1,1\rangle \ , A_3 : \langle 2,\tfrac{1}{2},2,\tfrac{1}{2},2,\tfrac{1}{2}\rangle \ . \end{array}$$

Here, A_3 is associated with $2^{\frac{6}{2}} = 8$ optimal annotations.

It is not hard to adapt this example to odd z and prove that for all $z \in \mathbb{N}$, the number of annotations associated to an atom is in $\Theta(|P|^{\lfloor \frac{z}{2} \rfloor})$. Therefore, if the number of distinct atoms occurring in the answer can grow linearly (as in the aforementioned fragment $\mathcal{O}2\mathcal{R}^-$), then $|\mathsf{opt}(P)|$ is in $\Theta(|P|^{\lfloor \frac{z}{2} \rfloor+1})$ and hence not linear for $z > 1$. ◇

If we further restrict our computations to the consequences above a given threshold (i.e., $\mathsf{opt}_{\mathbf{t}}(.)$), then some improvements may be possible (cf. Theorem 9). However it is clear that the worst case complexity of $\mathsf{opt}_{\mathbf{t}}(.)$ and $\mathsf{opt}(.)$ is the same (it suffices to set $\mathbf{t}$ to the least element of $\mathbf{D}$). Thus, neither $\mathsf{opt}_{\mathbf{t}}(.)$ nor $\mathsf{opt}(.)$ are suitable for our use-case scenario in the general case.

The last reasoning task, $\mathsf{above}_{\mathbf{t}}(P)$, returns atoms without annotations, and hence it is less informative than the other tasks. However, it does not suffer from the performance drawbacks of the other tasks: $\mathsf{above}_{\mathbf{t}}(.)$ does not increase the complexity of annotation-free reasoning because it only needs to drop the rules whose annotation is not above $\mathbf{t}$ and reason classically with the remaining rules, as formalised by the following proposition:

Theorem 10 *Let* $\mathrm{lm}(P^{\mathbf{t}})$ *denote the least Herbrand model of the (classical) program* $P^{\mathbf{t}}$. *Then* $\mathsf{above}_{\mathbf{t}}(P) = \mathrm{lm}(P_{\geq \mathbf{t}})$.

Proof: Let $\mathfrak{C}$ be the classical immediate consequence operator as per Section 2.3 and define $\mathfrak{C}_{P_{\geq \mathbf{t}}} \uparrow \alpha$ by analogy with $\mathfrak{T}_{P_{\geq \mathbf{t}}} \uparrow \alpha$. It is straightforward to see by induction on natural numbers that for all $i \geq 0$, $\mathfrak{T}_{P_{\geq \mathbf{t}}} \uparrow i(F) \neq \emptyset$ iff $F \in \mathfrak{C}_{P_{\geq \mathbf{t}}} \uparrow i$. This means that $F \in \mathrm{lm}(P_{\geq \mathbf{t}})$ iff $\mathfrak{T}_{P_{\geq \mathbf{t}}} \uparrow \omega(F) \neq \emptyset$, or equivalently, iff for some $\mathbf{d}$, $F{:}\mathbf{d} \in \mathsf{plain}(P_{\geq \mathbf{t}})$. Moreover, as already shown in the proof of Theorem 9, $\mathsf{plain}(P_{\geq \mathbf{t}}) = \mathsf{plain}(P)_{\geq \mathbf{t}}$, therefore $F{:}\mathbf{d} \in \mathsf{plain}(P)_{\geq \mathbf{t}}$ (for some $\mathbf{d}$) iff $F \in \mathsf{above}_{\mathbf{t}}(P)$. □

4.5. *Scalability issues:* opt *and* $\mathsf{opt}_{\mathbf{t}}$ *for our use-case*

The analysis carried out so far apparently suggests that $\mathsf{above}_{\mathbf{t}}$ is the only practically feasible inference among the four tasks. However, the output facts are not associated with annotations in this scheme, which may be required for many use-cases scenarios (including our use-case for repairing inconsistencies investigated later).

Although we have shown that opt and $\mathsf{opt}_{\mathbf{t}}$ are polynomial in the general case, our chosen Linked Data annotation domain—comprising of blacklisting, triple-rank and authoritativeness—enjoys certain properties that can be exploited to efficiently implement both opt and $\mathsf{opt}_{\mathbf{t}}$. Such properties bound the number of maximal labels with an expression that is constant with respect to P and depends only on the annotation domains: the most important property is that all D_is but one are finite (blacklisting and authoritativeness are boolean; only triple-ranks range over an infinite set of values). Thus, the number of maximal elements of any finite set of annotations is bounded by a linear function of $\mathbf{D}$.

Example: Here we see an example of a fact with four non-dominated annotations from our use-case domain.

$$\begin{array}{l} (\texttt{ex:Foo}, \texttt{ex:precedes}, \texttt{ex:Bar}) : \langle \mathbf{b}, \mathbf{na}, 0.4\rangle \ , \\ (\texttt{ex:Foo}, \texttt{ex:precedes}, \texttt{ex:Bar}) : \langle \mathbf{b}, \mathbf{a}, 0.3\rangle \ , \\ (\texttt{ex:Foo}, \texttt{ex:precedes}, \texttt{ex:Bar}) : \langle \mathbf{nb}, \mathbf{na}, 0.3\rangle \ , \\ (\texttt{ex:Foo}, \texttt{ex:precedes}, \texttt{ex:Bar}) : \langle \mathbf{nb}, \mathbf{a}, 0.2\rangle \ . \end{array}$$

Any additional annotation for this fact would either dominate or be dominated by a current annotation—in either case, the set of maximal annotations would maintain a cardinality of four. ◇

We now formalise and demonstrate this intuitive result. *For simplicity, we assume without further loss of generality that the finite domains are* $D_1, \ldots, D_{z-1}$*; we make no assumption on* D_z. [22]

First, with a little abuse of notation, given $\mathbf{D}' \subseteq \mathbf{D}$, $\max(\mathbf{D}')$ is the set the of maximal values of D', i.e., $\{\mathbf{d} \in \mathbf{D}' \mid \forall \mathbf{d}' \in \mathbf{D}', \ \mathbf{d} \not< \mathbf{d}'\}$.

Theorem 11 *If* $D_1, \ldots, D_{z-1}$ *are finite, then for all finite* $\mathbf{D}' \subseteq \mathbf{D}$, $|\max(\mathbf{D}')| \leq \Pi_1^{z-1}|D_i|$.

[22] Note that for convenience, this indexing of the (in)finite domains replaces the former indexing presented in Section 4.2 relating to how annotations are labelled—the two should be considered independent.

Proof: Clearly, there exist at most $\Pi_1^{z-1}|D_i|$ combinations of the first $z-1$ components. Therefore, if $|\max(\mathbf{D}')|$ was greater than $\Pi_1^{z-1}|D_i|$, there would be two annotations $\mathbf{d}_1$ and $\mathbf{d}_2$ in $\max(\mathbf{D}')$ that differ only in the last component. But in this case either $\mathbf{d}_1 > \mathbf{d}_2$ or $\mathbf{d}_1 < \mathbf{d}_2$, and hence they cannot be both in $\max(\mathbf{D}')$ (a contradiction). □

As a consequence, in our reference scenario (where $z = 3$ and $|D_1| = |D_2| = 2$) each atom can be associated with at most 4 different annotations. Therefore, if the rules of P belong to a linear fragment of OWL 2, then $\mathsf{opt}(P)$ grows linearly with the size of P.

However, a linear bound on the *output* of reasoning tasks does not imply the same bound on the intermediate steps (e.g., the alternative framework introduced in the next subsection needs to compute also non-maximal labels for a correct answer). Fortunately, a bottom-up computation that considers only maximal annotations is possible in this framework. Let $\mathfrak{T}_P^{\max}(I)$ be such that for all $F \in \mathcal{B}_P$:

$$\mathfrak{T}_P^{\max}(I)(F) := \max(\mathfrak{T}_P(I)(F))$$

and define its powers $\mathfrak{T}_P^{\max} \uparrow \alpha$ by analogy with $\mathfrak{T}_P \uparrow \alpha$.

Lemma 12 *For all ordinals α, $\mathfrak{T}_P^{\max} \uparrow \alpha = \max(\mathfrak{T}_P \uparrow \alpha)$.*

Proof: First we prove the following claim:

$$\max(\mathfrak{T}_P(\max(I))) = \max(\mathfrak{T}_P(I))\,.$$

The inclusion $\subseteq$ is trivial. For the other inclusion, assume that for some $F \in \mathcal{B}_P$, $\mathbf{d} \in \max(\mathfrak{T}_P(I)(F))$, this means that $\mathbf{d} \in \mathfrak{T}_P(I)(F)$ and for all $\mathbf{d}' \in \mathfrak{T}_P(I)(F)$, $\mathbf{d} \not< \mathbf{d}'$. As $\mathbf{d} \in \mathfrak{T}_P(I)(F)$, there exists a rule $F \leftarrow B_1, \ldots, B_m : \overline{\mathbf{d}}$ and some annotations $\mathbf{d}_1, \ldots, \mathbf{d}_m$ such that (i) $\mathbf{d} = \mathrm{glb}(\{\mathbf{d}_1, \mathbf{d}_m, \overline{\mathbf{d}}\})$ and (ii) for all $1 \leq j \leq m$, $\mathbf{d}_j \in I(B_j)$.

By definition, for all $1 \leq j \leq m$, there exists a $\mathbf{d}_j^{\max} \in \max(I)$ such that $\mathbf{d}_j \leq \mathbf{d}_j^{\max}$. Clearly, given

$$\mathbf{d}^{\max} = \mathrm{glb}(\{\mathbf{d}_1^{\max}, \ldots, \mathbf{d}_m^{\max}, \overline{\mathbf{d}}\})\,,$$

$\mathbf{d} \leq \mathbf{d}^{\max}$ and $\mathbf{d}^{\max} \in \mathfrak{T}_P(\max(I))$. However, since $\mathbf{d}$ is maximal wrt. $\mathfrak{T}_P(I)(F)$ and $\mathfrak{T}_P(\max(I))(F) \subseteq \mathfrak{T}_P(I)(F)$, then $\mathbf{d} \leq \mathbf{d}^{\max}$ and $\mathbf{d} \in \max(\mathfrak{T}_P(\max(I)))$. This proves the claim.

Now the lemma follows by an easy induction based on the claim. □

It follows from Lemma 12 that $\mathfrak{T}_P^{\max}$ reaches a fixpoint, although $\mathfrak{T}_P^{\max}$ is not monotonic (because annotations may be only temporarily optimal and be replaced at later steps). When P is Datalog this fixpoint is reached in a finite number of steps:

Theorem 13 *If P is Datalog, then there exists $i < \omega$ such that*

(i) $\mathfrak{T}_P^{\max} \uparrow i$ is a fixpoint of $\mathfrak{T}_P^{\max}$;

(ii) $\mathfrak{T}_P^{\max} \uparrow j$ is not a fixpoint of $\mathfrak{T}_P^{\max}$, for all $0 \leq j < i$;

(iii) $F{:}\mathbf{d} \in \mathsf{opt}(P)$ iff $\mathbf{d} \in \mathfrak{T}_P^{\max} \uparrow i(F)$.

Proof: If P is Datalog, by Lemma 7, for some $k < \omega$, $\mathfrak{T}_P \uparrow k = \mathrm{lfp}(\mathfrak{T}_P)$, we will show that $\mathfrak{T}_P^{\max} \uparrow k$ is a fixpoint as well. By definition

$$\mathfrak{T}_P^{\max}(\mathfrak{T}_P^{\max} \uparrow k) = \max(\mathfrak{T}_P(\mathfrak{T}_P^{\max} \uparrow k))\,,$$

by Lemma 12, $\mathfrak{T}_P^{\max} \uparrow k = \max(\mathfrak{T}_P \uparrow k)$, so we have

$$\mathfrak{T}_P^{\max}(\mathfrak{T}_P^{\max} \uparrow k) = \max(\mathfrak{T}_P(\max(\mathfrak{T}_P \uparrow k)))\,.$$

However, as already shown in the proof of Lemma 12, for any I, $\max(\mathfrak{T}_P(\max(I))) = \max(\mathfrak{T}_P(I))$. Therefore,

$$\mathfrak{T}_P^{\max}(\mathfrak{T}_P^{\max} \uparrow k) = \max(\mathfrak{T}_P(\mathfrak{T}_P \uparrow k))\,.$$

Finally, since $\mathfrak{T}_P \uparrow k$ is a fixpoint and reusing Lemma 12

$$\mathfrak{T}_P^{\max}(\mathfrak{T}_P^{\max} \uparrow k) = \max(\mathfrak{T}_P \uparrow k) = \mathfrak{T}_P^{\max} \uparrow k\,.$$

Thus, $\mathfrak{T}_P^{\max} \uparrow k$ is a fixpoint and hence, by finite regression, there exists an $0 \leq i \leq k$ such that $\mathfrak{T}_P^{\max} \uparrow i$ is a fixpoint and for all $0 \leq j < i$, $\mathfrak{T}_P^{\max} \uparrow j$ is not a fixpoint.

Clearly, $\mathfrak{T}_P^{\max} \uparrow k = \mathfrak{T}_P^{\max} \uparrow i$. Since $\mathfrak{T}_P^{\max} \uparrow k = \max(\mathfrak{T}_P \uparrow k)$ (Lemma 12), we finally have

$$\mathfrak{T}_P^{\max} \uparrow i = \max(\mathrm{lfp}(\mathfrak{T}_P))\,.$$

Therefore, $\mathbf{d} \in \mathfrak{T}_P^{\max} \uparrow i$ iff $F{:}\mathbf{d} \in \max(\mathsf{plain}(P)) = \mathsf{opt}(P)$. □

Theorem 11 ensures that at every step j, $\mathfrak{T}_P^{\max} \uparrow j$ associates each derived atom to a number of annotations that is constant with respect to $|P|$. By Theorem 9, the bottom-up construction based on $\mathfrak{T}_P^{\max}$ can be used also to compute $\mathsf{opt}_{\mathbf{t}}(P) = \mathsf{opt}(P_{\geq \mathbf{t}})$. Informally speaking, this means that if $D_1, \ldots, D_{z-1}$ are finite, then both $\mathsf{opt}(.)$ and $\mathsf{opt}_{\mathbf{t}}(.)$ are feasible.

Another useful property of our reference scenarios is that the focus is on non-blacklisted and authoritative consequences—that is, the threshold $\mathbf{t}$ in $\mathsf{opt}_{\mathbf{t}}(.)$ has $z-1$ components set to their maximum possible value. In this case, it turns out that each atom can be associated to one optimal annotation.

Example: Given our threshold $\mathbf{t} := \langle \mathbf{nb}, \mathbf{a}, 0 \rangle$ and the following triples:

$(\texttt{ex:Foo}, \texttt{ex:precedes}, \texttt{ex:Bar}) : \langle \mathbf{b}, \mathbf{na}, 0.4 \rangle$,
$(\texttt{ex:Foo}, \texttt{ex:precedes}, \texttt{ex:Bar}) : \langle \mathbf{b}, \mathbf{a}, 0.3 \rangle$,
$(\texttt{ex:Foo}, \texttt{ex:precedes}, \texttt{ex:Bar}) : \langle \mathbf{nb}, \mathbf{na}, 0.3 \rangle$,
$(\texttt{ex:Foo}, \texttt{ex:precedes}, \texttt{ex:Bar}) : \langle \mathbf{nb}, \mathbf{a}, 0.2 \rangle$,
$(\texttt{ex:Foo}, \texttt{ex:precedes}, \texttt{ex:Bar}) : \langle \mathbf{nb}, \mathbf{a}, 0.6 \rangle$,

we see that only the latter two triples are above the threshold, and only the last is optimal. Similarly, any additional annotation for this triple will either (i) be below the threshold, (ii) be optimal, or (iii) be dominated. In any case, a maximum of one annotation is maintained, which is non-blacklisted, authoritative, and contains the optimal rank value. ◇

We now briefly formalise this result. *For the sake of simplicity, we assume without loss of generality that the threshold elements set to the maximum value are the first $z-1$.*

Theorem 14 *Let $\mathbf{t} := \langle t_1, \ldots, t_z \rangle$. If $t_i = \top_i$ for $1 \leq i < z$, then for all $\mathbf{D}' \subseteq \mathbf{D}$, $|\max(\mathbf{D}'_{\geq \mathbf{t}})| \leq 1$.*

Proof: If not empty, all of the annotations in $\mathbf{D}'_{\geq \mathbf{t}}$ are of type $\langle \top_1, \ldots, \top_{z-1}, d_z \rangle$, thus $\max$ selects the one with the maximal value of d_z. □

As a consequence, each atom occurring in $\mathsf{opt}_{\mathbf{t}}(P)$ is associated to one annotation, and the same holds for the intermediate steps $\mathfrak{T}^{\max}_{P_{\geq \mathbf{t}}} \uparrow j$ of the iterative construction of $\mathsf{opt}_{\mathbf{t}}(P)$:

Theorem 15 *Assume that P is Datalog and the assumption of Theorem 14 holds. Let i be the least index such that $\mathfrak{T}^{\max}_{P_{\geq \mathbf{t}}} \uparrow i$ is a fixpoint of $\mathfrak{T}^{\max}_{P_{\geq \mathbf{t}}}$. Then*

(i) if $\{F{:}\mathbf{d}_1, F{:}\mathbf{d}_2\} \subseteq \mathsf{opt}_{\mathbf{t}}(P)$, then $\mathbf{d}_1 = \mathbf{d}_2$;
(ii) if $\{\mathbf{d}_1, \mathbf{d}_2\} \subseteq \mathfrak{T}^{\max}_{P_{\geq \mathbf{t}}} \uparrow j(F)$ $(0 \leq j \leq i)$, then $\mathbf{d}_1 = \mathbf{d}_2$.

Proof: We prove both the assertions by showing that for all $j \geq 0$ if $\{\mathbf{d}_1, \mathbf{d}_2\} \subseteq \mathfrak{T}^{\max}_{P_{\geq \mathbf{t}}} \uparrow j(F)$ $(0 \leq j \leq i)$, then $\mathbf{d}_1 = \mathbf{d}_2$.

For $j = 0$, the assertion is vacuously true. For $j > 0$, $\mathfrak{T}^{\max}_{P_{\geq \mathbf{t}}} \uparrow j(F) = \max(\mathfrak{T}_{P_{\geq \mathbf{t}}}(\mathfrak{T}^{\max}_{P_{\geq \mathbf{t}}} \uparrow (j-1))(F))$, therefore both $\mathbf{d}_1$ and $\mathbf{d}_2$ are maximal ($\mathbf{d}_1 \not\prec \mathbf{d}_2$ and $\mathbf{d}_1 \not\succ \mathbf{d}_2$). But $\mathbf{d}_1$ and $\mathbf{d}_2$ differ only for the last component z, and since $\leq_z$ is a total order, then $\mathbf{d}_1 = \mathbf{d}_2$. □

The experiments reported in the rest of the paper belong to this case: formally, the threshold $\mathbf{t}$ is $\langle \mathbf{nb}, \mathbf{a}, 0 \rangle = \langle \top_1, \top_2, \bot_3 \rangle$. Accordingly, the implementation maintains a single annotation for each derived atom, where rules and facts below the threshold can be pruned as part of a straightforward pre-processing step.

4.6. *Alternative approaches*

As a side note, the discussion above shows that scalability problems may arise in the general case from the existence of a polynomial number of maximal annotations for the same atom. Then it may be tempting to force a total order on annotations and keep for each atom only its (unique) best annotation, in the attempt to obtain a complexity similar to above-threshold reasoning. In our reference scenario, for example, it might make sense to order annotation triples lexicographically, thereby giving maximal importance to blacklisting (i.e. information correctness), medium importance to authoritativeness, and minimal importance to page ranking, so that—for example—$\langle \mathbf{nb}, \mathbf{na}, 0.9 \rangle \leq \langle \mathbf{nb}, \mathbf{a}, 0.8 \rangle$. Then interpretations could be restricted by forcing $I(F)$ to be always a singleton, containing the unique maximal annotation for F according to the lexicographic ordering.

Unfortunately, this idea does not work well together with the standard notion of rule satisfaction introduced before. In general, in order to infer the correct maximal annotation associated to an atom A it may be necessary to keep some non-maximal annotation, too (therefore the analogue of Lemma 12 does not hold in this setting). We illustrate the problem with a simple example.

Example: Consider for example the program:

$$H \leftarrow B : \langle \mathbf{nb}, \mathbf{na}, 1.0 \rangle$$
$$B : \langle \mathbf{nb}, \mathbf{na}, 0.9 \rangle$$
$$B : \langle \mathbf{nb}, \mathbf{a}, 0.8 \rangle .$$

The best proof of H makes use of the first two rules/facts of the program, and gives H the annotation $\langle \mathbf{nb}, \mathbf{na}, 0.9 \rangle$, as none of these rules/facts are blacklisted or authoritative, and the least page rank is 0.9. However, if we could associate each atom to its best annotation only, then B would be associated to $\langle \mathbf{nb}, \mathbf{a}, 0.8 \rangle$, and the corresponding label for H would only be forced to be $\langle \mathbf{nb}, \mathbf{na}, 0.8 \rangle$ by the definition of rule satisfaction; therefore this semantics (in conjunction with lexicographic orderings) does not faithfully reflect the properties of the best proof of H. ◇

One may consider a further alternative semantics whereby each dimension of the annotation domain is tracked independently of each other. However, with these semantics, if, for example, we find an inference which is associated with the ranks 0.2 and 0.8, and the blacklisted values $\mathbf{nb}$ and $\mathbf{b}$, we cannot distinguish whether the high rank was given by the derivation involving blacklisted knowledge, or non-blacklisted knowledge—in the general case, this would also pre-

scribe a more liberal semantics for our thresholding. [23]

Currently we do not know whether any other alternative, reasonable semantics can solve these problems, and we leave this issue as an open question—in any case, we note that this discussion does not affect our intended use-case of deriving $\mathsf{opt}_{\mathbf{t}}(P)$ for our threshold since, as per Theorem 15, we need only consider the total ordering given by the single dimension of triple ranks.

4.7. *Constraints*

In this paper, our referential use-case for annotated reasoning is a non-trivial repair of inconsistencies in Linked Data based on the strength of derivation for individual facts (as encoded by the annotations with which they are associated). Given that we apply rule-based reasoning, inconsistencies are axiomatised by constraints, which are expressed as rules without heads:

$$\leftarrow A_1, \ldots, A_n, T_1, \ldots, T_m \quad (n, m \geq 0) \qquad (4)$$

where $T_1, \ldots, T_m$ are T-atoms and $A_1, \ldots, A_n$ are A-atoms. As before, let $\mathsf{Body}(R) := \{A_1, \ldots, A_n, T_1, \ldots, T_m\}$ and let $\mathsf{TBody}(R) := \{T_1, \ldots, T_m\}$.

Example: Take the OWL 2 RL/RDF (meta-)constraint cax-dw:

$\leftarrow \underline{(\texttt{?c}_1, \texttt{owl:disjointWith}, \texttt{?c}_2)}, (\texttt{?x}, \texttt{a}, \texttt{?c}_1), (\texttt{?x}, \texttt{a}, \texttt{?c}_2)$

where $\mathsf{TBody}(\textsf{cax-dw})$ is again underlined (T_1). Any grounding of this rule denotes an inconsistency. ◇

Classical semantics prescribes that a Herbrand model I satisfies a constraint C iff I satisfies no instance of $\mathsf{Body}(C)$. Consequently, if P is a logic program with constraints, either the least model of P's rules satisfies all constraints in P or P is inconsistent (in this case, no reasoning can be carried out with P).

Annotations create an opportunity for a more flexible and reasonable use of constraints. Threshold-based reasoning tasks can be used to ignore the consequences of constraint violations based on low-quality or otherwise unreliable proofs. In the following, let $P := P^r \cup P^c$, where P^r is a set of rules and P^c a set of constraints.

Definition 4 *Let* $\mathbf{t} \in \mathbf{D}$. *$P$ is* $\mathbf{t}$*-consistent iff* $\mathsf{above}_{\mathbf{t}}(P^r)$ *satisfies all of the constraints of* P^c.

For example, if $\mathbf{t} := \langle \mathbf{nb}, \mathbf{a}, 0 \rangle$ and P is $\mathbf{t}$-consistent, then for all constraints $C \in P^c$ all of the proofs of $\mathsf{Body}(C)$ use either blacklisted facts or non-authoritative rules. In other words, all inconsistencies are the consequence of ill-formed knowledge and/or ontology hijacking. For example—and as we will see in more detail in our empirical analysis (Section 8)—we find that our Web corpus is consistent for the threshold $\langle \mathbf{nb}, \mathbf{a}, 0.001616 \rangle$.

This form of consistency can be equivalently characterised in terms of the other threshold-dependent reasoning task ($\mathsf{opt}_{\mathbf{t}}$). For all sets of annotated rules and facts S, let $[S] := \{R \mid R : \mathbf{d} \in S\}$. Then we have:

Proposition 16 *P is $\mathbf{t}$-consistent iff $[\mathsf{opt}_{\mathbf{t}}(P^r)]$ satisfies all of the constraints in P^c.*

More generally, the following definitions can be adopted for measuring the strength of constraint violations:

Definition 5 (Answers) *Let $G := \{A_1, \ldots, A_n\}$ be a set of atoms and let $P := P^r \cup P^c$. An* answer *for G (from P) is a pair $\langle \theta, \mathbf{d} \rangle$ where θ is a grounding substitution and*

(i) there exist $\mathbf{d}_1, \ldots, \mathbf{d}_n \in \mathbf{D}$ such that $P^r \models A_i\theta{:}\mathbf{d}_i$,

(ii) $\mathbf{d} = \mathrm{glb}\{\mathbf{d}_1, \ldots, \mathbf{d}_n\}$.

The set of all answers of G from P is denoted by $\mathsf{Ans}_P(G)$.

Definition 6 (Annotated constraints, violation degree) Annotated constraints *are expressions $C{:}\mathbf{d}$ where C is a constraint and $\mathbf{d} \in \mathbf{D}$.* [24] *The* violation degree *of $C{:}\mathbf{d}$ wrt. program P is* $\max\{\mathrm{glb}(\mathbf{d}, \mathbf{d}') \mid \langle \theta, \mathbf{d}' \rangle \in \mathsf{Ans}_P(\mathsf{Body}(C))\}$.

Intuitively, violation degrees provide a way of assessing the severity of inconsistencies by associating each constraint with the rankings of their strongest violations.

Example: As per the previous example, take the OWL 2 RL/RDF (meta-)constraint cax-dw : $\langle \top_{\mathbf{1}}, \top_{\mathbf{2}}, \top_{\mathbf{3}} \rangle$, and consider the following set of annotated facts:

$$(\texttt{foaf:Organization}, \texttt{owl:disjointWith}, \texttt{foaf:Person}): \langle \mathbf{nb}, \mathbf{a}, 0.6 \rangle,$$
$$(\texttt{ex:W3C}, \texttt{a}, \texttt{foaf:Organization}): \langle \mathbf{nb}, \mathbf{a}, 0.4 \rangle,$$
$$(\texttt{ex:W3C}, \texttt{a}, \texttt{foaf:Person}): \langle \mathbf{nb}, \mathbf{na}, 0.3 \rangle,$$
$$(\texttt{ex:TimBL}, \texttt{a}, \texttt{foaf:Organization}): \langle \mathbf{b}, \mathbf{na}, 0.5 \rangle,$$
$$(\texttt{ex:TimBL}, \texttt{a}, \texttt{foaf:Person}): \langle \mathbf{nb}, \mathbf{a}, 0.7 \rangle,$$

There are four *answers* to the constraint, given by the above facts; viz.:

[23] Given that we do not consider a lexicographical order, we could support these semantics by flattening the set of annotation triples associated with each rule/fact into a triple of sets of values, where to support the various reasoning tasks outlined (other than plain), we would have to make straightforward amendments to our max, opt, $\mathsf{opt}_{\mathbf{t}}$ and $\mathsf{above}_{\mathbf{t}}$ functions.

[24] Constraint annotations are produced by abstract annotation functions of type 3 ($\alpha_i : \mathsf{Rules} \times (\mathbf{S} \to 2^{\mathsf{Facts}}) \to D_i$), that apply to constraints with no modification to their signature (assuming that the set Rules comprises of constraints as well).

$\big(\{\texttt{?x/ex:W.}, \texttt{?c}_1\texttt{/foaf:P.}, \texttt{?c}_2\texttt{/foaf:O.}\}, \langle \mathbf{nb}, \mathbf{na}, 0.3\rangle\big)$,
$\big(\{\texttt{?x/ex:W.}, \texttt{?c}_1\texttt{/foaf:O.}, \texttt{?c}_2\texttt{/foaf:P.}\}, \langle \mathbf{nb}, \mathbf{na}, 0.3\rangle\big)$,
$\big(\{\texttt{?x/ex:T.}, \texttt{?c}_1\texttt{/foaf:P.}, \texttt{?c}_2\texttt{/foaf:O.}\}, \langle \mathbf{b}, \mathbf{na}, 0.5\rangle\big)$,
$\big(\{\texttt{?x/ex:T.}, \texttt{?c}_1\texttt{/foaf:O.}, \texttt{?c}_2\texttt{/foaf:P.}\}, \langle \mathbf{b}, \mathbf{na}, 0.5\rangle\big)$,

here abbreviating the respective CURIEs. Note that the first two and last two answers are given by the same sets of facts. Again, the annotations of the answers are derived from the glb function over the respective facts.

The violation degree of cax-dw is then $\{\langle \mathbf{nb}, \mathbf{na}, 0.3\rangle, \langle \mathbf{b}, \mathbf{na}, 0.5\rangle\}$ since neither annotation dominates the other. ◇

The computation of violation degrees can be reduced to opt by means of a simple program transformation. Suppose that $P^c := \{C_1{:}\mathbf{d}_1, \ldots, C_m{:}\mathbf{d}_m\}$. Introduce a fresh propositional symbol f_i for each C_i and let

$$P' := P^r \cup \{f_i \leftarrow \mathsf{Body}(C_i) : \mathbf{d}_i \mid i = 1, \ldots, m\}.$$

Proposition 17 *An annotation* $\mathbf{d}$ *belongs to the violation degree of* $C_i{:}\mathbf{d}_i$ *iff* $f_i{:}\mathbf{d} \in \mathsf{opt}(P')$.

Of course, violation degrees and thresholds can be combined by picking up only those annotations that are above threshold. This can be done by selecting all $\mathbf{d}$ such that $f_i : \mathbf{d} \in \mathsf{opt}_{\mathbf{t}}(P')$—as such, in our use-case we will again only be looking for violations above our threshold $\mathbf{t} := \langle \top_1, \top_2, \bot_3\rangle$.

Another possible use of annotations in relation to constraints is knowledge base repair. If $C_i{:}\mathbf{d}_i$ is violated, then the members of $\mathsf{Body}(C_i)$ with the weakest proof are good candidates for deletion. These ideas will be further elaborated in Section 8, where we sketch a scalable method for knowledge base repair.

5. Implementation and experimental setup

Having looked in detail at the formal aspects of the annotated program framework—in particular aspects relating to the scalability of the different reasoning tasks—we now move towards detailing our scalable implementation and empirical results. In the following sections, we look at the design, implementation and evaluation of (i) ranking (Section 6); (ii) annotated reasoning (Section 7); (iii) inconsistency detection and repair using constraint rules (Section 8). Before we continue, in this section we describe the distributed framework (see [37]) which forms the substrate of our experiments, and then describe our experimental Linked Data corpus.

5.1. *Distribution architecture*

Our methods are implemented on a shared-nothing distributed architecture [58] over a cluster of commodity hardware. In particular, we leverage the nature of our algorithms to attempt to perform the most expensive tasks in an *embarrassingly parallel* fashion, whereby there is little or no co-ordination required between machines.

The distributed framework consists of one master machine which orchestrates the given tasks, and several slave machines which perform parts of the task in parallel.

The master machine can instigate the following distributed operations:

- **scatter:** partition on-disk data into logical chunks given some local *split* function, and send the chunks to individual slave machines for subsequent processing;
- **run:** request the parallel execution of a task by the slave machines—such a task either involves processing of some local data (usually involving embarrassingly parallel execution), or execution of the **coordinate** method by the slave swarm;
- **gather:** gathers chunks of output data from the slave swarm and performs some local *merge* function over the data—this is usually performed to create a single output file for a task, or more usually to gather global knowledge required by all slave machines for a future task;
- **flood:** broadcast global knowledge required by all slave machines for a future task.

The master machine is intended to disseminate input data to the slave swarm, to provide the control logic required by the distributed task (commencing tasks, coordinating timing, ending tasks), to gather and locally perform tasks on global knowledge which the slave machines would otherwise have to replicate in parallel, and to transmit globally required knowledge.

The slave machines, as well as performing tasks in parallel, can perform the following distributed operation (on the behest of the master machine):

- **coordinate:** local data on each slave machine are partitioned according to some *split* function, with the chunks sent to individual machines in parallel; each slave machine also gathers the incoming chunks in parallel using some *merge* function.

The above operation allows slave machines to partition and disseminate intermediary data directly to other slave machines; the **coordinate** operation could be replaced by a pair of **gather/scatter** operations performed by the master machine, but we wish to avoid the channelling of all such intermediary data through one machine.

Note that herein, we assume that the input corpus is evenly distributed and split across the slave machines,

and that the slave machines have roughly even specifications: i.e., we do not consider any special form of load balancing, but instead aim to have uniform machines processing comparable data-chunks.

5.2. *Experimental setup*

We instantiate the distributed architecture described in Section 5.1 using the standard Java Remote Method Invocation libraries as a convenient means of development given our Java code-base.

All of our evaluation is based on nine machines connected by Gigabit ethernet,[25] each with uniform specifications; viz.: 2.2GHz Opteron x86-64, 4GB main memory, 160GB SATA hard-disks, running Java 1.6.0_12 on Debian 5.0.4. Again please note that much of the performance evaluation presented herein assumes that the slave machines have roughly equal specifications in order to ensure that tasks finish in roughly the same time, assuming even data distribution.

5.3. *Experimental corpus*

Later in this paper, we discuss the performance and fecundity of applying our methods over a corpus of 1.118b quadruples (triples with an additional *context* element encoding source) derived from an RDF/XML crawl of 3.985m Web documents in mid-May 2010. Of the 1.118b quads, 1.106b are unique, and 947m are unique triples. The data contain 23k unique predicates and 105k unique class terms (terms in the object position of an `rdf:type` triple). Please see [37, Appendix A] for further statistics relating to this corpus.

Quadruples are stored as GZip compressed flat files of N-Quads.[26] Also, all machines have a local copy of a sorted list of redirects encountered during the crawl: i.e., ordered pairs of the form $\langle f, t\rangle$ where $\mathsf{redir}(f) = t$. The input data are unsorted, not uniqued, and pre-distributed over eight slave machines according to a hash function of context; this is the direct result of our distributed crawl, details of which are available in [37].

To evaluate our methods for varying corpus sizes, we also create subsets of the corpus, containing one half, one quarter and one eighth of the quadruples: to do so, we extract the appropriate subset from the head of the full corpus—note that our raw data are naturally ordered according to when they were crawled, so smaller subsets taken from the head emulate a smaller crawl. Also, in order to evaluate our methods for a varying number of slave machines, we additionally merge each of the four sizes of corpora onto one, two and four machines by hashing on context. Thus, we have the one eighth, one quarter, one half and all of the corpus respectively split over one, two, four and eight machines; the average counts of quadruples per machine for each configuration are listed in Table 1. Note that henceforth, we use c, $\frac{\mathsf{c}}{2}$, $\frac{\mathsf{c}}{4}$ and $\frac{\mathsf{c}}{8}$ to denote the full, half, quarter and eighth corpus respectively, and use $1\mathsf{sm}$, $2\mathsf{sm}$, $4\mathsf{sm}$ and $8\mathsf{sm}$ to denote one, two, four and eight slave machines (and one master machine) respectively.

corpus	1sm (*total*)	2sm	4sm	8sm
c	1,117,567,842	558,783,921	280,159,613	139,859,503
$\frac{c}{2}$	558,783,923	279,391,962	140,079,807	69,929,752
$\frac{c}{4}$	279,391,964	139,695,982	70,039,904	34,964,876
$\frac{c}{8}$	139,695,985	69,847,992	34,923,996	17,461,998

Table 1
Average quadruples per machine for the four different sizes of corpora and four different distribution setups

Looking at how evenly the corpora are split across the machines, the average absolute deviation of quadruples—as a percentage of the mean—was 0.9%, 0.55% and 0.14% for eight, four and two slaves respectively, representing near-even data distribution given by hashing on context (these figures are independent of the size of the corpus).

Note that for our implementation, we use compressed files of N-Triple/N-Quad type syntax for serialising arbitrary-length line-delimited tuples of RDF terms.

6. Links-based analysis: implementation and evaluation

In this section, we describe and evaluate our distributed methods for applying a PageRank-inspired analysis of the data to derive rank annotations for facts in the corpus. We begin by discussing the high-level approach (Section 6.1), then by discussing the distributed implementation (Section 6.2), and finally we present evaluation for our corpus (Section 6.3).

6.1. *High-level ranking approach*

Again, our links-based analysis is inspired by the approach presented in [29], in which the authors present and compare two levels of granularity for deriving source-level ranks: (i) document-level granularity analysing inter-linkage between Web documents; and (ii) *pay-level-domain* (PLD) granu-

[25] We observe, e.g., a max FTP transfer rate of 38MB/sec between machines.
[26] `http://sw.deri.org/2008/07/n-quads/`

larity analysing inter-linkage between domains (e.g., `dbpedia.org`, `data.gov.uk`). Document-level analysis is more expensive—in particular, it generates a larger graph for analysis—but allows for more fine-grained ranking of elements resident in different documents. PLD-level analysis is cheaper—it generates a smaller graph—but ranks are more coarse-grained, and many elements can share identical ranks grouped by the PLDs they appear in.

In previous work [37], we employed in-memory techniques for applying a PageRank analysis of the PLD-granularity source graph: we demonstrated this to be efficiently computable in our distributed setup due to the small size of the graph extracted (in particular, the PageRank calculation took <1min in memory applying 10 iterations), and argued that PLD-level granularity was sufficient for that particular use-case (ranking entity 'importance', which combined with TF-IDF keyword search relevance forms the basis of prioritising entity-search results). However, in this paper we opt to apply the more expensive document-level analysis which provides more granular results useful for repairing inconsistent data in Section 8. Since we will deal with larger graphs, we opt for on-disk batch-processing techniques—mainly sorts, scans, and sort-merge joins of on-disk files—which are not hard-bound by in-memory capacity, but which are significantly slower than in-memory analysis.

6.2. *Distributed ranking implementation*

Individual tasks are computed using parallel sorts, scans and merge-joins over the slave machines in our cluster. For reference, we provide the detailed description of the ranking sub-algorithms in Appendix B, where our high-level distributed approach can be summarised as follows:

(i) **run/gather/flood**: each slave machine sorts its segment of the data by context, with a list of sorted contexts generated on each machine; the master machine gathers the list of contexts, merge-sorting a global list of contexts which is subsequently flooded to the slave machines;

(ii) **run**: the slave machines extract the source-level graph in parallel from their segment of the data (Algorithm 2), rewrite the vertices in the sub-graph using redirects (Algorithm 3), prune links in the sub-graph such that it only contains vertices in the global list of contexts (Algorithm 4), and finally sorts the sub-graph according to inlinks and outlinks;

(iii) **gather/flood**: the master machine gathers the sub-graphs (for both inlink and outlink order) from the slave machines and merge-sorts to create the global source-level graph; the master machine then performs the power iteration algorithm to derive PageRank scores for individual contexts using on-disk sorts and scans (Algorithms 5 & 6); ranks are subsequently flooded to each machine;

(iv) **run/coordinate**: each slave machine propagates ranks of contexts to individual quadruples (merge-join over ranks and data sorted by context); the ranked data are subsequently sorted by subject; each slave machine splits its segment of sorted ranked data by a hash-function (modulo slave-machine count) on the subject position, with split fragments sent to a target slave machine; each slave machine concurrently receives and stores split fragments from its peers;

(v) **run**: each slave machine merge-sorts the subject-hashed fragments it received from its peers, summating the ranks for triples which appear in multiple contexts while streaming the data.

The results of this process are: (a) data sorted and hashed by context on the slave machines; (b) ranks for contexts on all machines; (c) data sorted and hashed by subject on the slave machines, with aggregated ranks for triples. For (c), the result is simply a flat file of sorted quintuples of the form $\langle s, p, o, c, r\rangle$, where c denotes context and r rank, and where $r_i = r_j$ if $\langle s_i, p_i, o_i\rangle = \langle s_j, p_j, o_j\rangle$.

Note that, with respect to (b), we could select a data-partitioning strategy which hashes on any arbitrary (but consistent) element (or combination of elements) of each triple. We hash on the subject position since data-skew becomes a problem for other positions [47]; in our input data we observe that:

(i) hashing on predicates would lead to poor load-balancing, where, e.g., we observe that `rdf:type` appears as the predicate for 206.8m input quadruples (18.5% of the corpus);

(ii) hashing on objects would also lead to poor load-balancing, where, e.g., `foaf:Person` appears as the object for 163.7m input quadruples (14.6% of the corpus).

Conversely, in Figure 1, we present the distribution of subjects with respect to the number of quadruples they appear in: although there is an *apparent* power-law, we note that the most frequently occurring subject term

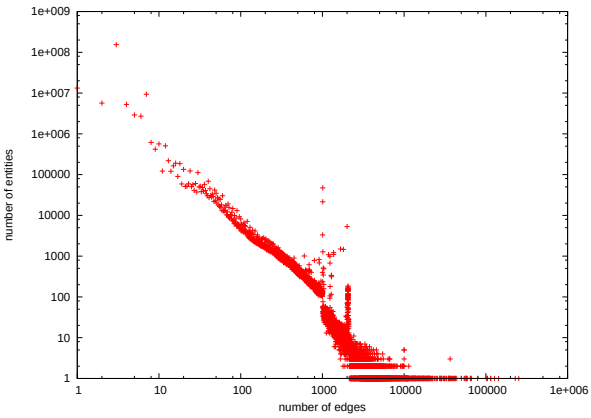

Fig. 1. Distribution of quadruples per subject

appeared in 252k quadruples (0.02% of the corpus). [27]

Subsequent to partitioning the full corpus by hashing on subject across the eight slave machines, we found an average absolute deviation of 166.3k triples from the mean (0.012% of the mean), representing near-optimal load balancing.

We note that in [47]—which argues against "single-element hashing" for partitioning RDF data—although the authors observe a high-frequency for common terms in a large corpus also acquired from the Web (14% of all triples had the literal object `"0"`), they do not give a positional analysis, and in our experience, none of the highlighted problematic terms commonly appear in the subject position. If the number of machines was to increase greatly, or the morphology of the input data meant that hashing on subject would lead to load-balancing problems, one could consider hashing on the entire triple, which would *almost surely* offer excellent load-balancing. Otherwise, we believe that hashing on subject is preferable given that (i) the low-level operation is cheaper, with less string-characters to hash; and (ii) less global duplicates will be produced by the distributed reasoning process (Section 7) during which machines operate independently. [28]

[27] There are some irregular "peaks" apparent the distribution, where we encounter unusually large numbers of subjects just above "milestone" counts of quadruples due to fixed limits in exporters: for example, the `hi5.com` exporter only allows users to have 2,000 values for `foaf:knows`—e.g., see: `http://api.hi5.com/rest/profile/foaf/100614697`.

[28] For example, using the LRU duplicate removal strategy mentioned in Section 7, we would expect little or no duplicates to be given by rule prp-dom when applied over a corpus hashed and sorted by subject.

6.3. *Ranking evaluation*

Applying distributed ranking over the full 1.118b statement corpus using eight slave machines and one master machine took just under 36hrs, with the bulk of time consumed as follows: (i) parallel sorting of data by context took 2.2hrs; (ii) parallel extraction and preparation of the source-level graph took 1.9hrs; (iii) ranking the source-level graph on the master machine took 26.1hrs (applying ten power iterations); (iv) propagating source ranks to triples and hashing/coordinating and sorting the data by subject took 3hrs; (v) merge-sorting the data segments and aggregating the ranks for triples took 1.2hrs.

We present the high-level results of applying the ranking for varying machines and corpus sizes (as per Table 1) in Figure 2. [29] For the purposes of comparison, when running experiments on one slave machine, we also include the coordinate step—splitting and merging the data—which would not be necessary if running the task locally. For reference, in Table 2 we present the factors by which the runtimes changed when doubling the size of the corpus and when doubling the number of slave machines. We note that:

– ranking with respect to $\frac{c}{4}$ takes, on average, 2.71× longer than ranking over $\frac{c}{8}$—otherwise, doubling the corpus size roughly equates to a doubling of runtime;
– when doubling the number of slave machines, runtimes do not halve: in particular, moving from four slaves to eight slaves reduces runtime by an average of 17%, and only 7% for the full corpus.

doubling data				**doubling slaves**			
sm	$c/\frac{c}{2}$	$\frac{c}{2}/\frac{c}{4}$	$\frac{c}{4}/\frac{c}{8}$	c	2sm/1sm	4sm/2sm	8sm/4sm
1sm	1.95	1.97	2.43	c	0.64	0.71	0.93
2sm	1.97	2.03	2.49	$\frac{c}{2}$	0.64	0.72	0.81
4sm	1.94	2.02	2.82	$\frac{c}{4}$	0.62	0.73	0.83
8sm	2.22	1.99	3.11	$\frac{c}{8}$	0.60	0.64	0.75
mean	2.02	2.00	2.71	**mean**	0.62	0.70	0.83

Table 2
On the left, we present the factors by which the ranking total runtimes changed when doubling the data. On the right, we present the factors by which the runtimes changed when doubling the number of slave machines.

With respect to the first observation, we note that the PageRank calculations on the master machine took 3.8× longer for $\frac{c}{4}$ than $\frac{c}{8}$: although the data and the number of sources roughly double from $\frac{c}{8}$ to $\frac{c}{4}$, the number of

[29] Note that all such figures in this paper are presented log/log with base 2, such that each major tick on the x-axis represents a doubling of slave machines, and each major tick on the y axis represents a doubling of time.

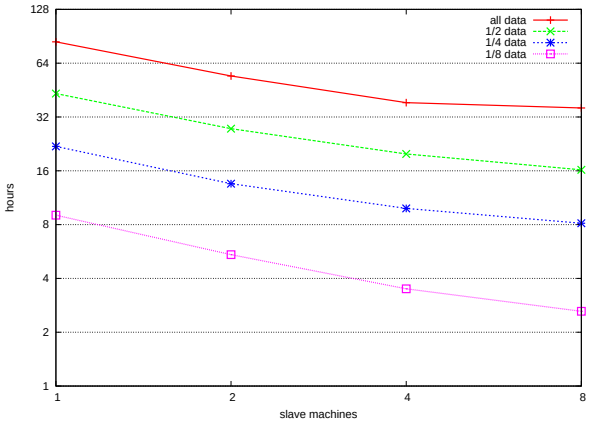

Fig. 2. Total time taken for ranking on 1/2/4/8 slave machines for varying corpus sizes

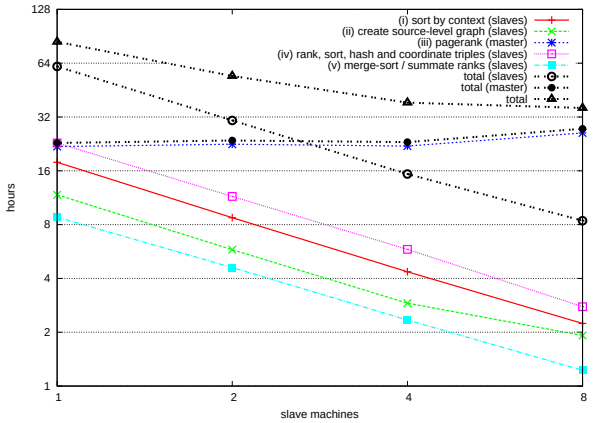

Fig. 3. Breakdown of relative ranking task times for 1/2/4/8 slave machines and full corpus

links between sources in the corpus quadrupled—we observed 11.3m links for $\frac{c}{8}$, 40.3m links for $\frac{c}{4}$, 80.7m links for $\frac{c}{2}$, and 183m links for c.

With respect to the second observation, we note that the local PageRank computation on the master machine causes an increasingly severe bottleneck as the number of slave machines increases: again, taking the full corpus, we note that this task takes 26.1hrs, irrespective of the number of slave machines available. In Figure 3, we depict the relative runtimes for each of the ranking tasks over the full corpus and varying number of machines. In particular, we see that the PageRank computation stays relatively static whilst the runtimes of the other tasks decrease steadily: at four slave machines, the total time spent in parallel execution is already less than the total time spent locally performing the PageRank. Otherwise—and with the exception of extracting the graph from $\frac{c}{4}$ and $\frac{c}{8}$ for reasons outlined above—we see that the total runtimes for the other (parallel) tasks roughly halve as the number of slave machines doubles.

With respect to redressing this local bottleneck, significant improvements could already be made by distributing the underlying sorts and scans required by the PageRank analysis. Further, we note that parallelising PageRank has been the subject of significant research, where we could look to leverage works as presented, e.g., in [24,45]. We leave this issue open, subject to investigation in another scope.

Focussing on the results for the full corpus, the source-level graph consisted of 3.985m vertices (sources) and 183m unique non-reflexive links. In Table 3, we provide the top 10 ranked documents. The top result refers to the RDF namespace, followed by the RDFS namespace, the DC namespace and the OWL namespace (the latter three are referenced by the RDF namespace, and amongst themselves). Subsequently, the fifth result contains some links to multi-lingual translations of labels for RDFS terms, and is linked to by the RDFS document; the sixth result refers to an older version of DC (extended by result 3); the seventh result is the SKOS vocabulary; the eighth result provides data and terms relating to the GRDDL W3C standard; the ninth result refers to the FOAF vocabulary; the tenth is a French translation of RDFS terms. Within the top ten, all of the documents are primarily concerned with terminological data (e.g., are ontologies or vocabularies, or describe classes or properties): the most wide-spread re-use of terms across the Web of Data is on a terminological level, representing a higher in-degree and subsequent rank for terminological documents. The average rank annotation for the triples in the corpus was 1.39×10^{-5}.

7. Reasoning with annotations—implementation and evaluation

In order to provide a scalable and distributed implementation for annotated reasoning, we extend the SAOR system described in [38] to support annotations. Herein, we summarise the distributed reasoning steps, where we refer the interested reader to [38] for more information about the (non-annotated) algorithms. We begin by discussing the high-level approach (Section 7.1), then discuss the distributed approach (Section 7.2), the extension to support annotations (Section 7.3), and present evaluation (Section 7.4).

#	Document	Rank
1	http://www.w3.org/1999/02/22-rdf-syntax-ns	0.112
2	http://www.w3.org/2000/01/rdf-schema	0.104
3	http://dublincore.org/2008/01/14/dcelements.rdf	0.089
4	http://www.w3.org/2002/07/owl	0.067
5	http://www.w3.org/2000/01/rdf-schema-more	0.045
6	http://dublincore.org/2008/01/14/dcterms.rdf	0.032
7	http://www.w3.org/2009/08/skos-reference/skos.rdf	0.028
8	http://www.w3.org/2003/g/data-view	0.014
9	http://xmlns.com/foaf/spec/	0.014
10	http://www.w3.org/2000/01/combined-ns-translation.rdf.fr	0.010

Table 3
Top 10 ranked documents

7.1. *High-level reasoning approach*

Given our explicit requirement for scale, in [38] we presented a generalisation of existing scalable rule-based reasoning systems which rely on a separation of T-Box from A-Box [36,65,63,62]: we call this generalisation the *partial-indexing* approach, which may only require indexing small subsets of the corpus, depending on the program/ruleset. The core idea follows the discussion of the T-split least fixpoint in Section 2.5 where our reasoning is broken into two distinct phases: (i) T-Box level reasoning over terminological data and rules where the result is a set of T-Box inferences and a set of T-ground rules; (ii) A-Box reasoning with respect to the set of proper T-ground rules.

With respect to RDF(S)/OWL(2) reasoning, we assume that we can distinguish terminological data (as presented in Section 2.5) and that the program consists of T-split rules, with known terminological patterns. Based on the *assumption* that the terminological segment of a corpus is relatively small, and that it is static with respect to reasoning over non-terminological data, we can perform some pre-compilation with respect to the terminological data and the given program before accessing the main corpus of assertional data. [30]

The high-level (local) procedure is as follows:

(i) identify and separate out the terminological data (T-Box) from the main corpus;
(ii) recursively apply axiomatic rules (Table A.1) and rules which do not require assertional knowledge (Table A.2) over the T-Box—the results are included in the A-Box and if terminological, may be recursively included in the T-Box, where the T-Box will thereafter remain static;
(iii) for rules containing both T-atoms and A-atoms, ground the T-atoms wrt. the T-Box facts, applying the (respective) most general unifier over the remainder of the rule—the result is the assertional program where all T-atoms are ground;
(iv) recursively apply the T-ground program over the assertional data.

The application of rules in Step (ii) and the grounding of T-atoms in Step (iii) are performed by standard semi-naïve evaluation over the indexed T-Box. However, in Step (iv), instead of bulk-indexing the A-Box, we apply the rules by means of a scan. This process is summarised in Algorithm 1.

First note that duplicate inference steps may be applied for rules with only one atom in the body (Lines 11–14): one of the main optimisations of our approach is that it minimises the amount of data that we need to index, where we only wish to store triples which may be necessary for later inference, and where triples *only* grounding single atom rule bodies need not be indexed. To provide *partial* duplicate removal, we instead use a Least-Recently-Used (LRU) cache over a sliding window of recently encountered triples (Lines 7 & 8)—outside of this window, we may not know whether a triple has been encountered before or not, and may repeat inferencing steps. In [38], we found that 84% of non-filtered inferences were unique using a fixed LRU-cache with a capacity of 50k over data grouped by context. [31]

Thus, in this *partial-indexing* approach, we need only index triples which are matched by a rule with a multi-atom body (Lines 15–25). For indexed triples, aside from the LRU cache, we can additionally check to see if that triple has been indexed before (Line 19) and we can apply a semi-naïve check to ensure that we

[30] If the former assumption does not hold, then our approach is likely to be less efficient than standard (e.g., semi-naïve) approaches. If the latter assumption does not hold (e.g., due to extension of RDF(S)/OWL terms), then our approach will be incomplete [38].

[31] Note that without the cache, cycles are detected by maintaining a set of inferences for a given input triple (cf. Algorithm 1; Line 30)—the cache is to avoid duplication, not to ensure termination.

Algorithm 1 Reason over the A-Box

Require: ABOX: A /* $\{t_0 \dots t_m\}$ */
Require: ASSERTIONAL PROGRAM: AP /* $\{R_0 \dots R_n\}, \mathsf{TBody}(R_i) = \emptyset$ */
1: Index := {} /* *triple index* */
2: LRU := {} /* *fixed-size, least recently used cache* */
3: **for all** $t \in$ A **do**
4: $G_0 := \{\}, G_1 := \{t\}, i := 1$
5: **while** $G_i \neq G_{i-1}$ **do**
6: **for all** $t_\delta \in G_i \setminus G_{i-1}$ **do**
7: **if** $t_\delta \notin$ LRU **then** /* *if* $t_\delta \in$ LRU, t_δ *most recent* */
8: add t_δ to LRU /* *remove eldest if necessary* */
9: output(t_δ)
10: **for all** $R \in AP$ **do**
11: **if** $|\mathsf{Body}(R)| = 1$ **then**
12: **if** $\exists\theta$ s.t. $\{t_\delta\} = \mathsf{Body}(R)\theta$ **then**
13: $G_{i+1} := G_{i+1} \cup \mathsf{Head}(R)\theta$
14: **end if**
15: **else**
16: **if** $\exists\theta$ s.t. $t_\delta \in \mathsf{Body}(R)\theta$ **then**
17: $card := |\text{Index}|$
18: Index := Index $\cup \{t_\delta\}$
19: **if** $card \neq |\text{Index}|$ **then**
20: **for all** θ s.t. $\mathsf{Body}(R\theta) \subseteq \text{Index} \wedge t_\delta \in \mathsf{Body}(R\theta)$ **do**
21: $G_{i+1} := G_{i+1} \cup \mathsf{Head}(R\theta)$
22: **end for**
23: **end if**
24: **end if**
25: **end if**
26: **end for**
27: **end if**
28: **end for**
29: i++
30: $G_{i+1} :=$ copy(G_i) /* *avoids cycles* */
31: **end while**
32: **end for**
33: **return** output /* *on-disk inferences* */

only materialise inferences which involve the current triple (Line 20). We note that as the assertional index is required to store more data, the two-scan approach becomes more inefficient than the "full-indexing" approach; in particular, a rule with a body atom containing all variable terms will require indexing of all data, negating the benefits of the approach; e.g., if the rule OWL 2 RL/RDF rule eq-rep-s:

(?s′, ?p, ?o) ← (?s, owl:sameAs, ?s′), (?s, ?p, ?o)

is included in the assertional program, the entire corpus of assertional data must be indexed (in this case according to subject) because of the latter "open" atom. We emphasise that our partial-indexing performs well if the assertional index remains small and performs best if every proper rule in the assertional program has only one A-atom in the body—in the latter case, no assertional indexing is required.

If the original program does not contain rules with multiple A-atoms in the body—such as is the case for our subset $\mathcal{O}2\mathcal{R}^-$ of OWL 2 RL/RDF rules—no indexing of the A-Box is required (Lines 11–15) other than for the LRU cache. [32]

Note that, depending on the nature of the T-Box and the original program, Step (ii) may result in a very large assertional program containing many T-ground rules where Algorithm 1 can be seen as a brute-force method for applying all rules to all statements. In [38], we derived 301k raw T-ground rules from the Web, and estimated that brute force application of all rules to all ∼1b triples would take >19 years. We thus proposed a number of optimisations for the partial-indexing approach; we refer the interested reader to [38] for details, but the main optimisation was to use a linked-rule index. Rules are indexed according to their atom patterns such that, given a triple, the index can retrieve all rules containing an atom for which that triple is a ground instantiation—i.e., the index maps triples to relevant rules. Also, the rules are linked according to their dependencies, whereby the grounding of a head in one rule may lead to the application of another—in such cases, explicit links are materialised to reduce the amount of rule-index lookups required.

7.2. *Distributed reasoning*

Assuming that our rules only contain zero/one A-atoms, and that the T-Box is relatively small, distribution of the procedure is straight-forward [38]. We summarise the approach as follows:

(i) **run/gather:** identify and separate out the T-Box from the main corpus in parallel on the slave machines, and subsequently merge the T-Box on the master machine;
(ii) **run:** apply axiomatic and "T-Box only" rules on the master machine, ground the T-atoms in rules with non-empty T-body/A-body, and build the linked rule index;
(iii) **flood/run:** send the linked rule index to all slave machines, and reason over the main corpus in parallel on each machine.

The results of this operation are: (a) data inferred through T-Box only reasoning resident on the master machine; (b) data inferred through A-Box level reasoning resident on the slave machines.

[32] In previous works we have discussed and implemented techniques for scalable processing of rules with multiple A-atoms; however, these techniques were specific to a given ruleset [36]—as are related techniques presented in the literature [62]—and again break the linear bound on materialisation wrt. the A-Box.

7.3. *Extending with annotations*

We now discuss extension of the classical reasoning engine (described above) to support the annotated reasoning task $\mathtt{opt}_{\mathbf{t}}(P)$ for our annotation domain which we have shown to be theoretically scalable in Section 4. As mentioned at the end of Section 4.3, for our experiments we assume a threshold: $\mathbf{t} := \langle \mathbf{nb}, \mathbf{a}, 0 \rangle = \langle \top_1, \top_2, \bot_3 \rangle$. Thus, our reasoning task becomes $\mathtt{opt}_{\mathbf{t}}(P)$. By Theorem 8, we can filter dominated facts/rules; by Theorem 9, we can immediately filter any facts/rules which are blacklisted/non-authoritative. Thus, in practice, once blacklisted and non-authoritative facts/rules have been removed from the program, we need only maintain ranking annotations.

Again, currently, we do not use the blacklisting annotation. In any case, assuming a threshold for non-blacklisted annotations, blacklisted rules/facts in the program could simply be filtered in a pre-processing step.

For the purposes of the authoritativeness annotation, the authority of individual terms in ground T-atoms are computed during the T-Box extraction phase by the slave machines using redirects information from the crawl. This intermediary *term-level authority* is then used by the master machine to annotate T-ground rules with the final authoritative annotation. Recall that all initial ground facts and proper rules $\mathcal{O}2\mathcal{R}^-$ are annotated as authoritative, and that only T-ground rules with non-empty ABody can be annotated as **na**, and subsequently that ground atoms can only be annotated with **na** if produced by such a non-authoritative rule; thus, wrt. our threshold $\mathbf{t}$, we can filter any T-ground rules annotated **na** when first created on the master machine—thereafter, only **a** annotations can be encountered.

Thus, we are left to consider rank annotations. In fact, following the discussion of Section 4.3 and the aforementioned thresholding, the extension is fairly straightforward. All axiomatic triples (mandated by OWL 2 RL/RDF), and all initial (meta-)rules are annotated with $\top$ (the value 1). All other ground facts in the corpus are pre-assigned a rank annotation by the links-based ranking procedure.

The first step involves extracting and reasoning over the rank-annotated T-Box. Terminological data are extracted in parallel on the slave machines from the ranked corpus. These data are gathered onto the master machine. Ranked axiomatic and terminological facts are used for annotated T-Box level reasoning: internally, we store annotations using a map (alongside the triple store itself), and the semi-naïve evaluation only considers unique or non-dominated annotated facts in the delta. Inferred annotations are computed using the glb function as described in Definition 3.

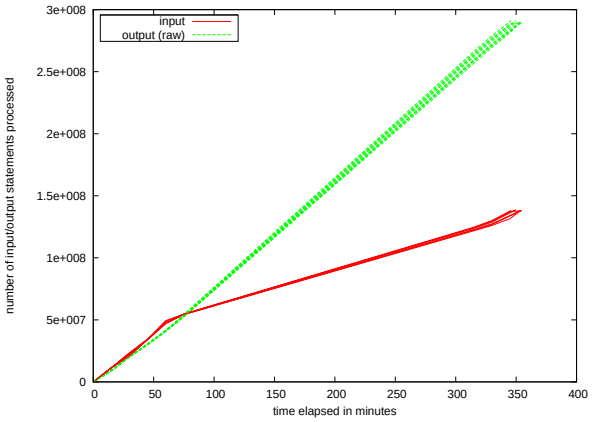

Fig. 4. Input/output throughput during distributed A-Box reasoning overlaid for each slave machine

Next, rules with non-empty ABody are T-ground. If TBody is not empty, the T-ground rule annotation is again given by the glb aggregation of the T-ground instance annotations; otherwise, the annotation remains $\top$. These annotated rules are flooded by the master machine to all slave machines, who then begin the A-Box reasoning procedure.

Since our T-ground rules now only contain a single A-atom, during A-Box reasoning, the glb function takes the annotation of the instance of the A-atom and the annotation of the T-ground rule to produce the inferred annotation. For the purposes of duplicate removal, our LRU cache again considers facts with dominated annotations as duplicate.

Finally, since our A-Box reasoning procedure is not semi-naïve and can only perform partial duplicate detection, we may have duplicates or dominated facts in the output. To ensure optimal output—and thus achieve $\mathtt{opt}_{\mathbf{t}}(P)$—we must finally:

(iv) **coordinate** inferred data on the slave machines by sorting and hashing on subject; also include the T-Box reasoning results resident on the master machine;

(v) **run** an on-disk merge-sort of ranked input and inferred segments on all slave machines, filtering dominated data and streaming the final output.

7.4. *Reasoning evaluation*

In total, applying the distributed reasoning procedure with eight slave machines and one master machine

over the full 1.118b quadruple corpus—including aggregation of the final results—took 14.6hrs. The bulk of time was consumed as follows: (i) extracting the T-Box in parallel and gathering the data onto the master machine took 53mins; (ii) locally reasoning over the T-Box and grounding the T-atoms of the rules with non-empty ABody took 16mins; (iii) parallel reasoning over the A-Box took 6hrs; (iv) sorting and coordinating the inferred data by subject over the slave machines took 4.6hrs; (v) merge-sorting the inferred and input data and aggregating the rank annotations for all triples to produce the final optimal output (above the given threshold) took 2.7hrs. Just over half the total time is spent in the final aggregation of the optimal reasoning output. In Figure 4, we overlay the input/output performance of each slave machine during the A-Box reasoning scan—notably, the profile of each machine is very similar.

In Figure 5, we present the high-level results of applying distributed reasoning for varying corpus sizes and number of slave machines (as per Table 1). For reference, in Table 4 we present the factors by which the runtimes changed when doubling the size of the corpus and when doubling the number of slave machines. We note that:

– although doubling the amount of data from $\frac{\mathsf{c}}{8}$ to $\frac{\mathsf{c}}{4}$ causes the runtimes to (on average) roughly double, moving from $\frac{\mathsf{c}}{4}$ to $\frac{\mathsf{c}}{2}$ and $\frac{\mathsf{c}}{2}$ to c cause runtimes to increase by (on average) 2.2×;
– when doubling the number of slave machines from 1sm to 2sm, runtimes only decrease by (on average) a factor of 0.6×; for 4sm to 2sm and 2sm to 1sm, the factor falls to 0.55× and stays steady.

doubling data				**doubling slaves**			
sm	$\mathsf{c}/\frac{\mathsf{c}}{2}$	$\frac{\mathsf{c}}{2}/\frac{\mathsf{c}}{4}$	$\frac{\mathsf{c}}{4}/\frac{\mathsf{c}}{8}$	c	2sm/1sm	4sm/2sm	8sm/4sm
1sm	2.14	2.23	2.11	c	0.63	0.54	0.53
2sm	2.28	2.25	2.06	$\frac{\mathsf{c}}{2}$	0.59	0.55	0.54
4sm	2.25	2.24	2.03	$\frac{\mathsf{c}}{4}$	0.59	0.55	0.56
8sm	2.21	2.16	1.94	$\frac{\mathsf{c}}{8}$	0.61	0.56	0.59
mean	2.22	2.22	2.03	**mean**	0.60	0.55	0.55

Table 4
On the left, we present the factors by which the annotated reasoning total runtimes changed when doubling the data. On the right, we present the factors by which the runtimes changed when doubling the number of slave machines.

With respect to the first observation, we note that as the size of the corpus increases, so too does the amount of terminological data, leading to an increasingly large assertional program. We found 2.04m, 1.93m, 1.71m and 1.533m T-Box statements in the c, $\frac{\mathsf{c}}{2}$, $\frac{\mathsf{c}}{4}$ and $\frac{\mathsf{c}}{8}$ corpora respectively, creating 291k, 274k, 230k and 199k assertional rules respectively. Here, we note that the rate of growth of the terminological data slows down as the corpus sizes gets larger (i.e., the crawl gets further), perhaps since vocabularies on the Web of Data are well interlinked (cf. Table 3) and will be discovered earlier in the crawl. Further—and perhaps relatedly—we note that the total amount of inferences and the final aggregated output of the annotated reasoning process grows at a rate of between 2.14× and 2.22× when the corpus is doubled in size: we observed 1,889m, 857m, 385m and 180m aggregated (optimal) output facts for the c, $\frac{\mathsf{c}}{2}$, $\frac{\mathsf{c}}{4}$ and $\frac{\mathsf{c}}{8}$ corpora respectively.

With respect to the second observation, in Figure 6, we give the breakdown of the runtimes for the individual tasks when applied over the full corpus using one, two, four and eight slave machines. We note that the percentage of total runtime spent on the master machine is between 0.5% for 1sm and 3.4% for 8sm (∼19.2mins); we observe that for our full corpus, the number of machines can be doubled approximately a further five times (∼256 machines) before the processing on the master machine takes >50% of the runtime. Although the time taken for extracting the T-Box roughly halves every time the number of slaves doubles, we note that the runtimes for inferencing, sorting, hashing and merging do not: in particular, when moving from 1sm to 2sm, we note that A-Box reasoning takes 37% less time; hashing, sorting and scattering the inferences takes 26% less time; and merge-sorting and removing dominated facts takes 43% less time. We attribute this to the additional duplicates created when inferencing over more machines: reasoning over the A-Box of the full corpus generates 1.349b 1.901b, 2.179b and 2.323b *raw* inferences for 1sm, 2sm, 4sm and 8sm, respectively. Note that we use a fixed-size LRU cache of 50k for removing duplicate/-dominated facts, and since our data are sorted by subject, we would not only expect duplicate inferences for each common subject group, but also for data grouped together under the same namespace, where having all of the data on fewer machines increases the potential for avoiding duplicates. Thus, when increasing the number of machines, there are more data to post-process (although the results of that post-processing are the same), but we would expect this effect to converge as the number of machines continues to increase (as perhaps evidenced by Figure 6 where runtimes almost halve for doubling from 2sm to 4sm and from 4sm to 8sm).

For the full corpus, a total of 1.1m (∼0.1%) input T-Box triples were extracted, where T-Box level reasoning produced an additional 2.579m statements. The average rank annotation of the input T-facts was 9.94×10^{-4}, whereas the average rank annotation of the reasoned T-facts was 3.67×10^{-5}. Next, 291k T-ground rules were produced for subsequent application over the A-Box,

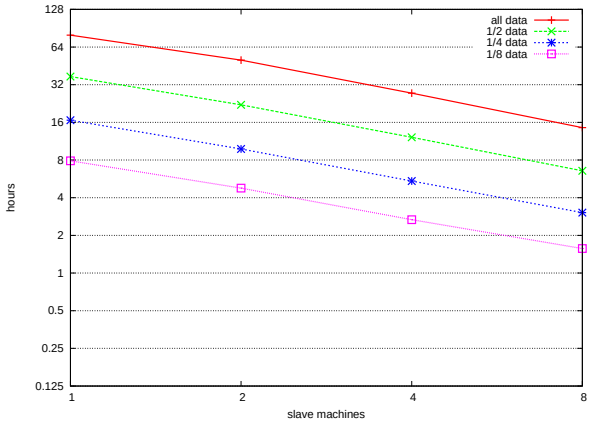

Fig. 5. Total time taken for reasoning on 1/2/4/8 slave machines for varying corpus sizes

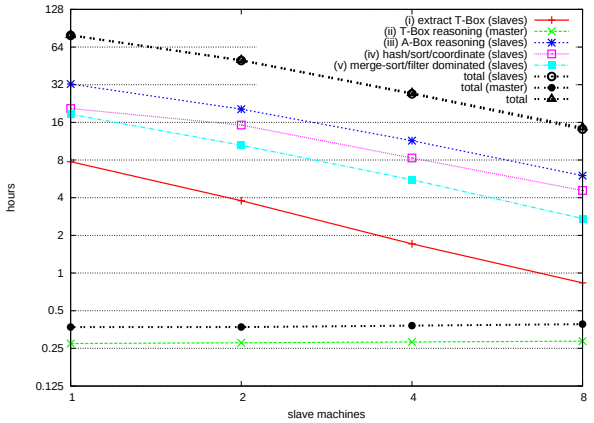

Fig. 6. Breakdown of relative reasoning task times for 1/2/4/8 slave machines and full corpus

within which, 1.655m dependency links were materialised.

Next, in order to demonstrate the effects of authoritative reasoning wrt. our rules and corpus, we applied reasoning over the top ten asserted classes and properties.[33] For each class c, we performed reasoning—wrt. the T-ground program and the authoritatively T-ground program—over a single assertion of the form (`x`, `rdf:type`, `c`) where `x` is an arbitrary unique name; for each property p, we performed the same over a single assertion of the form (`x1`, `p`, `x2`). Subsequently, we only count inferences mentioning an individual name `x*`—e.g., we would not count (`foaf:Person`, `owl:sameAs`, `foaf:Person`). Table 6 gives the results (cf. older results in [36]). Notably, the non-authoritative inference sizes are on average 55.46× larger than the authoritative equivalent. Much of this is attributable to noise in and around core RDF(S)/OWL terms;[34] thus, in the table we also provide results for the core top-level concepts (`rdfs:Resource` and `owl:Thing`) and `rdf:type`, and provide equivalent counts for inferences not relating to these concepts—still, for these popular terms, non-authoritative inferencing creates 12.74× more inferences than the authoritative equivalent.

We now compare authoritative and non-authoritative inferencing in more depth for the most popular class in our input data: `foaf:Person`. Excluding the top-level concepts `rdfs:Resource` and `owl:Thing`, and the inferences possible therefrom, each `rdf:type` triple with `foaf:Person` as value leads to five authoritative inferences and twenty-six *additional* non-authoritative inferences (all class memberships). Of the latter twenty-six, fourteen are anonymous classes. Table 5 enumerates the five authoritatively-inferred class memberships and the remaining twelve non-authoritatively inferred *named* class memberships; also given are the occurrences of the class as a value for `rdf:type` in the raw data. Although we cannot claim that all of the additional classes inferred non-authoritatively are *noise*—although classes such as `b2r2008:Controlled_vocabularies` appear to be—we can see that they are infrequently used and arguably *obscure*. Although some of the inferences we omit may of course be serendipitous—e.g., perhaps `po:Person`—again we currently cannot distinguish such cases from noise or blatant spam.

Overall, performing A-Box reasoning over the full corpus using all eight slave machines produced 2.323b raw (authoritative) inferences, which were immediately filtered down to 1.879b inferences (80.9%) by removing non-RDF and *tautological triples* (reflexive `owl:sameAs`, `rdf:type owl:Thing`, `rdf:type rdfs:Resource` statements which hold for every ground term in the graph which are not useful in our use-case SWSE). Of the 1.879b inferences, 1.866b (99.34%) inherited their annotation from an assertional fact (as opposed to a T-ground rule), seemingly since terminological facts are generally more highly ranked by our approach than assertional facts. Notably, the LRU cache detected and filtered a total of 12.036b duplicate/dominated statements.

After hashing on subject, for each slave machine, the average absolute deviation from the mean was 203.9k

[33] We performed a count of the occurrences of each term as a value for `rdf:type` (class membership) or predicate position (property membership) in the quadruples of our input corpus.

[34] We note that much of the noise is attributate to the `opencalais.com` domain; cf. `http://d.opencalais.com/1/type/em/r/PersonAttributes.rdf` and `http://groups.google.com/group/pedantic-web/browse_thread/thread/5e5bd42a9226a419`

Class	(Raw) Count
Authoritative	
foaf:Agent	8,165,989
wgs84:SpatialThing	64,411
contact:Person	1,704
dct:Agent	35
contact:SocialEntity	1
Non-Authoritative (additional)	
po:Person	852
wn:Person	1
aifb:Kategorie-3AAIFB	0
b2r2008:Controlled_vocabularies	0
foaf:Friend_of_a_friend	0
frbr:Person	0
frbr:ResponsibleEntity	0
pres:Person	0
po:Category	0
sc:Agent_Generic	0
sc:Person	0
wn:Agent-3	0

Table 5
Breakdown of non-authoritative and authoritative inferences for foaf:Person, with number of appearances as a value for rdf:type in the raw data

triples (0.08% of the mean) representing near-optimal data distribution. In the final aggregation of rank annotations, from a total of 2.987b raw statements, 1.889b (63.2%) unique and optimal triples were extracted; of the filtered, 1.008b (33.7%) were duplicates with the same annotation,[35] 89m were (properly) dominated reasoned triples (2.9%), and 1.5m (0.05%) were (properly) dominated asserted triples. The final average rank annotation for the aggregated triples was 5.29×10^{-7}.

8. Use-case: Repairing inconsistencies—approach, implementation and evaluation

In this section, we discuss our approach and implementation for handling inconsistencies in the annotated corpus (including asserted and reasoned data). We begin by formally sketching the repair process applied (Section 8.1). Thereafter, we describe our distributed implementation for detecting and extracting sets of annotated facts which together constitute an inconsistency (Section 8.2) and present the results of applying this process over our Linked Data evaulation corpus (Section 8.3). We then present our distributed implementation of the inconsistency repair process (Section 8.4) and the results for our corpus (Section 8.5).

[35] Note that this would include the 171 million duplicate asserted triples from the input.

8.1. *Repairing inconsistencies: formal approach*

Given a (potentially large) set of annotated constraint violations, herein we sketch an approach for repairing the corpus from which they were derived, such that the result of the repair is a consistent corpus as defined in Section 4.7.[36] In particular, we re-use notions from the seminal work of Reiter [55] on diagnosing faulty systems.

For the moment—and unlike loosely related works on debugging unsatisfiable concepts in OWL terminologies—we only consider repair of assertional data: all of our constraints involve some assertional data, and for the moment, we consider terminological data as correct. Although this entails the possibility of removing atoms above the degree of a particular violation to repair that violation, in Section 7.4 we showed that for our corpus, 99.34% of inferred annotations are derived from an assertional fact.[37] Thus, as an approximation, we can reduce our repair to being wrt. the T-ground program $P := P^c \cup P^r$, where P^c is the set of (proper) T-ground constraint rules, and P^r is the set of proper T-ground positive rules derived in Section 7. Again, given that each of our constraints requires assertional knowledge—i.e., that the T-ground program P only contains *proper* constraint rules—P is necessarily consistent.

Moving forward, we introduce some necessary definitions adapted from [55] for our scenario. Firstly, we give:

> **The Principle of Parsimony:** A diagnosis is a conjecture that some minimal set of components are faulty [55].

This captures our aim to find a non-trivial (minimal) set of assertional facts which diagnose the inconsistency of our model. Next, we define a *conflict set* which denotes a set of inconsistent facts, and give a *minimal conflict set* which denotes the least set of facts which preserves an inconsistency wrt. a given program P:

Definition 7 (Conflict set) *A* conflict set *is a Herbrand interpretation $C := \{F_1, \ldots, F_n\}$ such that $P \cup C$ is inconsistent.*

Definition 8 (Minimal conflict set) *A* minimal conflict set *is a Herbrand interpretation $C := \{F_1, \ldots, F_n\}$*

[36] Note that a more detailed treatment of repairing inconsistencies on the Web is currently out of scope, and would deserve a more dedicated analysis in future work. Herein, our aim is to sketch one particular approach feasible in our scenario.

[37] Note also that since our T-Box is also part of our A-Box, we may defeat facts which are terminological, but only based on inferences possible through their assertional interpretation.

Term	n	**a**	n * **a**	**a**$^-$	n * **a**$^-$	**na**	n * **na**	**na**$^-$	n * **na**$^-$
Core classes (∼top-level concepts)									
rdfs:Resource	12,107	0	0	0	0	108	1,307,556	0	0
owl:Thing	679,520	1	679,520	0	0	109	74,067,680	0	0
Core property									
rdf:type	206,799,100	1	206,799,100	0	0	109	22,541,101,900	0	0
Top ten asserted classes									
foaf:Person	163,699,161	6	982,194,966	5	818,495,805	140	22,917,882,540	31	5,074,673,991
foaf:Agent	8,165,989	2	16,331,978	1	8,165,989	123	1,004,416,647	14	114,323,846
skos:Concept	4,402,201	5	22,011,005	3	13,206,603	115	506,253,115	6	26,413,206
mo:MusicArtist	4,050,837	1	4,050,837	0	0	132	534,710,484	23	93,169,251
foaf:PersonalProfileDocument	2,029,533	2	4,059,066	1	2,029,533	114	231,366,762	5	10,147,665
foaf:OnlineAccount	1,985,390	2	3,970,780	1	1,985,390	110	218,392,900	1	1,985,390
foaf:Image	1,951,773	1	1,951,773	0	0	110	214,695,030	1	1,951,773
opiumfield:Neighbour	1,920,992	1	1,920,992	0	0	109	209,388,128	0	0
geonames:Feature	983,800	2	1,967,600	1	938,800	111	109,201,800	2	1,967,600
foaf:Document	745,393	1	745,393	0	0	113	84,229,409	4	2,981,572
Top ten asserted properties (after rdf:type)									
rdfs:seeAlso	199,957,728	0	0	0	0	218	43,590,784,704	0	0
foaf:knows	168,512,114	14	2,359,169,596	12	2,022,145,368	285	48,025,952,490	67	11,290,311,638
foaf:nick	163,318,560	0	0	0	0	0	0	0	0
bio2rdf:linkedToFrom	31,100,922	0	0	0	0	0	0	0	0
lld:pubmed	18,776,328	0	0	0	0	0	0	0	0
rdfs:label	14,736,014	0	0	0	0	221	3,256,659,094	3	44,208,042
owl:sameAs	11,928,308	5	59,641,540	1	11,928,308	221	2,636,156,068	3	35,784,924
foaf:name	10,192,187	5	50,960,935	2	20,384,374	256	2,609,199,872	38	387,303,106
foaf:weblog	10,061,003	8	80,488,024	5	50,305,015	310	3,118,910,930	92	925,612,276
foaf:homepage	9,522,912	8	76,183,296	5	47,614,560	425	4,047,237,600	207	1,971,242,784
total	**1,035,531,872**	**65**	**3,873,126,401**	**37**	**2,997,244,745**	**3,439**	**155,931,914,709**	**497**	**19,982,077,064**

Table 6

Summary of authoritative inferences vs. non-authoritative inferences for core properties, classes, and top-ten most frequently asserted classes and properties: given are the number of asserted memberships of the term n, the number of unique inferences (which mention an "individual name") possible for an arbitrary membership assertion of that term wrt. the authoritative T-ground program (**a**), the product of the number of assertions for the term and authoritative inferences possible for a single assertion (n * **a**), respectively, the same statistics excluding inferences involving top-level concepts (**a**$^-$ / n * **a**$^-$), statistics for non-authoritative inferencing (**na** / n * **na**) and also non-authoritative inferences minus inferences through a top-level concept (**na**$^-$ / n * **na**$^-$)

such that $P \cup C$ is inconsistent, and for every $C' \subset C$, $P \cup C'$ is consistent.

Next, we define the notions of a hitting set and a minimal hitting set as follows:

Definition 9 (Hitting set) *Let $\mathcal{I} := \{I_1, \ldots, I_n\}$ be a set of Herbrand interpretations, and $H := \{F_1, \ldots, F_n\}$ be a single Herbrand interpretation. Then, H is a* hitting set *for $\mathcal{I}$ iff for every $I_j \in \mathcal{I}$, $H \cap I_j \neq \emptyset$.*

Definition 10 (Minimal hitting set) *A* minimal hitting set *for $\mathcal{I}$ is a hitting set H for $\mathcal{I}$ such that for every $H' \subset H$, H' is not a hitting set for $\mathcal{I}$.*

Given a set of minimal conflict sets $\mathcal{C}$, the set of corresponding minimal hitting sets $\mathcal{H}$ represents a set of diagnoses thereof [55]; selecting one such minimal hitting set and removing all of its members from each set in $\mathcal{C}$ would resolve the inconsistency for each conflict set $C \in \mathcal{C}$ [55].

This leaves three open questions: (i) how to compute the minimal conflict sets for our reasoned corpus; (ii) how to compute and select an appropriate hitting set as the diagnosis of our inconsistent corpus; (iii) how to repair our corpus wrt. the selected diagnosis.

8.1.1. *Computing the (extended) minimal conflict sets*

In order to compute the set of minimal conflict sets, *we leverage the fact that the program P^r does not contain rules with multiple A-atoms in the body.* We leave extension of the repair process for programs with multiple A-atoms in the body for another scope.

First, we must consider the fact that our corpus Γ already represents the least model of $\Gamma \cup P^r$ and define an extended minimal conflict set as follows:

Definition 11 (Extended minimal conflict set) *Let Γ be a Herbrand model such that $\Gamma := \mathrm{lm}(\Gamma \cup P^r)$, and let $C := \{F_1, \ldots, F_n\}, C \subseteq \Gamma$ denote a minimal conflict set for Γ. Let*

$$E(F) \coloneqq \{F' \in \Gamma \mid F \in \mathrm{lm}(P^r \cup \{F'\})\}$$

be the set of all facts in Γ *from which some* F *can be derived wrt. the linear program* P^r *(note* $F \in E(F)$*). We define the* extended minimal conflict set *(EMCS) for* C *wrt.* Γ *and* P^r *as* $\mathcal{E} \coloneqq \{E(F)|F \in C\}$.

Thus, given a minimal conflict set, the extended minimal conflict set encodes choices of sets of facts that must be removed from the corpus Γ to repair the violation, such that the original *seed fact* cannot subsequently be re-derived by running the program P^r over the reduced corpus. The concept of a (minimal) hitting set for a collection of EMCSs follows naturally and similarly represents a diagnosis for the corpus Γ.

8.1.2. *Preferential strategies for annotated diagnoses*

Before we continue, we discuss two competing models for deciding an appropriate diagnosis for subsequent reparation of the annotated corpus. Consider a set of violations that could be solved by means of removing one 'strong' fact—e.g., a single fact associated with a highly-ranked document—or removing many weak facts—e.g., a set of facts derived from a number of low-ranked documents: should one remove the strong fact or the set of weak facts? Given that the answer is non-trivial, we identify two particular means of deciding a suitable diagnosis: i.e., the characteristics of an appropriate minimal hitting set wrt. to our annotations. Given any such strategy, selecting the most appropriate diagnosis then becomes an optimisation problem.

Strategy 1: we prefer a diagnosis which minimises the number of facts to be removed in the repair. This can be applied independently of the annotation framework. However, this diagnosis strategy will often lead to trivial decisions between elements of a minimal conflicting set with the same cardinality; also, we deem this strategy to be vulnerable to spamming such that a malicious low-ranked document may publish a number of facts which conflict and defeat a fact in a high-ranked document. Besides spamming, in our repair process, it may also trivially favour, e.g., memberships of classes which are part of a deep class hierarchy (the memberships of the super-classes would also need to be removed).

Strategy 2: we prefer a diagnosis which minimises the strongest annotation to be removed in the repair. This has the benefit of exploiting the information in the annotations, and being computable with the $\mathrm{glb}/\mathrm{lub}$ functions defined in our annotation framework; however, for general annotations in the domain $\mathbf{D}$ only a *partial-ordering* is defined, and so there may not be an unambiguous strongest/weakest annotation—in our case, with our pre-defined threshold removing blacklisted and non-authoritative inferences from the corpus, we need only consider rank annotations for which a total-ordering is present. Also, this diagnosis strategy may often lead to trivial decisions between elements of a minimal conflicting set with identical annotations—most likely facts from the same document which we have seen to be a common occurrence in our constraint violations (54.1% of the total *raw* cax-dw violations we empirically observe).

Strategy 3: we prefer a diagnosis which minimises the total sum of the rank annotation involved in the diagnosis. This, of course, is domain-specific and also falls outside of the general annotation framework, but will likely lead to less trivial decisions between equally 'strong' diagnoses. In the naïve case, this strategy is also vulnerable to spamming techniques, where one 'weak' document can make a large set of weak assertions which culminate to defeat a 'strong' fact in a more trustworthy source.

In practice, we favour *Strategy 2* as exploiting the additional information of the annotations and being less vulnerable to spamming; when *Strategy 2* is inconclusive, we resort to *Strategy 3* as a more granular method of preference, and thereafter if necessary to *Strategy 1*. If all preference orderings are inconclusive, we then select an arbitrary syntactic ordering.

Going forward, we formalise a total ordering $\leq_I$ over a pair of (annotated) Herbrand interpretations which denotes some ordering of preference of diagnoses based on the 'strength' of a set of facts—a stronger set of facts denotes a higher order. The particular instantiation of this ordering depends on the repair strategy chosen, which may in turn depend on the specific domain of annotation.

Towards giving our notion of $\leq_I$, let I_1 and I_2 be two Herbrand interpretations with annotations from the domain $\mathbf{D}$, and let $\leq_{\mathbf{D}}$ denote the partial-ordering defined for $\mathbf{D}$. Starting with *Strategy 2*—slightly abusing notation—if $\mathrm{lub}\{I_1\} <_{\mathbf{D}} \mathrm{lub}\{I_1\}$, then $I_1 <_r I_2$; if $\mathrm{lub}\{I_1\} >_{\mathbf{D}} \mathrm{lub}\{I_2\}$, then $I_1 >_I I_2$; otherwise (if $\mathrm{lub}\{I_1\} =_{\mathbf{D}} \mathrm{lub}\{I_2\}$ or $\leq_{\mathbf{D}}$ is undefined for I_1,I_2), we resort to *Strategy 3* to order I_1 and I_2: we apply a domain-specific "summation" of annotations (ranks) denoted $\Sigma_{\mathbf{D}}$ and define the order of I_1, I_2 such that if $\Sigma_{\mathbf{D}}\{I_1\} <_{\mathbf{D}} \Sigma_{\mathbf{D}}\{I_2\}$, then $I_1 <_I I_2$, and so forth. If still equals (or uncomparable), we use the cardinality of the sets, and thereafter consider an arbitrary syntactic order. Thus, sets are given in ascending order of their single strongest fact, followed by the order of their rank summation, followed by their cardinality (*Strategy 1*), followed by an arbitrary syntactic ordering. Given $<_I$, the functions $\max_I$ and $\min_I$ follow naturally.

8.1.3. *Computing and selecting an appropriate diagnosis*

Given that in our Linked Data scenario, we may encounter many non-trivial (extended) conflict sets—i.e., conflict sets with cardinality greater than one—we would wisely wish to avoid materialising an exponential number of all possible hitting sets. Similarly, we wish to avoid expensive optimisation techniques [59] for deriving the minimal diagnosis w.r.t $\leq_I$. Instead, we use a heuristic to materialise one hitting set which gives us an *appropriate*, but possibly sub-optimal diagnosis. Our diagnosis is again a flat set of facts, which we denote by Δ.

First, to our diagnosis we immediately add the union of all members of singleton (trivial) EMCSs, where these are necessarily part of any diagnosis. This would include, for example, all facts which must be removed from the corpus to ensure that no violation of dt-not-type can remain or be re-derived in the corpus.

For the non-trivial EMCSs, we first define an ordering of conflict sets based on the annotations of its members, and then cumulatively derive a diagnosis by means of a descending iteration of the ordered sets For the ordered iteration of the EMCS collection, we must define a total ordering $\leq_{\mathcal{E}}$ over $\mathcal{E}$ which directly corresponds to $\min_I(E_1) \leq_I \min_I(E_2)$—a comparison of the weakest set in both.

We can then apply the following diagnosis strategy: iterate over $\mathcal{E}$ in descending order wrt. $\leq_{\mathcal{E}}$, such that for each E:

$$\text{if } \nexists I \in E \text{ such that } I \subseteq \Delta \text{ then } \Delta := \Delta \cup \min_I(E)$$

where after completing the iteration, the resulting Δ represents our diagnosis. Note of course that Δ may not be optimal according to our strategy, but we leave further optimisation techniques to future work.

8.1.4. *Repairing the corpus*

Removing the diagnosis Δ from the corpus Γ will lead to consistency in $P \cup (\Gamma \setminus \Delta)$. However, we also wish to remove the facts that are inferable through Δ with respect to P, which we denote as Δ^+. We also want to identify facts in Δ^+ which have alternative derivations from the non-diagnosed input data ($\Gamma_{raw} \setminus \Delta$), and include them in the repaired output, possibly with a weaker annotation: we denote this set of re-derived facts as Δ^-. *Again, we sketch a scan-based approach which is contingent on P only containing proper rules with one atom in the body, as is the case for any assertional program derived from $\mathcal{O}2\mathcal{R}^-$.*

First, we determine the set of statements inferable from the diagnosis, given as:

$$\Delta^+ := \text{lm}(P \cup \Delta) \setminus \Delta\,.$$

Secondly, we scan the *raw* input corpus Γ_{raw} as follows. First, let $\Delta^- := \{\}$. Let

$$\Gamma^{\not\Delta}_{raw} := \{F{:}\mathbf{d} \in \Gamma_{raw} \mid \nexists \mathbf{d}'(F{:}\mathbf{d}' \in \Delta)\}$$

denote the set of annotated facts in the raw input corpus not appearing in the diagnosis. Then, scanning the raw input (ranked) data, for each $F_i \in \Gamma^{\not\Delta}_{raw}$, let

$$\begin{aligned}\delta_i^- :=&\{F'{:}\mathbf{d}' \in \text{lm}(P \cup \{F_i{:}\mathbf{d}_i\}) \mid \\ &\exists \mathbf{d}_x(F'{:}\mathbf{d}_x \in \Delta^+), \nexists \mathbf{d}_y \geq \mathbf{d}'(F'{:}\mathbf{d}_y \in \Delta^-)\}\end{aligned}$$

denote the intersection of facts derivable from both F_i and Δ^+ which are not dominated by a previous rederivation; we apply $\Delta_i^- := \max(\Delta_{i-1}^- \cup \delta_i^-)$, maintaining the dominant set of rederivations. After scanning all raw input facts, the final result is Δ^-:

$$\Delta^- = \max\{F{:}\mathbf{d} \in \text{lm}(P \cup \Gamma^{\not\Delta}_{raw}) \mid \exists \mathbf{d}'(F{:}\mathbf{d}' \in \Delta^+)\}$$

the dominant rederivations of facts in Δ^+ from the non-diagnosed facts of the input corpus.

Finally, we scan the entire corpus Γ and buffer any facts not in $\Delta \cup \Delta^+ \setminus \Delta^-$ to the final output, and if necessary, weaken the annotations of facts to align with Δ^-.

We note that if there are terminological facts in Δ, the T-Box inferences possible through these facts may remain in the final corpus, even though the corpus is consistent. However, such cases are rare in practice. For a T-Box statement to be defeated in our repair, it would have to cause an inconsistency in a "punned" A-Box sense: that is to say, a core RDFS or OWL meta-class or property would have to be involved in an axiom—e.g., a disjointness constraint—which could lead to inconsistency. However, such constraints are not authoritatively specified for core RDFS or OWL terms, and so cannot be encountered in our approach. One possible case which *can* occur is a T-Box statement containing an ill-typed literal, but we did not encounter such a case for our corpus and selected ruleset. If required, removal of all such T-Box inferences would instead require rerunning the entire reasoning process over $\Gamma_{raw} \setminus \Delta$—the repaired raw corpus. [38]

8.2. *Inconsistency detection: implementation*

We now begin discussing our implementation and evaluation of this repair process, beginning in this sec-

[38] We note that other heuristics would be feasible, such as "pre-validating" the T-Box for the *possibility* of inconsistency, but leave such techniques as an open question.

tion with the distributed implementation of inconsistency detection for our scenario.

In Table A.6, we provide the list of OWL 2 RL/RDF constraint rules which we use to detect inconsistencies. The observant reader will note that these rules require A-Box joins (have multiple A-atoms) which we have thus far avoided in our approach. However, up to a point, we can leverage a similar algorithm to that presented in Section 7.1 for reasoning.

First, we note that the rules are by their nature not recursive (have empty heads). Second, we postulate that many of the ungrounded atoms will have a high selectivity (have a low number of ground instances in the knowledge-base)—in particular, for the moment we *partially assume* that only one atom in each constraint rule has a low selectivity. When this assumption holds, the rules are amenable to computation using the partial-indexing approach: any atoms with high selectivity are ground in-memory to create a set of (partially) grounded rules, which are subsequently flooded across all machines. Since the initial rules are not recursive, the set of (partially) grounded rules remains static. Assuming that at most one pattern has low selectivity, we can efficiently apply our single-atom rules in a distributed scan as per the previous section.

However, the second assumption may not always hold: a constraint rule may contain more than one low-selectivity atom. In this case, we manually apply a distributed on-disk merge-join operation to ground the remaining atoms of such rules.

Note that during the T-Box extraction in the previous section, we additionally extract the T-atoms for the constraint rules, and apply authoritative analysis analogously—we briefly give an example.

Example: Again take cax-dw as follows:

$\leftarrow$ (?c$_1$, owl:disjointWith, ?c$_1$), (?x, a, ?c$_1$), (?x, a, ?c$_2$)

Here, $\mathsf{vars}^{TA}(\text{cax-dw}) = \{\texttt{?c1},\texttt{?c2}\}$. Further assume that we have the triple (foaf:Person, owl:disjointWith, geoont:TimeZone) given by a source s. For the respective T-ground rule:

$\leftarrow$ (?x, a, foaf:Person), (?x, a, geoont:TimeZone)

to be authoritative, s must correspond to either deref(foaf:Person) or deref(geoont:TimeZone)—it must be authoritative for at least one T-substitution of $\{\texttt{?c1},\texttt{?c2}\}$.[39] ◇

Then, the process of extracting constraint violations is as follows:

(i) **local:** apply an authoritative T-grounding of the constraint rules in Table A.6 from the T-Box resident on the master machine;
(ii) **flood/run:** flood the non-ground A-atoms in the T-ground constraints to all slave machines, which extract the selectivity (number of ground instances) of each pattern for their segment of the corpus, and locally buffer the instances to a separate corpus;
(iii) **gather:** gather and aggregate the selectivity information from the slave machines, and for each T-ground constraint, identify A-atoms with a selectivity below a given threshold (in-memory capacity);
(iv) ***reason:*** for rules with zero or one low-selectivity A-atoms, run the distributed reasoning process described in Section 7.2, where the highly-selective A-atoms can be considered "T-atoms";
(v) **run(/coordinate):** for any constraints with more than one low-selectivity A-atoms, apply a manual on-disk merge-join operation to complete the process.

The end result of this process is sets of annotated atoms constituting constraint violations distributed across the slave machines.

8.3. *Inconsistency detection: evaluation*

Distributed extraction of the inconsistencies from the aggregated annotated data took, in total, 2.9hrs. Of this: (i) 2.6mins were spent building the authoritative T-ground constraint rules from the local T-Box on the master machine; (ii) 26.4mins were spent extracting—in parallel on the slave machines—the cardinalities of the A-atoms of the T-ground constraint bodies from the aggregated corpus; (iii) 23.3mins were spent extracting ground instances of the high-selectivity A-atoms from the slave machines; (iv) 2hrs were spent applying the partially-ground constraint rules in parallel on the slave machines.

In Figure 7, we present the high-level results of applying inconsistency detection for varying corpus sizes and number of slave machines (as per Table 1). For reference, in Table 7 we present the factors by which the runtimes changed when doubling the size of the corpus and when doubling the number of slave machines. We note that:

– doubling the amount of input data increases the runtime (on average) by a factor of between 2.06 and 2.26 respectively;
– when doubling the number of slave machines from

[39] It corresponds to the latter: http://geoontology.altervista.org/schema.owl#Timezone.

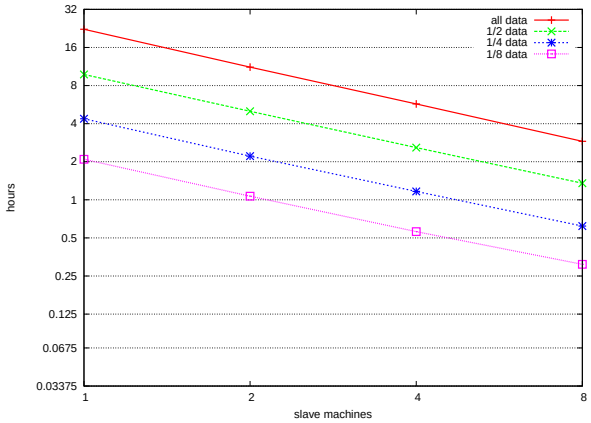

Fig. 7. Total time taken for inconsistency detection on 1/2/4/8 slave machines for varying corpus sizes

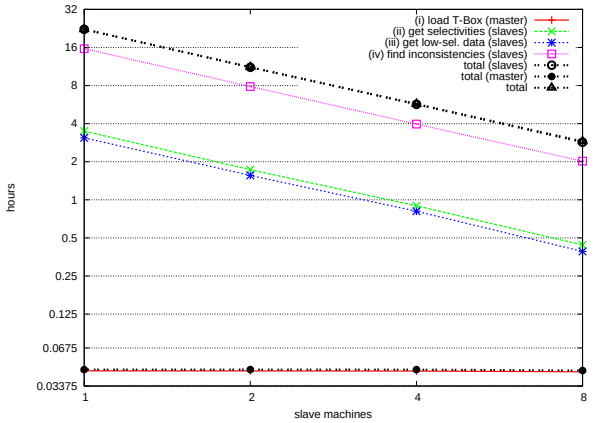

Fig. 8. Breakdown of distributed inconsistency-detection task times for 1/2/4/8 slave machines and full corpus

1sm to 2sm, runtimes approximately halve each time.

doubling data				**doubling slaves**			
sm	c/$\frac{c}{2}$	$\frac{c}{2}$/$\frac{c}{4}$	$\frac{c}{4}$/$\frac{c}{8}$	c	2sm/1sm	4sm/2sm	8sm/4sm
1sm	2.28	2.24	2.09	c	0.50	0.51	0.51
2sm	2.23	2.26	2.07	$\frac{c}{2}$	0.51	0.52	0.52
4sm	2.21	2.22	2.08	$\frac{c}{4}$	0.51	0.53	0.53
8sm	2.14	2.18	2.01	$\frac{c}{8}$	0.51	0.52	0.55
mean	2.22	2.23	2.06	**mean**	0.51	0.52	0.53

Table 7
On the left, we present the factors by which the inconsistency detection total runtimes changed when doubling the data. On the right, we present the factors by which the runtimes changed when doubling the number of slave machines.

With respect to the first observation, this is to be expected given that the outputs of the reasoning process in Section 7.4—which serve as the inputs for this evaluation—are between 2.14× and 2.22× larger for 2× the raw data.

With respect to the second observation, we note that in particular, for the smallest corpus, and doubling the number of machines from four to eight, the runtime does not quite halve: in this case, the master machine takes up 17% of the total runtime with local processing. However, this becomes increasingly less observable for larger corpora where the percentage of terminological data decreases (cf. Secton 7.4): when processing the full corpus, note that the amount of time spent on the master machine was between 0.2% for 1sm and 1.7% for 8sm (∼2.5mins). Further, in Figure 8 we present the breakdown of the individual inconsistency detection tasks for the full corpus using one, two, four and eight slave machines; we observe that for our full corpus, the number of machines can be doubled approximately a further six times (512 machines) before the processing on the master machine takes >50% of the runtime.

Focussing again on the results for the full corpus, a total of 301k constraint violations were found; in Table 8, we give a breakdown showing the number of T-ground rules generated, the number of total ground instances (constraint violations found), and the total number of unique violations found (a constraint may fire more than once over the same data, where for example in rule cax-dw, `?c1` and `?c2` can be ground interchangeably). Notably, the table is very sparse: we highlight the constraints requiring new OWL 2 constructs in italics, where we posit that OWL 2 has not had enough time to gain traction on the Web. In fact, all of the T-ground prp-irp and prp-asyp rules come from one document [40], and all cax-adc T-ground rules come from one directory of documents [41]. The only two constraint rules with violations in our Linked Data corpus were dt-not-type (97.6%) and cax-dw (2.4%).

Overall, the average violation rank degree was 1.19×10^{-7} (vs. an average rank-per-fact of 5.29×10^{-7} in the aggregated corpus). The single strongest violation degree is shown in Listing 1 and was given by dt-not-type where the term `"True"^^xsd:integer` is invalid. In fact, the document serving this triple is ranked 23rd overall out of our 3.985m sources—indeed, it seems that even highly ranked documents are prone to publishing errors and inconsistencies. Similar inconsistencies were also found with similar strengths in other documents within that FreeBase domain. [42] Thus, only a minute

[40] `http://models.okkam.org/ENS-core-vocabulary#country_of_residence`
[41] `http://ontologydesignpatterns.org/cp/owl/fsdas/`
[42] Between our crawl and time of writing, these errors have been fixed.

Listing 1. Strongest constraint violation

```
# ABox Source – http://rdf.freebase.com/rdf/type/key/namespace
# Annotation – <nb,a,0.001616>
 (fb:type.key.namespace, fb:type.property.unique, "True"^^xsd:integer)
```

Listing 2. Strongest disjoint-class violation

```
# TBox Source – foaf: (amongst others)
# Annotations – <nb,a,0.024901>
 (foaf:primaryTopic, rdfs:domain, foaf:Document)
 (foaf:Document, owl:disjointWith, foaf:Agent)

# TBox Source – geospecies:
# Annotation – <nb,a,0.000052>
 (geospecies:hasWikipediaArticle, rdfs:domain, foaf:Person)

# ABox Source – kingdoms:Aa?format=rdf
# Annotations – <nb,a,0.000038>
 (kingdoms:Aa, foaf:PrimaryTopic, kingdoms:Aa)
 (kingdoms:Aa , geospecies:hasWikipediaArticle, enwiki:Animal)

# Violation
# Annotations – <nb,a,0.000038>
 (kingdoms:Aa, rdf:type, foaf:Document) # Inferred
 (kingdoms:Aa, rdf:type, foaf:Person) # Inferred
```

Listing 3. Disjoint-class violation involving strongest fact

```
# TBox Source – foaf: (amongst others)
# Annotations – <nb,a,0.024901>
 (foaf:knows, rdfs:domain, foaf:Person)
 (foaf:Organization, owl:disjointWith, foaf:Person)

# Comment – dav:this refers to the company OpenLink Software
# ABox Source – 5,549 documents including dav:
# Annotation – <nb,a,0.000391>
 (dav:this, rdf:type, foaf:Organization)
# ABox Source – dav: (alone)
# Annotation – <nb,a,0.000020>
 (dav:this, foaf:knows, vperson:kidehen@openlinksw.com#this)

# Violation
# Annotation – <nb,a,0.000391>
 (dav:this, rdf:type, foaf:Organization)
# Annotation – <nb,a,0.000020>
 (dav:this, rdf:type, foaf:Person) # Inferred
```

fraction (∼0.0006%) of our corpus is above the consistency threshold.

With respect to cax-dw, we give the top 10 pairs of disjoint classes in Table 9. The single strongest violation degree for cax-dw is given in Listing 2 where we see that the inconsistency is given by one document, and may be attributable to use of properties without verifying their defined domain; arguably, the entity `kingdoms:Aa` is unintentionally a member of both the FOAF disjoint classes, where the entity is explicitly a member of `geospecies:KingdomConcept`. Taking a slightly different example, the cax-dw violation involving the strongest A-atom annotation is given in Listing 3 where we see a conflict between a statement asserted in thousands of documents, and a statement inferable from a single document.

Rule	T-ground	Violations
eq-diff1	–	0
eq-diff2	–	0
eq-diff3	–	0
eq-irp	–	0
prp-irp	10	0
prp-asyp	9	0
prp-pdw	0	0
prp-adp	0	0
prp-npa1	–	0
prp-npa2	–	0
cls-nothing	–	0
cls-com	14	0
cls-maxc1	0	0
cls-maxqc1	0	0
cls-maxqc2	0	0
cax-dw	1,772	7,114
cax-adc	232	0
dt-not-type	–	294,422

Table 8
Number of T-ground rules, violations, and unique violations found for each OWL 2 RL/RDF constraint rule—rules involving new OWL 2 constructs are *italicised*

Class 1	Class 2	Violations
`foaf:Agent`	`foaf:Document`	3,842
`foaf:Document`	`foaf:Person`	2,918
`sioc:Container`	`sioc:Item`	128
`foaf:Person`	`foaf:Project`	100
`ecs:Group`	`ecs:Individual`	38
`skos:Concept`	`skos:Collection`	36
`foaf:Document`	`foaf:Project`	26
`foaf:Organization`	`foaf:Person`	7
`sioc:Community`	`sioc:Item`	3
`ecs:Fax`	`ecs:Telephone`	3

Table 9
Top 10 disjoint-class pairs

Of the cax-dw constraint violations, 3,848 (54.1%) involved A-atoms with the same annotation (such as in the former cax-dw example—likely stemming from the same A-Box document). All of the constraint violations were given by A-atoms (i.e., an A-atom represented the weakest element of each violation).

8.4. *Inconsistency repair: implementation*

Given the set of inconsistencies extracted from the corpus in the previous section, and the methodology of repair sketched in Section 8.1, we now briefly describe the distributed implementation of the repair process.

To derive the complete collection of EMCSs from our corpus, we apply the following process. Firstly, for each constraint violation extracted in the previous steps, we create and load an initial (minimal) conflict set into memory. From this, we create an extended version rep-

resenting each member of the original conflict set (*seed fact*) by a singleton in the extended set. (Internally, we use a map structure to map from facts to the extended set(s) that contain it, or or course, `null` if no such conflict set exists.) We then reapply P^r over the corpus in parallel, such that—here using notation which corresponds to Algorithm 1—for each input triple t being reasoned over, for each member t_δ of the subsequently inferred set G_n, if t_δ is a member of an EMCS, we add t to that EMCS.

Consequently, we populate the collection of EMCS, where removing all of the facts in one member of each EMCS constitutes a repair (a diagnosis). With respect to distributed computation of the EMCSs, we can run the procedure in parallel on the slave machines, and subsequently merge the results on the master machine to derive the global collection of EMCSs. We can then apply the diagnosis procedure and compute Δ, Δ^+ and Δ^- needed for the repair of the closed corpus.

We now give the overall distributed steps; note that Steps (i), (iii) & (v) involve aggregating and diagnosing inconsistencies and are performed locally on the master machine, whereas Steps (ii), (iv) & (vi) involve analysis of the input and inferred data on the slave machines and are run in parallel:

(i) **gather**: the set of conflict sets (constraint violations) detected in the previous stages of the process (Sections 8.2 & 8.3) are gathered onto the master machine;

(ii) **flood/run**: the slave machines receive the conflict sets from the master machine and reapply the (positive) T-ground program over the *closed* corpus; any triple involved in the inference of a member of a conflict set is added to an extended conflict set;

(iii) **gather**: the respective extended conflict sets are merged on the master machine, and the sets are ordered by $\leq_{\mathcal{E}}$ and iterated over—the initial diagnosis Δ is thus generated; the master machine applies reasoning over Δ to derive Δ^+ and **flood**s this set to the slave machines;

(iv) **flood/run**: the slave machines re-run the reasoning over the *input* corpus to try to find alternate (non-diagnosed) derivations for facts in Δ^-;

(v) **gather**: the set of alternate derivations are gathered and aggregated on the master machine, which prepares the final Δ^- set (maintaining only dominant rederivations in the merge);

(vi) **flood/run**: the slave machines accept the final diagnosis and scan the *closed* corpus, buffering a repaired (consistent) corpus.

8.5. *Inconsistency repair: evaluation*

The total time taken for applying the distributed diagnosis and repair of the full corpus using eight slave machines was 2.82hrs; the bulk of the time was taken for (i) extracting the extended conflict sets from the closed corpus on the slave machines which took 24.5mins; (ii) deriving the alternate derivations Δ^- over the input corpus which took 18.8mins; (iii) repairing the corpus which took 1.94hrs. [43]

In Figure 9, we present the high-level results of applying the diagnosis and repair for varying sizes of input corpora and number of slave machines (as per Table 1). For reference, in Table 7 we present the factors by which the runtimes changed when doubling the size of the corpus and when doubling the number of slave machines. We note that:

– doubling the amount of input data increases the runtime (on average) by a factor of between 1.96 and 2.16 respectively;
– when doubling the number of slave machines from 1sm to 2sm, runtimes approximately halve each time.

doubling data				**doubling slaves**			
sm	c/$\frac{c}{2}$	$\frac{c}{2}$/$\frac{c}{4}$	$\frac{c}{4}$/$\frac{c}{8}$	c	2sm/1sm	4sm/2sm	8sm/4sm
1sm	2.26	2.06	2.07	1sm	0.50	0.52	0.51
2sm	2.16	2.14	1.98	2sm	0.52	0.51	0.54
4sm	2.18	2.02	1.96	4sm	0.50	0.55	0.55
8sm	2.06	1.98	1.83	8sm	0.53	0.55	0.59
mean	2.16	2.05	1.96	**mean**	0.51	0.53	0.54

Table 10
On the left, we present the factors by which the inconsistency detection total runtimes changed when doubling the data. On the right, we present the factors by which the runtimes changed when doubling the number of slave machines.

With respect to the first observation—and as per Section 8.3—this is again to be expected given that when the raw corpus increases by 2×, the union of the inferred and raw input data increases by between 2.14× and 2.22×. Thus, the tasks which extend the conflict sets and repair the final corpus must deal with more than double the input data.

With respect to the second observation, as per Section 8.3 we again note that in particular, for the smallest corpus and doubling the number of machines from four to eight, the runtime does not quite halve: this time, the master machine takes up 25.2% of the total runtime with local processing. However, (and again, as per Section 8.3) this becomes increasingly less observable for

[43] Note that for the first two steps, we use an optimisation technique to skip reasoning over triples whose Herbrand universe (set of RDF constants) does not intersect with that of Δ/Δ^+ resp.

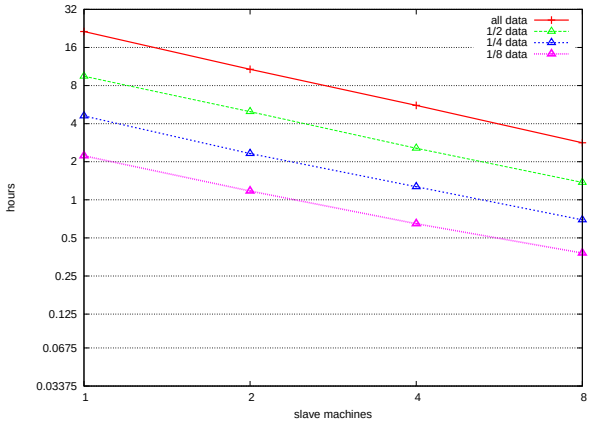

Fig. 9. Total time taken for diagnosis/repair on 1/2/4/8 slave machines for varying corpus sizes

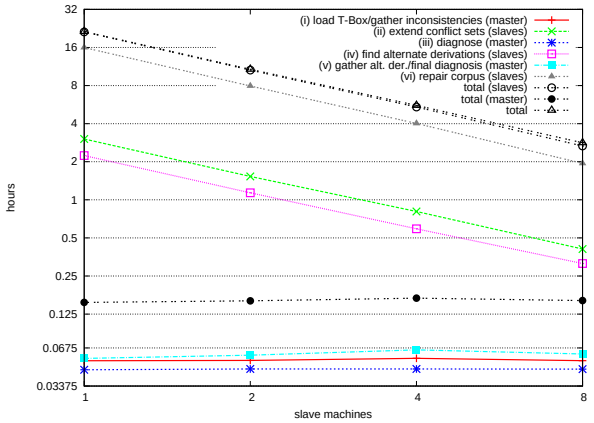

Fig. 10. Breakdown of distributed diagnosis/repair task times for 1/2/4/8 slave machines and full corpus

larger corpora where the percentage of terminological data decreases: when processing the full corpus, note that the amount of time spent on the master machine was between 0.7% for 1sm and 4.2% for 8sm (∼9.6mins). Further, in Figure 8 we present the breakdown of the individual repair tasks for the full corpus using one, two, four and eight slave machines; we observe that for our full corpus, the number of machines can be doubled approximately a further four times (128 machines) before the processing on the master machine takes >50% of the runtime.

With respect to the results of repairing the full corpus, the initial diagnosis over the extended conflict set contained 316,884 entries, and included 16,733 triples added in the extension of the conflict set (triples which inferred a member of the original conflict set). 413,744 facts were inferable for this initial diagnosis, but alternate derivations were found for all but 101,018 (24.4%) of these; additionally, 123 weaker derivations were found for triples in the initial diagnosis. Thus, the entire repair involved removing 417,912 facts and weakening 123 annotations, touching upon 0.02% of the closed corpus.

9. Discussion

In this section, we wrap up the contributions of this paper by (i) giving a summary of the performance of all distributed tasks (Section 9.1); (ii) detail potential problems with respect to scaling further (Section 9.2); (iii) discuss extending our approach to support a larger subset of OWL 2 RL/RDF rules (Section 9.3).

9.1. *Summary of performance/scalability*

Herein, we briefly give an overall summary of all distributed tasks presented in Sections 6–8.

Along these lines, in Figure 9, we summarise the total runtimes for all high-level tasks for varying corpus size and number of machines; as before, we give the breakdown of factors by which the runtimes changed in Table 11. Again, we note that for doubling the size of the corpus, the runtimes more than double; for doubling the number of slave machines, the runtimes don't quite halve. In Figure 12, we present the relative runtimes for each of the high-level tasks; we note that the most expensive tasks are the ranking and annotated reasoning, followed by the inconsistency detection and repair processes. Quite tellingly, the total execution time spent on the master machine is dominated by the time taken for the PageRank computation (also shown in Figure 12 for reference). Similarly, for eight slave machines, the total time spent computing the PageRank of the source-level graph locally on the master machine roughly equates to the total time spent in distributed execution for all other tasks: as the number of machines increases, this would become a more severe bottleneck relative to the runtimes of the distributed tasks—again, distributing PageRank is a mature topic in itself, and subject to investigation in another scope. Besides the ranking, we see that the other high-level tasks behave relatively well when doubling the number of slave machines.

9.2. *Scaling Further*

Aside from the aforementioned ranking bottleneck, other local aggregation tasks—e.g., processing of termi-

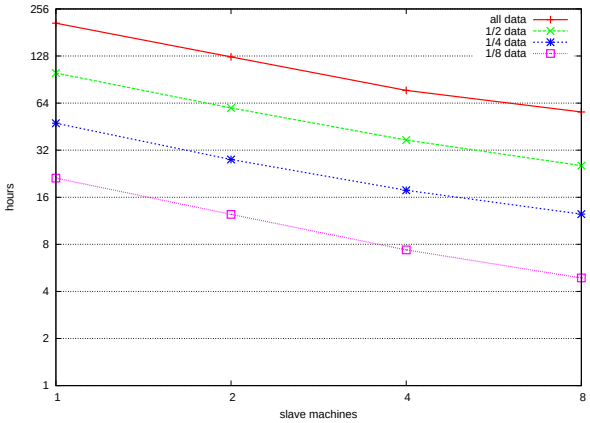

Fig. 11. Total time taken for all tasks on 1/2/4/8 slave machines for varying corpus sizes

doubling data				doubling slaves			
sm	c/$\frac{c}{2}$	$\frac{c}{2}$/$\frac{c}{4}$	$\frac{c}{4}$/$\frac{c}{8}$	c	2sm/1sm	4sm/2sm	8sm/4sm
1sm	2.08	2.10	2.24	1sm	0.61	0.61	0.73
2sm	2.12	2.14	2.24	2sm	0.60	0.62	0.69
4sm	2.08	2.10	2.40	4sm	0.59	0.64	0.70
8sm	2.20	2.04	2.56	8sm	0.59	0.59	0.66
mean	2.12	2.09	2.36	**mean**	0.59	0.62	0.69

Table 11
On the left, we present the factors by which the overall runtimes changed when doubling the data. On the right, we present the factors by which the runtimes changed when doubling the number of slave machines.

nological knowledge—may become a more significant factor for a greatly increased number of machines, especially when the size of the corpus remains low. Similarly, the amount of duplicates and/or dominated inferences produced during reasoning increase along with the number of machines: however, as discussed in Section 7.4, the total amount of duplicates should converge and become less of an issue for higher numbers of machines.

With respect to reasoning in general, our scalability is predicated on the segment of terminological data being relatively small and efficient to process and access; note that for our corpus, we found that ∼0.1% of our corpus was what we considered to be terminological. Since all machines currently must have access to all of the terminology—in one form or another, be it the raw triples or partially evaluated rules—increasing the number of machines in our setup does not increase the amount of terminology the system can handle efficiently. Similarly, the terminology is very frequently accessed, and thus the system must be able to service lookups against it in a very efficient manner; currently, we store the terminology/partially-evaluated rules in

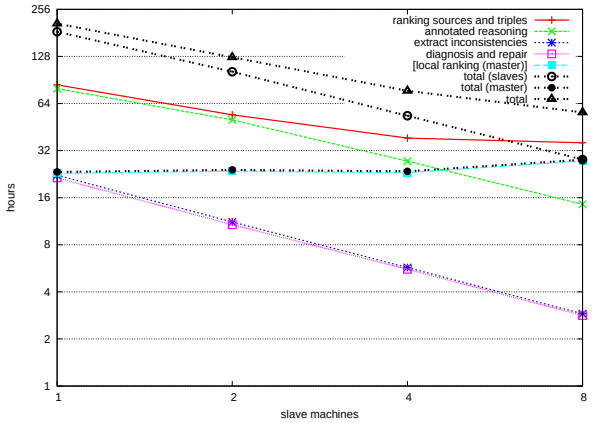

Fig. 12. Breakdown of distributed ranking, reasoning, inconsistency detection, and repair task-times for 1/2/4/8 slave machines and full corpus

memory, and with this approach, the scalability of our system is a function of how much terminology can be fit on the machine with the smallest main-memory in the cluster. However, in situations where there is insufficient main memory to compute the task in this manner, we believe that given the apparent power-law distribution for class and property memberships (see [37, Figs. A.2 & A.3]), a cached on-disk index would work sufficiently well, enjoying a high-cache hit rate and thus a low average lookup time.

Also, although we know that the size of the materialised data is linear with respect to the assertional data (cf. [32]), another limiting factor for scalability is how much materialisation the terminology mandates—or, put another way, how deep the taxonomic hierarchies are under popularly instantiated classes and properties. For the moment, with some careful pruning, the volume of materialised data roughly mirrors the volume of input data; however, if, for example, the FOAF vocabulary today added ten subclasses of `foaf:Person`, the volume of authoritatively materialised data would dramatically increase.

Some of our algorithms require hashing on specific triple elements to align the data required for joins on specific machines; depending on the distribution of the input identifiers, hash-based partitioning of data across machines may lead to load balancing issues. In order to avoid such issues, we do not hash on the predicate position or object position of triples: otherwise, for example, the slave machine that receives triples with the predicate `rdf:type` or object `foaf:Person` would likely have significantly more data to process than its peers (cf. [37, Figs A.1 & A.2]). Instead, we only hash on subject or context, arguing throughout the paper that hashing on

these elements does not cause notable data skew for out current corpus—even still, this may become an issue if, for example, the number of machines is significantly increased, in which case, one may consider alternate data distribution strategies, such as hashing over the entire triple, etc.

Finally, many of our methods also rely on external merge-sorts, which have the linearithmic complexity $O(n\log(n))$; moving towards true Web(-of-Data)-scale, the $\log(n)$ factor can become conspicuous with respect to performance. Although $\log(n)$ grows rather slowly, from a more immediate perspective, performance can also depreciate as the number of on-disk sorted batches required for external merge-sorts increases, which in turn increases the movement of the mechanical disk arm from batch to batch—at some point, a multi-pass merge-sort may become more effective, although we have yet to investigate low-level optimisations of this type. Similarly, many operations on a micro-level—for example, operations on individual resources (subject groups) or batches of triples satisfying a join—are of higher complexity; typically, these batches are processed in memory, which may not be possible given a different morphology of data to that of our current corpus.

In any case, we believe that we have provided a sound basis for scalable, distributed implementation of our methods, given a non-trivial demonstration of scale and feasibility for an input corpus in the order of a billion triples of arbitrary Linked Data, and provided extensive discussion on possible obstacles that may be encountered when considering further levels of scale.

9.3. *Extending OWL 2 RL/RDF support*

Our current subset of OWL 2 RL/RDF rules is chosen since it (i) allows for reasoning via a triple-by-triple scan, (ii) guarantees linear materialisation with respect to the bulk of purely assertional data, and (iii) allows for distributed assertional reasoning without any need for coordination between machines (until aggregation of the output).

First, we note that the logical framework presented in Section 4 can also be used—without modification—for rulesets containing multiple A-atoms in the body (A-Box join rules). However, the distributed implementation for reasoning presented in Section 7, and the methods and implementation for repairing the corpus presented in Section 8, are based on the premise that the original program only contains rules with zero or one A-atoms in the body.

With respect to annotated reasoning, we note that our local assertional reasoning algorithm (as per Algorithm 1) can support A-Box join rules; this is discussed in Section 7.1. However, in the worst case when supporting such rules (and as given by the eq-rep-* OWL 2 RL/RDF rules), we may have to index the entire corpus of assertional and inferred data, which would greatly increase the expense of the current reasoning process.

Further, the growth in materialised data may become quadratic when considering OWL 2 RL/RDF A-Box join rules—such as prp-trp which supports transitivity. In fact, the worst case for materialisation with respect to OWL 2 RL/RDF rules is cubic, which can be demonstrated with a simple example involving two triples:

(`owl:sameAs`, `owl:sameAs`, `rdf:type`)
(`owl:sameAs`, `rdfs:domain`, `ex:BadHub`)

Adding these two triples to any arbitrary RDF graph will lead to the inference of all possible (generalised) triples by the OWL 2 RL/RDF rules: i.e., the inference of $C \times C \times C$ (a.k.a. the Herbrand base), where $C \subset \mathbf{C}$ is the set of RDF constants mentioned in the OWL 2 RL/RDF ruleset and the graph (a.k.a. the Herbrand universe). [44] Note however that not all multiple A-atom rules can produce quadratic (or cubic) inferencing with respect to assertional data: some rules (such as cls-int1, cls-svf1) are what we call *A-guarded*, whereby (loosely) the head of the rule contains only one variable not ground by partial evaluation with respect to the terminology, and thus the number of materialised triples that can be generated for such a rule is linear with respect to the assertional data.

Despite this, such A-guarded rules would not fit neatly into our partial-indexing approach, and may require indexing of a significant portion of the assertional data (and possibly all such data). Also, A-Box join rules would not fit into our current distributed implementation, where assertional data must then be coordinated between machines to ensure correct computation of joins (e.g., as presented in [62]).

Assuming a setting whereby A-Box join rules could be supported for classical (non-annotated) materialisation, annotated reasoning would additionally require each rule and fact to be associated with a set of annotation tuples. As per the formal results of Section 4.3, for deriving only optimal annotations with respect to the presented annotation domain, each rule or fact need only be associated with a maximum of four annotation tuples; assuming our threshold, each rule or fact

[44] The rules required are the prp-dom rule for supporting `rdfs:domain` and the eq-* rules for supporting the semantics of `owl:sameAs`.

need only be associated with a single (optimal) rank annotation—both of these annotation settings should have a relatively small performance/scalability impact for a reasoner supporting the classical inferences of a larger subset of OWL 2 RL/RDF.

Finally, we note that our repair process currently does not support A-Box join rules. Assuming that the original program does not contain such rules allows for the simplified scan-based procedure sketched in Section 8.1; although we believe that certain aspects of our approach may be re-used for supporting a fuller profile of OWL 2 RL/RDF, we leave the nature of this extension as an open question.

10. Related work

In this section, we introduce related works in the various fields touched upon in this paper. We begin in Section 10.1 by discussing related works in the field of annotated programs and annotated reasoning; we discuss scalable and distributed RDF(S) and OWL reasoning in Section 10.2, followed by Web-reasoning literature in Section 10.3, and finally inconsistency repair in 10.4.

10.1. *Annotated programs*

With respect to databases, [26] proposed *semiring structures* for propagating annotations through queries and views in a generic way, particularly for provenance annotations, but also applying this approach for instance to probabilistic annotations, or annotations modeling tuple cardinality (i.e., bag semantics). For such forms of annotations, [26] gives an algorithm that can decide query answers for annotated Datalog [45] — materialisation in their general case, though, can be infinite. In contrast, our work focuses exactly on efficient and scalable materialisation, however (i) in a very specialised setting of Datalog rules, (ii) for a less flexible annotation structure, such that we can guarantee finiteness and scalability.

Cheney et al. [10] present a broad survey of provenance techniques deployed in database approaches, where they categorise provenance into (i) *why*-provenance—annotations that model the raw tuples from which answers/inferences were obtained, (ii) *how* provenance—annotations that model the *combinations* of raw tuples by which answers/inferences were obtained, and (iii) *where* provenance—annotations that model the combinations of source tuples from which answers/inferences were obtained. In this categorisation, the above mentioned provenance semirings of Green et al. [26] fall into the so-called *how*-provenance approaches. We note, however, that our annotations are orthogonal to all of these provenance notations of [10]: although our annotations for authoritativeness and ranking are informed by data provenance, our annotations do not directly model or track "raw provenance", although we consider a scalable extension of our approach in this manner an interesting direction for future research. [46]

Perhaps of more immediate interest to us, in [41] the authors of [26] extend their work with various annotations *related* to provenance, such as confidence, rank, relationships between triples (where, remarkably, they also discuss—somewhat orthogonal to our respective notion—authoritativeness). All of these are again modelled using semirings, and an implementation is provided by means of translation to SQL.

In [7] the Datalog language has been extended with weights that can represent metainformation relevant to Trust Management systems, such as costs, preferences and trust levels associated to policies or credentials. Weights are taken from a c-semiring where the two operations $\times$ and $+$ are used, respectively, to compose the weights associated to statements, and select the best derivation chain. Some example of c-semirings are provided where weights assume the meaning of costs, probabilities and fuzzy values. In all of these examples the operator $\times$ is idempotent, so that the c-semiring induces a complete lattice where $+$ is the lub and $\times$ is the glb. In these cases, by using our reasoning task $\mathsf{opt}(P)$, the proposed framework can be naturally translated into our annotated programs. The complexity of Weighted Datalog is just sketched—only decidability is proven. Scalability issues are not tackled and no experimental results are provided.

More specifically with respect to RDF, in [22] the provenance of inferred RDF data is tackled by augmenting triples with a fourth component named *colour* representing the collection of the different data sources used to derive a triple. A binary operation $+$ over colours forms a semigroup; the provenance of derived triples is the sum of the provenance of the supporting triples. Clearly, also this framework can be simulated by our annotated programs by adopting a flat lattice where all of the elements (representing different provenances) are mutually incomparable. Then, for each derived atom,

[45] Here, only the ground data rather than the proper rules are annotated.

[46] Further, in our case we consider provenance as representing a container of information—a Web document—rather than a "method of derivation".

$\mathsf{plain}(P)$ collects all of the provenances employed according to [22]. Although the complexity analysis yields an almost linear upper bound, no experimental results are provided.

Dividino et al. [17] introduce RDF$^+$: a language which allows for expressing "meta-knowledge" about the core set of facts expressed as RDF. Meta knowledge can, for example, encode provenance information such as the URL of an originating document, a timestamp, a certainty score, etc.; different properties of meta knowledge are expressible in the framework as appropriate for different applications. The approach is based on named graphs and internal statement identifiers; meta statements are given as RDF triples where a statement identifier appears as the subject. Semantics are given for RDF$^+$ through standard interpretations and interpretations for each individual property of meta knowledge; mappings between RDF and RDF$^+$ are defined. Unlike us, Dividino et al. focus on extending SPARQL query processing to incorporate such meta knowledge: they do not consider, e.g., OWL semantics for reasoning over such meta knowledge. Later results by Schenk et al. [56] look at reasoning and debugging meta-knowledge using the Pellet (OWL DL) reasoner, but only demonstrate evaluation over ontologies in the order of thousands of axioms.

Lopes et al. [49] define a general annotation framework for RDFS, together with AnQL: a query language inspired by SPARQL which includes querying over annotations. Annotations are formalised in terms of residuated bounded lattices, which can be specialised to represent different types of meta-information (temporal constraints, fuzzy values, provenance etc.). A general deductive system—based on abstract algebraic structures—has been provided and proven to be in PTIME as far as the operations defined in the structure can be computed in polynomial time. The Annotated RDFS framework enables the representation of a large spectrum of different meta-information. Some of them—in particular those where an unbounded number of different values can be assigned to an inferred triple—do not fit our framework. On the other hand, the framework of [49] is strictly anchored to RDFS, while our annotated programs are founded on OWL 2 RL/RDF and hence are transversable with respect to the underlying ontology language. Moreover, our results place more emphasis on scalability. Along similar lines, work presented in [9] also focuses on the definition of a general algebra for annotated RDFS, rather than on computational aspects which we are concerned about here.

10.2. *Scalable/distributed reasoning*

From the perspective of scalable RDF(S)/OWL reasoning, one of the earliest engines to demonstrate reasoning over datasets in the order of a billion triples was the commercial system BigOWLIM [6], which is based on a scalable and custom-built database management system over which a rule-based materialisation layer is implemented, supporting fragments such as RDFS and pD*, and more recently OWL 2 RL/RDF. Most recent results claim to be able to load 12b statements of the LUBM synthetic benchmark, and 20.5b statements statements inferrable by pD* rules on a machine with 2x Xeon 5430 (2.5GHz, quad-core), and 64GB (FB-DDR2) of RAM. [47] We note that this system has been employed for relatively high-profile applications, including use as the content management system for a live BBC World Cup site. [48] BigOWLIM features distribution, but only as a replication strategy for fault-tolerance and supporting higher query load.

The Virtuoso SPARQL engine [19] features support for inference rules; details of implementation are sketchy, but the `rdfs:subClassOf`, `rdfs:subPropertyOf`, `owl:equivalentClass`, `owl:equivalentProperty` and `owl:sameAs` primitives are partially supported by a form of internal backward-chaining, and other user-defined rules—encoded in SPARQL syntax—can also be passed to the engine. [49]

A number of other systems employ a similar strategy to ours for distributed RDFS/OWL reasoning: i.e., separating out terminological data, flooding these data to all machines, and applying parallel reasoning over remote segments of the assertional data.

One of the earliest such distributed reasoning approaches is DORS [20], which uses DHT-based partitioning to split the input corpus over a number of nodes. Data segments are stored in remote databases, where a tableaux-based reasoner is used to perform reasoning over the T-Box, and where a rule-based reasoner is used to perform materialisation. Since they apply the DLP ruleset [27]—which contains rules with multiple A-atoms—the nodes in their distributed system must coordinate to perform the required assertional joins, which again is performed using DHT-based partitioning. However, they hash assertional data based on the

[47] `http://www.ontotext.com/owlim/benchmarking/lubm.html`
[48] `http://www.readwriteweb.com/archives/bbc_world_cup_website_semantic_technology.php`
[49] `http://docs.openlinksw.com/virtuoso/rdfsparqlrule.html`

predicate and the value of `rdf:type` which would cause significant data-skew for Linked Data corpora (cf. [37, Figs. A.2 & A.3]). Their evaluation is with respect to ∼2m triples of synthetically generated data on up to 32 nodes.

Weaver & Hendler [65] discuss a similar approach for distributed materialisation with respect to RDFS—they also describe a separation of terminological (what they call ontological) data from assertional data. Thereafter, they identify that all RDFS rules have only one assertional atom and, like us, use this as the basis for a scalable distribution strategy: they flood the terminological data and split the assertional data over their machines. Inferencing is done over an in-memory RDF store. They evaluate their approach over a LUBM-generated synthetic corpus of 345.5m triples using a maximum of 128 machines (each with two dual-core 2.6 GHz AMD Opteron processors and 16 GB memory); with this setup, reasoning in memory takes just under 5 minutes, producing 650m triples.

Urbani et al. [63] use MapReduce [13] for distributed RDFS materialisation over 850m Linked Data triples. They also consider a separation of terminological (what they call schema) data from assertional data as a core optimisation of their approach, and—likewise with [65]—identify that RDFS rules only contain one assertional atom. As a pre-processing step, they sort their data by subject to reduce duplication of inferences. Based on inspection of the rules, they also identify an ordering (stratification) of RDFS rules which (again assuming standard usage of the RDFS meta-vocabulary) allows for completeness of results without full recursion—unlike us, they do reasoning on a per-rule basis as opposed to our per-triple basis. Unlike us, they also use a 8-byte dictionary encoding of terms. Using 32 machines (each with 4 cores and 4 GB of memory) they infer 30b triples from 865m triples in less than one hour; however, they do not materialise or *decode* the output—a potentially expensive process. Note that they do not include any notion of authority (although they mention that in future, they may include such analysis): they attempted to apply pD* on 35m Web triples and stopped after creating 3.8b inferences in 12hrs, lending strength to our arguments for authoritative reasoning.

In more recent work [62], (approximately) the same authors revisit the topic of materialisation with respect to pD*. They again use a separation of terminological data from assertional data as the basis for scalable distributed reasoning, but since pD* contains rules with multiple A-atoms, they define bespoke MapReduce procedures to handle each such rule, some of which are similar in principle to those presented in [36] (and later on) such as canonicalisation of terms related by `owl:sameAs`. They demonstrate their methods over three datasets; (i) 1.51b triples of UniProt data, generating 2.03b inferences in 6.1hrs using 32 machines; (ii) 0.9b triples of LDSR data (discussed later), generating 0.94b inferences in 3.52hrs using 32 machines; (iii) 102.5b triples of synthetic LUBM data, generating 47.6b inferences in 45.7hrs using 64 machines. The latter experiment is two orders of magnitude above our current experiments, and features rules which require A-Box joins; however, the authors do not look at open Web data, stating that "reasoning over arbitrary triples retrieved from the Web would result in useless and unrealistic derivations" [62]. They do, however, mention the possibility of including our authoritative reasoning algorithm in their approach, in order to prevent such adverse affects.

In very recent work [46], Kolovski et al. have presented an (Oracle) RDBMS-based OWL 2 RL/RDF materialisation approach. They again use some similar optimisations to the scalable reasoning literature, including parallelisation, canonicalisation of `owl:sameAs` inferences, and also partial evaluation of rules based on highly selective patterns—from discussion in the paper, these selective patterns seem to correlate with the terminological patterns of the rule. They also discuss many low-level engineering optimisations and Oracle tweaks to boost performance. Unlike the approaches mentioned thus far, Kolovski et al. tackle the issue of updates, proposing variants of semi-naïve evaluation to avoid rederivations. The authors evaluate their work for a number of different datasets and hardware configurations; the largest scale experiment they present consists of applying OWL 2 RL/RDF materialisation over 13 billion triples of LUBM using 8 nodes (Intel Xeon 2.53 GHz CPU, 72GB memory each) in just under 2 hours.

10.3. *Web reasoning*

As previously mentioned, in [63], the authors discuss reasoning over 850m Linked Data triples—however, they only do so over RDFS and do not consider any issues relating to provenance.

In [43], the authors apply reasoning over 0.9b Linked Data triples using the aforementioned BigOWLIM reasoner; however, this dataset is manually selected as a merge of a number of smaller, known datasets as opposed to an arbitrary corpus—they do not consider any general notions of provenance or Web tolerance. (As aforementioned, the WebPie system [62] has also been applied over the LDSR dataset.)

In a similar approach to our authoritative analysis, Cheng et al. [12] introduced restrictions for accepting sub-class and equivalent-class axioms 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 both simplifications are not compatible with the common use of PURL services [50]: (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 [11] better refined the notion of an *authoritative description* to one based on redirects—and one which aligns very much with our notion of authority. They use their notion of authority to do reasoning over class hierarchies, but only include custom support of `rdfs:subClassOf` and `owl:equivalentClass`, as opposed to our general framework for authoritative reasoning over arbitrary T-split rules.

A viable alternative approach to authority—and which also looks more generally at provenance for Web reasoning—is that of "quarantined reasoning", described in [14]. The core intuition is to consider applying reasoning on a per-document basis, taking each Web document and its recursive (implicit and explicit) imports and applying reasoning over the union of these documents. The reasoned corpus is then generated as the merge of these per-document closures. In contrast to our approach where we construct one authoritative terminological model for all Web data, their approach uses a bespoke trusted model for each document; thus, they would infer statements within the local context which we would consider to be non-authoritative, but our model is more flexible for performing inference over the merge of documents. [51] As such, they also consider a separation of terminological and assertional data; in this case ontology documents and data documents. Their evaluation was performed in parallel using three machines (quad-core 2.33GHz CPU with 8GB memory each); they reported loading, on average, 40 documents per second.

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

[51] Although it should be noted that without considering rules with assertional joins, our ability to make inferences across documents is somewhat restricted.

10.4. *Inconsistency repair*

Most legacy works in this area (e.g., see DION and MUPSTER [57] and a repair tool for unsatisfiable concepts in Swoop [40]) focus on debugging singular OWL ontologies within a Description Logics formalism, in particular focussing on fixing terminologies (T-Boxes) which include unsatisfiable concepts—not of themselves an inconsistency, but usually indicative of a modelling error (termed incoherence) in the ontology. Such approaches usually rely on the extraction and analysis of MUPs (Minimal Unsatisfiability Preserving Sub-terminologies) and MIPs (Minimal Incoherence Preserving Sub-terminologies), usually to give feedback to the ontology editor during the modelling process. However, these approaches again focus on debugging terminologies, and have been shown in theory and in practice to be expensive to compute—please see [59] for a survey (and indeed critique) of such approaches.

There are a variety of other approaches for handling inconsistencies in OWL ontologies, including works on paraconsistent reasoning using multi-valued, probabalistic, or possibilistic appraoches, by, e.g., Ma et al. [52], Zhang et al. [67], Huang et al. [39], Qi et al. [54], etc. [52] However, all such approaches are again based on Description Logics formalisms, and only demonstrate evaluation over one (or few) ontologies containing in the order of thousands, tens of thousands, up to hundreds of thousands of axioms.

11. Conclusion

In this paper, we have given a comprehensive discussion on methods for incorporating notions of provenance during Linked Data reasoning. In particular, we identified three dimensions of trust-/provenance-related annotations for data: viz. (i) *blacklisting*, where a particular source or type of information is distrusted and should not be considered during reasoning or be included in the final corpus; (ii) *authoritativeness* which analyses the source of terminological facts to determine whether assertional inferences they mandate can be trusted; (iii) *ranking* which leverages the well-known PageRank algorithm for links-based analysis of the source-level graph, where ranking values are subsequently propagated to facts in the corpus.

We continued by discussing a formal framework based on annotated logic programs for tracking these

[52] For a survey of the use of multi-valued, probabilistic and possibilistic approaches for uncertainty in Descrption Logics, see [50].

dimensions of trust and provenance during the reasoning procedure. We gave various formal properties of the program—some specific to our domain of annotation, some not—which demonstrated desirable properties relating to termination, growth of the program, and efficient implementation. Later, we provided a use-case for our annotations involving detection and repair of inconsistencies.

We introduced our distribution framework for implementing and running our algorithms over a cluster of commodity hardware, subsequently detailing non-trivial implementations for deriving rank annotations, for applying reasoning wrt. the defined logic program, for detecting inconsistencies in the corpus, and for leveraging the annotations in repairing inconsistencies. All of our methods were individually justified by evaluation over a 1.118b quadruple Linked Data corpus, with a consistent unbroken thread of evaluation throughout. In so doing, we have looked at non-trivial application and analysis of Linked Data principles, links-based analysis, annotated logic programs, OWL 2 RL/RDF rules (including the oft overlooked constraint rules), and inconsistency repair techniques, incorporating them into a coherent system for scalable and tolerant Web reasoning.

As the Web of Data expands and diversifies, the need for reasoning will grow more and more apparent, as will the implied need for methods of handling and incorporating notions of trust and provenance which scale to large corpora, and which are tolerant to spamming and other malicious activity. We hope that this paper represents a significant step forward with respect to research into scalable Web-tolerant reasoning techniques which incorporate provenance and trust.

Acknowledgements We would like to thank Antoine Zimmermann for providing feedback on earlier drafts of this paper. We would also like to thank the anonymous reviewers and the editors for their time and comments. The work presented in this paper has been funded in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2), and by an IRCSET postgraduate scholarship.

References

[1] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. Objectrank: authority-based keyword search in databases. In *Proceedings of the 13th International Conference on Very Large Data Bases*, pages 564–575, 2004.

[2] David Beckett and Tim Berners-Lee. Turtle – Terse RDF Triple Language. W3C Team Submission, January 2008. `http://www.w3.org/TeamSubmission/turtle/`.

[3] Tim Berners-Lee. Linked Data. Design issues for the World Wide Web, World Wide Web Consortium, 2006. `http://www.w3.org/DesignIssues/LinkedData.html`.

[4] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, January 2005. `http://tools.ietf.org/html/rfc3986`.

[5] Mark Birbeck and Shane McCarron. CURIE Syntax 1.0 – A syntax for expressing Compact URIs. W3C Recommendation, January 2009. `http://www.w3.org/TR/curie/`.

[6] Barry Bishop, Atanas Kiryakov, Damyan Ognyanoff, Ivan Peikov, Zdravko Tashev, and Ruslan Velkov. OWLIM: A family of scalable semantic repositories. *Semantic Web Journal*, 2011. In press; available at `http://www.semantic-web-journal.net/sites/default/files/swj97_0.pdf`.

[7] Stefano Bistarelli, Fabio Martinelli, and Francesco Santini. Weighted Datalog and Levels of Trust. In *Proceedings of the The Third International Conference on Availability, Reliability and Security, ARES 2008, March 4-7, 2008, Technical University of Catalonia, Barcelona , Spain*, pages 1128–1134, 2008.

[8] Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks*, 30(1-7):107–117, 1998.

[9] Peter Buneman and Egor Kostylev. Annotation algebras for rdfs. In *The Second International Workshop on the role of Semantic Web in Provenance Management (SWPM-10)*. CEUR Workshop Proceedings, 2010.

[10] James Cheney, Laura Chiticariu, and Wang-Chiew Tan. Provenance in Databases: Why, How, and Where. *Foundations and Trends in Databases*, 1:379–474, April 2009.

[11] Gong Cheng, Weiyi Ge, Honghan Wu, and Yuzhong Qu. Searching Semantic Web Objects Based on Class Hierarchies. In *Proceedings of Linked Data on the Web Workshop*, 2008.

[12] Gong Cheng and Yuzhong Qu. Term Dependence on the Semantic Web. In *International Semantic Web Conference*, pages 665–680, oct 2008.

[13] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, pages 137–150, 2004.

[14] Renaud Delbru, Axel Polleres, Giovanni Tummarello, and Stefan Decker. Context Dependent Reasoning for Semantic Documents in Sindice. In *Proc. of 4th SSWS Workshop*, October 2008.

[15] Renaud Delbru, Nickolai Toupikov, Michele Catasta, Giovanni Tummarello, and Stefan Decker. Hierarchical Link Analysis for Ranking Web Data. In *ESWC (2)*, pages 225–239, 2010.

[16] Li Ding, Rong Pan, Timothy W. Finin, Anupam Joshi, Yun Peng, and Pranam Kolari. Finding and Ranking Knowledge on the Semantic Web. In *International Semantic Web Conference*, pages 156–170, 2005.

[17] Renata Queiroz Dividino, Sergej Sizov, Steffen Staab, and Bernhard Schueler. Querying for provenance, trust, uncertainty and other meta knowledge in RDF. *J. Web Sem.*, 7(3):204–219, 2009.

[18] Michael Schneider (ed.). OWL 2 Web Ontology Language: RDF-Based Semantics. W3C Working Draft, October 2009. `http://www.w3.org/TR/2009/REC-owl2-rdf-based-semantics-20091027/`.

[19] Orri Erling and Ivan Mikhailov. RDF Support in the Virtuoso DBMS. In *Networked Knowledge – Networked Media*, volume 221 of *Studies in Computational Intelligence*, pages 7–24. Springer, 2009.

[20] Qiming Fang, Ying Zhao, Guangwen Yang, and Weimin Zheng. Scalable Distributed Ontology Reasoning Using DHT-Based Partitioning. In *ASWC*, pages 91–105, 2008.

[21] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999. http://www.ietf.org/rfc/rfc2616.txt.

[22] Giorgos Flouris, Irini Fundulaki, Panagiotis Pediaditis, Yannis Theoharis, and Vassilis Christophides. Coloring RDF Triples to Capture Provenance. In *The Semantic Web - ISWC 2009, 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009. Proceedings*, pages 196–212, 2009.

[23] Thomas Franz, Antje Schultz, Sergej Sizov, and Steffen Staab. Triplerank: Ranking semantic web data by tensor decomposition. In *International Semantic Web Conference*, pages 213–228, 2009.

[24] David Gleich, Leonid Zhukov, and Pavel Berkhin. Fast Parallel PageRank: A Linear System Approach. Technical Report YRL-2004-038, Yahoo! Research Labs, 2004. http://www.stanford.edu/~dgleich/publications/prlinear-dgleich.pdf.

[25] Bernardo Cuenca Grau, Boris Motik, Zhe Wu, Achille Fokoue, and Carsten Lutz (eds.). OWL 2 Web Ontology Language: Profiles. W3C Working Draft, April 2008. http://www.w3.org/TR/owl2-profiles/.

[26] Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'07)*, pages 31–40, Beijing, China, June 2007. ACM Press.

[27] 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.

[28] Christophe Guéret, Paul T. Groth, Frank van Harmelen, and Stefan Schlobach. Finding the Achilles Heel of the Web of Data: Using Network Analysis for Link-Recommendation. In *International Semantic Web Conference (1)*, pages 289–304, 2010.

[29] Andreas Harth, Sheila Kinsella, and Stefan Decker. Using Naming Authority to Rank Data and Ontologies for Web Search. In *International Semantic Web Conference*, pages 277–292, 2009.

[30] Patrick Hayes. RDF semantics. W3C Recommendation, February 2004. http://www.w3.org/TR/rdf-mt/.

[31] Pascal Hitzler and Frank van Harmelen. A Reasonable Semantic Web. *Semantic Web Journal*, 1(1), 2010. (to appear – available from http://www.semantic-web-journal.net/).

[32] Aidan Hogan. *Exploiting RDFS and OWL for Integrating Heterogeneous, Large-Scale, Linked Data Corpora*. PhD thesis, Digital Enterprise Research Institute, National University of Ireland, Galway, 2011. Available from http://aidanhogan.com/docs/thesis/.

[33] Aidan Hogan, Andreas Harth, and Stefan Decker. ReConRank: A Scalable Ranking Method for Semantic Web Data with Context. In *2nd Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2006)*, 2006.

[34] Aidan Hogan, Andreas Harth, and Stefan Decker. Performing Object Consolidation on the Semantic Web Data Graph. In *1st I3 Workshop: Identity, Identifiers, Identification Workshop*, 2007.

[35] Aidan Hogan, Andreas Harth, Alexandre Passant, Stefan Decker, and Axel Polleres. Weaving the Pedantic Web. In *3rd International Workshop on Linked Data on the Web (LDOW2010)*, Raleigh, USA, April 2010.

[36] Aidan Hogan, Andreas Harth, and Axel Polleres. Scalable Authoritative OWL Reasoning for the Web. *Int. J. Semantic Web Inf. Syst.*, 5(2):49–90, 2009.

[37] Aidan Hogan, Andreas Harth, Jürgen Umbrich, Sheila Kinsella, Axel Polleres, and Stefan Decker. Searching and Browsing Linked Data with SWSE: the Semantic Web Search Engine. Technical Report DERI-TR-2010-07-23, Digital Enterprise Research Institute (DERI), 2010. http://www.deri.ie/fileadmin/documents/DERI-TR-2010-07-23.pdf.

[38] Aidan Hogan, Jeff Z. Pan, Axel Polleres, and Stefan Decker. SAOR: Template Rule Optimisations for Distributed Reasoning over 1 Billion Linked Data Triples. In *International Semantic Web Conference*, 2010.

[39] Zhisheng Huang, Frank van Harmelen, and Annette ten Teije. Reasoning with Inconsistent Ontologies. In *IJCAI*, pages 454–459, 2005.

[40] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and Bernardo Cuenca Grau. Repairing unsatisfiable concepts in owl ontologies. In *ESWC*, pages 170–184, 2006.

[41] Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. Querying data provenance. In *Proceedings of the 2010 International Conference on Management of Data (SIGMOD'10)*, pages 951–962, New York, NY, USA, 2010. ACM.

[42] Michael Kifer and V. S. Subrahmanian. Theory of Generalized Annotated Logic Programming and its Applications. *J. Log. Program.*, 12(3&4), 1992.

[43] Atanas Kiryakov, Damyan Ognyanoff, Ruslan Velkov, Zdravko Tashev, and Ivan Peikov. LDSR: a Reason-able View to the Web of Linked Data. In *Semantic Web Challenge (ISWC2009)*, 2009.

[44] Jon M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM*, 46(5):604–632, 1999.

[45] Christian Kohlschütter, Paul-Alexandru Chirita, and Wolfgang Nejdl. Efficient Parallel Computation of PageRank. In *ECIR*, pages 241–252, 2006.

[46] Vladimir Kolovski, Zhe Wu, and George Eadon. Optimizing Enterprise-scale OWL 2 RL Reasoning in a Relational Database System. In *International Semantic Web Conference*, 2010.

[47] Spyros Kotoulas, Eyal Oren, and Frank van Harmelen. Mind the Data Skew: Distributed Inferencing by Speeddating in Elastic Regions. In *WWW*, pages 531–540, 2010.

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

[49] Nuno Lopes, Axel Polleres, Umberto Straccia, and Antoine Zimmermann. AnQL: SPARQLing Up Annotated RDFS. In *The Semantic Web - ISWC 20010, 9th International Semantic Web Conference, ISWC 2010, Shangai, Cina, November 7-11, to appear. Proceedings*, 2010.

[50] Thomas Lukasiewicz and Umberto Straccia. Managing uncertainty and vagueness in description logics for the Semantic Web. *J. Web Sem.*, 6(4):291–308, 2008.

[51] Yue Ma and Pascal Hitzler. Paraconsistent reasoning for owl 2. In *Third International Conference on Web Reasoning and Rule Systems (RR2009)*, pages 197–211, 2009.

[52] Yue Ma, Pascal Hitzler, and Zuoquan Lin. Algorithms for Paraconsistent Reasoning with OWL. In *ESWC*, pages 399–413, 2007.

[53] Axel Polleres and David Huynh, editors. *Journal of Web Semantics, Special Issue: The Web of Data*, volume 7(3). Elsevier, 2009.

[54] Guilin Qi, Qiu Ji, Jeff Z. Pan, and Jianfeng Du. Extending description logics with uncertainty reasoning in possibilistic logic. *Int. J. Intell. Syst.*, 26(4):353–381, 2011.

[55] Raymond Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, 1987.

[56] Simon Schenk, Renata Queiroz Dividino, and Steffen Staab. Reasoning With Provenance, Trust and all that other Meta Knowledge in OWL. In *SWPM*, 2009.

[57] Stefan Schlobach, Zhisheng Huang, Ronald Cornet, and Frank van Harmelen. Debugging Incoherent Terminologies. *J. Autom. Reasoning*, 39(3):317–349, 2007.

[58] Michael Stonebraker. The Case for Shared Nothing. *IEEE Database Eng. Bull.*, 9(1):4–9, 1986.

[59] Heiner Stuckenschmidt. Debugging owl ontologies - a reality check. In *EON*, 2008.

[60] J. D. Ullman. *Principles of Database and Knowledge Base Systems*. Computer Science Press, New York, NY, USA, 1989.

[61] Jeffrey D. Ullman. *Principles of Database and Knowledge Base Systems*. Computer Science Press, 1989.

[62] Jacopo Urbani, Spyros Kotoulas, Jason Maassen, Frank van Harmelen, and Henri E. Bal. OWL Reasoning with WebPIE: Calculating the Closure of 100 Billion Triples. In *ESWC (1)*, pages 213–227, 2010.

[63] Jacopo Urbani, Spyros Kotoulas, Eyal Oren, and Frank van Harmelen. Scalable Distributed Reasoning Using MapReduce. In *International Semantic Web Conference*, pages 634–649, 2009.

[64] Denny Vrandečíc, Markus Krötzsch, Sebastian Rudolph, and Uta Lösch. Leveraging non-lexical knowledge for the linked open data web. *Review of April Fool's day Transactions (RAFT)*, 5:18–27, 2010.

[65] Jesse Weaver and James A. Hendler. Parallel Materialization of the Finite RDFS Closure for Hundreds of Millions of Triples. In *International Semantic Web Conference (ISWC2009)*, pages 682–697, 2009.

[66] Eyal Yardeni and Ehud Y. Shapiro. A Type System for Logic Programs. *J. Log. Program.*, 10(1/2/3&4):125–153, 1991.

[67] Xiaowang Zhang, Guohui Xiao, and Zuoquan Lin. A Tableau Algorithm for Handling Inconsistency in OWL. In *ESWC*, pages 399–413, 2009.

Appendix A. Rule Tables

Herein, we list the subset of OWL 2 RL/RDF rules we apply in our scenario categorised by the assertional and terminological arity of the rule bodies, including rules with no antecedent (Table A.1), rules with only T-atoms in the body (Table A.2), rules with only a single A-atom in the body (Table A.3), and rules with some T-atoms and a single A-atom in the body (Table A.4). Also, in Table A.5, we give an indication as to how recursive application of rules in Table A.4 can be complete, even if the inferences from rules in Table A.2 are omitted. Finally, in Table A.6, we give the OWL 2 RL/RDF rules used for consistency checking.

Body$(R) = \emptyset$		
OWL2RL	**Consequent**	**Notes**
prp-ap	*?p* a owl:AnnotationProperty .	For each built-in annotation property
cls-thing	owl:Thing a owl:Class .	-
cls-nothing	owl:Nothing a owl:Class .	-
dt-type1	*?dt* a rdfs:Datatype .	For each built-in datatype
~~dt-type2~~	~~*?l* a *?dt* .~~	~~For all *?l* in the value space of datatype *?dt*~~
~~dt-eq~~	~~$?l_1$ owl:sameAs $?l_2$.~~	~~For all $?l_1$ and $?l_2$ with the same data value~~
~~dt-diff~~	~~$?l_1$ owl:differentFrom $?l_2$.~~	~~For all $?l_1$ and $?l_2$ with different data values~~

Table A.1
Rules with empty body (axiomatic triples); we strike out the datatype rules which we currently do not support

TBody$(R) \neq \emptyset$, ABody$(R) = \emptyset$		
OWL2RL	**Antecedent**	**Consequent**
	terminological	
cls-00	*?c* owl:oneOf ($?x_1$...$?x_n$) .	$?x_1$...$?x_n$ a *?c* .
scm-cls	*?c* a owl:Class .	*?c* rdfs:subClassOf *?c* ; rdfs:subClassOf owl:Thing ; owl:equivalentClass *?c* . owl:Nothing rdfs:subClassOf *?c* .
scm-sco	$?c_1$ rdfs:subClassOf $?c_2$. $?c_2$ rdfs:subClassOf $?c_3$.	$?c_1$ rdfs:subClassOf $?c_3$.
scm-eqc1	$?c_1$ owl:equivalentClass $?c_2$.	$?c_1$ rdfs:subClassOf $?c_2$. $?c_2$ rdfs:subClassOf $?c_1$.
scm-eqc2	$?c_1$ rdfs:subClassOf $?c_2$. $?c_2$ rdfs:subClassOf $?c_1$.	$?c_1$ owl:equivalentClass $?c_2$.
scm-op	*?p* a owl:ObjectProperty .	*?p* rdfs:subPropertyOf *?p* . *?p* owl:equivalentProperty *?p* .
scm-dp	*?p* a owl:DatatypeProperty .	*?p* rdfs:subPropertyOf *?p* . *?p* owl:equivalentProperty *?p* .
scm-spo	$?p_1$ rdfs:subPropertyOf $?p_2$. $?p_2$ rdfs:subPropertyOf $?p_3$.	$?p_1$ rdfs:subPropertyOf $?p_3$.
scm-eqp1	$?p_1$ owl:equivalentProperty $?p_2$.	$?p_1$ rdfs:subPropertyOf $?p_2$. $?p_2$ rdfs:subPropertyOf $?p_1$.
scm-eqp2	$?p_1$ rdfs:subPropertyOf $?p_2$. $?p_2$ rdfs:subPropertyOf $?p_1$.	$?p_1$ owl:equivalentProperty $?p_2$.
scm-dom1	*?p* rdfs:domain $?c_1$. $?c_1$ rdfs:subClassOf $?c_2$.	*?p* rdfs:domain $?c_2$.
scm-dom2	$?p_2$ rdfs:domain *?c* . $?p_1$ rdfs:subPropertyOf $?p_2$.	$?p_1$ rdfs:domain *?c* .
scm-rng1	*?p* rdfs:range $?c_1$. $?c_1$ rdfs:subClassOf $?c_2$.	*?p* rdfs:range $?c_2$.
scm-rng2	$?p_2$ rdfs:range *?c* . $?p_1$ rdfs:subPropertyOf $?p_2$.	$?p_1$ rdfs:range *?c* .
scm-hv	$?c_1$ owl:hasValue *?i* ; owl:onProperty $?p_1$. $?c_2$ owl:hasValue *?i* ; owl:onProperty $?p_2$. $?p_1$ rdfs:subPropertyOf $?p_2$.	$?c_1$ rdfs:subClassOf $?c_2$.
scm-svf1	$?c_1$ owl:someValuesFrom $?y_1$; owl:onProperty *?p* . $?c_2$ owl:someValuesFrom $?y_2$; owl:onProperty *?p* . $?y_1$ rdfs:subClassOf $?y_2$.	$?c_1$ rdfs:subClassOf $?c_2$.
scm-svf2	$?c_1$ owl:someValuesFrom *?y* ; owl:onProperty $?p_1$. $?c_2$ owl:someValuesFrom *?y* ; owl:onProperty $?p_2$. $?p_1$ rdfs:subPropertyOf $?p_2$.	$?c_1$ rdfs:subClassOf $?c_2$.
scm-avf1	$?c_1$ owl:allValuesFrom $?y_1$; owl:onProperty *?p* . $?c_2$ owl:allValuesFrom $?y_2$; owl:onProperty *?p* . $?y_1$ rdfs:subClassOf $?y_2$.	$?c_1$ rdfs:subClassOf $?c_2$.
scm-avf2	$?c_1$ owl:allValuesFrom *?y* ; owl:onProperty $?p_1$. $?c_2$ owl:allValuesFrom *?y* ; owl:onProperty $?p_2$. $?p_1$ rdfs:subPropertyOf $?p_2$.	$?c_1$ rdfs:subClassOf $?c_2$.
scm-int	*?c* owl:intersectionOf ($?c_1$...$?c_n$) .	*?c* rdfs:subClassOf $?c_1$...$?c_n$.
scm-uni	*?c* owl:unionOf ($?c_1$...$?c_n$) .	$?c_1$...$?c_n$ rdfs:subClassOf *?c* .

Table A.2
Rules containing only T-atoms in the body

ABody$(R) \neq \emptyset$, TBody$(R) = \emptyset$		
OWL2RL	**Antecedent**	**Consequent**
	assertional	
~~eq-ref~~	~~*?s ?p ?o* .~~	~~*?s* owl::sameAs *?s* . *?p* owl::sameAs *?p* . *?o* owl::sameAs *?o* .~~
eq-sym	*?x* owl::sameAs *?y* .	*?y* owl::sameAs *?x* .

Table A.3
Rules containing no T-atoms, but one A-atom in the body; we strike out the rule supporting the reflexivity of equality which we choose not to support since it adds a large bulk of trivial inferences to the set of materialised facts

TBody(R) $\neq \emptyset$ and |ABody(R)| $= 1$			
OWL2RL	**Antecedent**		**Consequent**
	terminological	*assertional*	
prp-dom	***?p*** rdfs:domain *?c* .	*?x ?p ?y* .	*?x* a *?c* .
prp-rng	***?p*** rdfs:range *?c* .	*?x ?p ?y* .	*?y* a *?c* .
prp-symp	***?p*** a owl:SymmetricProperty .	*?x ?p ?y* .	*?y ?p ?x* .
prp-spo1	***?p***$_1$ rdfs:subPropertyOf *?p*$_2$.	*?x ?p*$_1$ *?y* .	*?x ?p*$_2$ *?y* .
prp-eqp1	***?p***$_1$ owl:equivalentProperty *?p*$_2$.	*?x ?p*$_1$ *?y* .	*?x ?p*$_2$ *?y* .
prp-eqp2	*?p*$_1$ owl:equivalentProperty ***?p***$_2$.	*?x ?p*$_2$ *?y* .	*?x ?p*$_1$ *?y* .
prp-inv1	***?p***$_1$ owl:inverseOf *?p*$_2$.	*?x ?p*$_1$ *?y* .	*?y ?p*$_2$ *?x* .
prp-inv2	*?p*$_1$ owl:inverseOf ***?p***$_2$.	*?x ?p*$_2$ *?y* .	*?y ?p*$_1$ *?x* .
cls-int2	***?c*** owl:intersectionOf (*?c*$_1$...*?c*$_n$) .	*?x* a *?c* .	*?x* a *?c*$_1$...*?c*$_n$.
cls-uni	*?c* owl:unionOf (*?c*$_1$...***?c***$_i$...*?c*$_n$) .	*?x* a *?c*$_i$	*?x* a *?c* .
cls-svf2	*?x* owl:someValuesFrom owl:Thing ; owl:onProperty ***?p*** .	*?u ?p ?v* .	*?u* a *?x* .
cls-hv1	***?x*** owl:hasValue *?y* ; owl:onProperty ***?p*** .	*?u* a *?x* .	*?u ?p ?y* .
cls-hv2	*?x* owl:hasValue ***?y*** ; owl:onProperty *?p* .	*?u ?p ?y* .	*?u* a *?x* .
cax-sco	***?c***$_1$ rdfs:subClassOf *?c*$_2$.	*?x* a *?c*$_1$.	*?x* a *?c*$_2$.
cax-eqc1	***?c***$_1$ owl:equivalentClass *?c*$_2$.	*?x* a *?c*$_1$.	*?x* a *?c*$_2$.
cax-eqc2	*?c*$_1$ owl:equivalentClass ***?c***$_2$.	*?x* a *?c*$_2$.	*?x* a *?c*$_1$.

Table A.4
Rules containing some T-atoms and precisely one A-atom in the body with authoritative variable positions in bold

OWL2RL	**partially covered by recursive rule(s)**
scm-cls	*incomplete* for `owl:Thing` membership inferences
scm-sco	cax-sco
scm-eqc1	cax-eqc1, cax-eqc2
scm-eqc2	cax-sco
scm-op	*no unique assertional inferences*
scm-dp	*no unique assertional inferences*
scm-spo	prp-spo1
scm-eqp1	prp-eqp1, prp-eqp2
scm-eqp2	prp-spo1
scm-dom1	prp-dom, cax-sco
scm-dom2	prp-dom, prp-spo1
scm-rng1	prp-rng, cax-sco
scm-rng2	prp-rng, prp-spo1
scm-hv	prp-rng, prp-spo1
scm-svf1	*incomplete:* cls-svf1, cax-sco
scm-svf2	*incomplete:* cls-svf1, prp-spo1
scm-avf1	*incomplete:* cls-avf, cax-sco
scm-avf2	*incomplete:* cls-avf, prp-spo1
scm-int	cls-int2
scm-uni	cls-uni

Table A.5
Informal indication of the coverage in case of the omission of rules in Table A.2 wrt. inferencing over assertional knowledge by recursive application of rules in Table A.4: underlined rules are not supported, and thus we would encounter incompleteness wrt. assertional inference (would not affect a full OWL 2 RL/RDF reasoner which includes the underlined rules).

Head(R) $= \bot$		
OWL2RL	**Antecedent**	
	terminological	*assertional*
eq-diff1	-	*?x* owl:sameAs *?y* . *?x* owl:differentFrom *?y* .
eq-diff2	-	*?x* a owl:AllDifferent ; owl:members (*?z*$_1$...*?z*$_n$) . *?z*$_i$ owl:sameAs *?z*$_j$. (*i*≠*j*)
eq-diff3	-	*?x* a owl:AllDifferent ; owl:distinctMembers (*?z*$_1$...*?z*$_n$) . *?z*$_i$ owl:sameAs *?z*$_j$. (*i*≠*j*)
eq-irp [a]	-	*?x* owl:differentFrom *?x* .
prp-irp	***?p*** a owl:IrreflexiveProperty .	*?x ?p ?x* .
prp-asyp	***?p*** a owl:AsymmetricProperty	*?x ?p ?y* . *?y ?p ?x* .
prp-pdw	***?p***$_1$ owl:propertyDisjointWith ***?p***$_2$.	*?x ?p*$_1$ *?y* ; *?p*$_2$ *?y* .
prp-adp	*?x* a owl:AllDisjointProperties . *?x* owl:members (...***?p***$_i$...***?p***$_j$...) .	*?u ?p*$_i$ *?y* ; *?p*$_j$ *?y* . (*i*≠*j*)
prp-npa1	-	*?x* owl:sourceIndividual *?i*$_1$. *?x* owl:assertionProperty *?p* . *?x* owl:targetIndividual *?i*$_2$. *?i*$_1$ *?p* *?i*$_2$.
prp-npa2	-	*?x* owl:sourceIndividual *?i* . *?x* owl:assertionProperty *?p* . *?x* owl:targetValue *?lt* . *?i ?p ?lt* .
cls-nothing2	-	*?x* a owl:Nothing .
cls-com	***?c***$_1$ owl:complementOf ***?c***$_2$.	*?x* a *?c*$_1$, *?c*$_2$.
cls-maxc1	***?x*** owl:maxCardinality 0 . ***?x*** owl:onProperty ***?p*** .	*?u* a *?x* ; *?p ?y* .
cls-maxqc1	***?x*** owl:maxQualifiedCardinality 0 . ***?x*** owl:onProperty ***?p*** . ***?x*** owl:onClass ***?c*** .	*?u* a *?x* ; *?p ?y* . *?y* a *?c* .
cls-maxqc2	***?x*** owl:maxQualifiedCardinality 0 . ***?x*** owl:onProperty ***?p*** . ***?x*** owl:onClass owl:Thing .	*?u* a *?x* ; *?p ?y* .
cax-dw	***?c***$_1$ owl:disjointWith ***?c***$_2$.	*?x* a *?c*$_1$, *?c*$_2$.
cax-adc	*?x* a owl:AllDisjointClasses . *?x* owl:members (...***?c***$_i$...***?c***$_j$...) .	*?z* a *?c*$_i$, *?c*$_j$. (*i*≠*j*)
dt-not-type*	-	*?s ?p ?lt* . [b]

Table A.6
Constraint rules with authoritative variables in bold

[a] This is a non-standard rule we support since we do not materialise the reflexive `owl:sameAs` statements required by eq-diff1.

[b] Where *?lt* is an ill-typed literal: this is a non-standard version of dt-not-type which captures approximately the same inconsistencies, but without requiring rule dt-type2.

Appendix B. Ranking algorithms

Herein, we provide the detailed algorithms used for extracting, preparing and ranking the source level graph. In particular, we provide the algorithms for parallel extraction and preparation of the sub-graphs on the slave machines: (i) extracting the source-level graph (Algorithm 2); (ii) rewriting the graph with respect to redirect information (Algorithm 3); (iii) pruning the graph with respect to the list of valid contexts (Algorithm 4). Subsequently, the subgraphs are merge-sorted onto the master machine, which calculates the PageRank scores for the vertices (sources) in the graph as follows: (i) count the vertices and derive a list of dangling-nodes (Algorithm 5); (ii) perform the power iteration algorithm to calculate the ranks (Algorithm 6).

The algorithms are heavily based on on-disk operations: in the algorithms, we use `typewriter font` to denote on-disk operations and files. In particular, the algorithms are all based around sorting/scanning and *merge-joins*: a merge-join requires two or more lists of tuples to be sorted by a common join element, where the tuples can be iterated over in sorted order with the iterators kept "aligned" on the join element; we mark use of merge-joins in the algorithms using "m-join" in the comments.

Algorithm 2 Extract raw sub-graph

Require: QUADS: $\mathtt{Q}$ /* $\langle s,p,o,c\rangle_{0\ldots n}$ *sorted by* c */
1: $links \coloneqq \{\}$, $\mathtt{L} \coloneqq \{\}$
2: **for all** $\langle s,p,o,c\rangle_i \in \mathtt{Q}$ **do**
3: **if** $c_i \neq c_{i-1}$ **then**
4: $\mathtt{write}(links, \mathtt{L})$
5: $links \coloneqq \{\}$
6: **end if**
7: **for all** $u \in \mathbf{U} \mid u \in \{s_i, p_i, o_i\} \wedge u \neq c_i$ **do**
8: $links \coloneqq links \cup \{\langle c_i, u\rangle\}$
9: **end for**
10: **end for**
11: $\mathtt{write}(links, \mathtt{L})$
12: **return** $\mathtt{L}$ /* *unsorted on-disk outlinks* */

Algorithm 3 Rewrite graph wrt. redirects

Require: RAW LINKS: $\mathtt{L}$ /* $\langle u,v\rangle_{0\ldots m}$ *unsorted* */
Require: REDIRECTS: $\mathtt{R}$ /* $\langle f,t\rangle_{0\ldots n}$ *sorted by unique* f */
Require: MAX. ITERATIONS: I /* *typ.* $I \coloneqq 5$ */
1: $\mathtt{R}^- \coloneqq \mathtt{sortUnique}^-(\mathtt{L})$ /* *sort by* v */
2: $i \coloneqq 0$; $\mathtt{G}^-_\delta \coloneqq \mathtt{G}^-$
3: **while** $\mathtt{G}^-_\delta \neq \emptyset \wedge i < I$ **do**
4: $k \coloneqq 0$; $\mathtt{G}^-_i \coloneqq \{\}$; $\mathtt{G}^-_{tmp} \coloneqq \{\}$
5: **for all** $\langle u,v\rangle_j \in \mathtt{G}^-_\delta$ **do**
6: **if** $j = 0 \vee v_j \neq v_{j-1}$ **then**
7: $rewrite \coloneqq \bot$
8: **if** $\exists \langle f,t\rangle_k \in \mathtt{R} \mid f_k = v_j$ **then** /* *m-join* */
9: $rewrite \coloneqq t_k$
10: **end if**
11: **end if**
12: **if** $rewrite = \bot$ **then**
13: $\mathtt{write}(\langle u,v\rangle_j, \mathtt{G}^-_i)$
14: **else if** $rewrite \neq u_j$ **then**
15: $\mathtt{write}(\langle u_j, rewrite\rangle, \mathtt{G}^-_{tmp})$
16: **end if**
17: **end for**
18: $i{+}{+}$; $\mathtt{G}^-_\delta \coloneqq \mathtt{G}^-_{tmp}$;
19: **end while**
20: $\mathtt{G}^-_r \coloneqq \mathtt{mergeSortUnique}(\{\mathtt{G}^-_0, \ldots, \mathtt{G}^-_{i-1}\})$
21: **return** $\mathtt{G}^-_r$ /* *on-disk, rewritten, sorted inlinks* */

Algorithm 4 Prune graph by contexts

Require: NEW LINKS: $\mathtt{G}^-_r$ /* $\langle u,v\rangle_{0\ldots m}$ *sorted by* v */
Require: CONTEXTS: $\mathtt{C}$ /* $\langle c_1, \ldots, c_n\rangle$ *sorted* */
1: $\mathtt{G}^-_p \coloneqq \{\}$
2: **for all** $\langle u,v\rangle_i \in \mathtt{G}^-_r$ **do**
3: **if** $i = 0 \vee c_i \neq c_{i-1}$ **then**
4: $write \coloneqq$ **false**
5: **if** $c_j \in \mathtt{C}$ **then** /* *m-join* */
6: $write \coloneqq$ **true**
7: **end if**
8: **end if**
9: **if** $write$ **then**
10: $\mathtt{write}(\langle u,v\rangle_i, \mathtt{G}^-_p)$
11: **end if**
12: **end for**
13: **return** $\mathtt{G}^-_p$ /* *on-disk, pruned, rewritten, sorted inlinks* */

Algorithm 5 Analyse graph

Require: OUT LINKS: G /* $\langle u, v\rangle_{0\ldots n}$ *sorted by* u */
Require: IN LINKS: $\mathtt{G}^-$ /* $\langle w, x\rangle_{0\ldots n}$ *sorted by* x */
1: $V := 0$ /* *vertex count* */
2: $u_{-1} := \bot$
3: **for all** $\langle u, v\rangle_i \in \mathtt{G}$ **do**
4: **if** $i = 0 \vee u_i \neq u_{i-1}$ **then**
5: $V{+}{+}$
6: **for all** $\langle w, x\rangle_j \in \mathtt{G}^- \mid u_{i-1} < x_j < u_i$ **do** /* *m-join* */
7: $V{+}{+}$; $\mathtt{write}(x_j, \mathtt{DANGLE})$
8: **end for**
9: **end if**
10: **end for**
11: **for all** $\langle w, x\rangle_j \in \mathtt{G}^- \mid x_j > u_n$ **do** /* *m-join* */
12: $V{+}{+}$; $\mathtt{write}(x_j, \mathtt{DANGLE})$
13: **end for**
14: **return** DANGLE /* *sorted, on-disk list of dangling vertices* */
15: **return** V /* *number of unique vertices* */

Algorithm 6 Rank graph

Require: OUT LINKS: G /* $\langle u, v\rangle_{0\ldots m}$ *sorted by* u */
Require: DANGLING: DANGLE /* $\langle y_0, \ldots y_n\rangle$ *sorted* */
Require: MAX. ITERATIONS: I /* *typ.* $I := 10$ */
Require: DAMPING FACTOR: D /* *typ.* $D := 0.85$ */
Require: VERTEX COUNT: V
1: $i := 0$; $initial := \frac{1}{V}$; $min := \frac{1-D}{V}$
2: $dangle := D * initial * |\mathtt{DANGLE}|$
3: /* *GENERATE UNSORTED VERTEX/RANK PAIRS* */
4: **while** $i < I$ **do**
5: $min_i := min + \frac{dangle}{V}$; $\mathtt{PR}_{tmp} := \{\}$;
6: **for all** $z_j \in \mathtt{DANGLE}$ **do** /* z_j *has no outlinks* */
7: $\mathtt{write}(\langle z_j, min_i\rangle, \mathtt{PR}_{tmp})$
8: **end for**
9: $out_j := \{\}$; $rank := initial$
10: **for all** $\langle u, v\rangle_j \in \mathtt{G}$ **do** /* *get ranks thru strong links* */
11: **if** $j \neq 0 \wedge u_j \neq u_{j-1}$ **then**
12: $\mathtt{write}(\langle u_{j-1}, min_i\rangle, \mathtt{PR}_{tmp})$
13: **if** $i \neq 0$ **then**
14: $rank := \mathtt{getRank}(u_{j-1}, \mathtt{PR}_i)$ /* *m-join* */
15: **end if**
16: **for all** $v_k \in out$ **do**
17: $\mathtt{write}(\langle v_k, \frac{rank}{|out|}\rangle, \mathtt{PR}_{tmp})$
18: **end for**
19: **end if**
20: $out_j := out_j \cup \{v_j\}$
21: **end for**
22: **do** lines 12–18 for last $u_{j-1} := u_m$
23: /* *SORT/AGGREGATE VERTEX/RANK PAIRS* */
24: $\mathtt{PR}_{i+1} := \{\}$; $dangle := 0$
25: **for all** $\langle z, r\rangle_j \in \mathtt{sort}(\mathtt{PR}_{tmp})$ **do**
26: **if** $j \neq 0 \wedge z_j \neq z_{j-1}$ **then**
27: **if** $z_{j-1} \in \mathtt{DANGLE}$ **then** /* *m-join* */
28: $dangle := dangle + rank$
29: **end if**
30: $\mathtt{write}(\langle z_{j-1}, rank\rangle, \mathtt{PR}_{i+1})$
31: **end if**
32: $rank := rank + r_j$
33: **end for**
34: **do** lines 27–30 for last $z_{j-1} := z_l$
35: $i{+}{+}$ /* *iterate* */
36: **end while**
37: **return** $\mathtt{PR}_I$ /* *on-disk, sorted vertex/rank pairs* */

Appendix C. Prefixes

In Table C.1, we provide the prefixes used throughout the paper.

Prefix	URI
Vocabulary prefixes	
aifb:	http://www.aifb.kit.edu/id/
b2r2008:	http://bio2rdf.org/bio2rdf-2008.owl#
bio2rdf:	http://bio2rdf.org/bio2rdf_resource:
contact:	http://www.w3.org/2000/10/swap/pim/contact#
dct:	http://purl.org/dc/terms/
ecs:	http://rdf.ecs.soton.ac.uk/ontology/ecs#
fb:	http://rdf.freebase.com/ns/
foaf:	http://xmlns.com/foaf/0.1/
frbr:	http://purl.org/vocab/frbr/core#
geonames:	http://www.geonames.org/ontology#
geospecies:	http://rdf.geospecies.org/ont/geospecies#
lld:	http://linkedlifedata.com/resource/entrezgene/
mo:	http://purl.org/ontology/mo/
opiumfield:	http://rdf.opiumfield.com/lastfm/spec#
owl:	http://www.w3.org/2002/07/owl#
po:	http://purl.org/ontology/po/
pres:	http://www.w3.org/2004/08/Presentations.owl#
rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs:	http://www.w3.org/2000/01/rdf-schema#
sc:	http://umbel.org/umbel/sc/
sioc:	http://rdfs.org/sioc/ns#
skos:	http://www.w3.org/2004/02/skos/core#
wgs84:	http://www.w3.org/2003/01/geo/wgs84_pos#
wn:	http://xmlns.com/wordnet/1.6/
Instance-data prefixes	
dav:	http://www.openlinksw.com/dataspace/org.../dav#
enwiki:	http://en.wikipedia.org/wiki/
kingdoms:	http://lod.geospecies.org/kingdoms/
vperson:	http://virtuoso.openlinksw.com/dataspace/person/

Table C.1
CURIE prefixes used in this paper