A general framework for representing, reasoning and querying with annotated Semantic Web data

Antoine Zimmermann [c,*], Nuno Lopes [a], Axel Polleres [a], Umberto Straccia [b]

[a] Digital Enterprise Research Institute, National University of Ireland, Galway, Ireland
[b] Istituto di Scienza e Tecnologie dell'Informazione (ISTI – CNR), Pisa, Italy
[c] INSA-Lyon, LIRIS, UMR5205, F-69621, France

ABSTRACT

We describe a generic framework for representing and reasoning with annotated Semantic Web data, a task becoming more important with the recent increased amount of inconsistent and non-reliable meta-data on the Web. We formalise the annotated language, the corresponding deductive system and address the query answering problem. Previous contributions on specific RDF annotation domains are encompassed by our unified reasoning formalism as we show by instantiating it on (i) temporal, (ii) fuzzy, and (iii) provenance annotations. Moreover, we provide a generic method for combining multiple annotation domains allowing to represent, e.g., temporally-annotated fuzzy RDF. Furthermore, we address the development of a query language – AnQL – that is inspired by SPARQL, including several features of SPARQL 1.1 (subqueries, aggregates, assignment, solution modifiers) along with the formal definitions of their semantics.

1. Introduction

RDF (Resource Description Framework) [1] is the widely used representation language for the Semantic Web and the Web of Data. RDF exposes data as triples, consisting of *subject*, *predicate* and *object*, stating that *subject* is related to *object* by the *predicate* relation. Several extensions of RDF were proposed in order to deal with time [2–4], truth or imprecise information [5,6], trust [7,8] and provenance [9]. All these proposals share a common approach of extending the RDF language by attaching meta-information about the RDF graph or triples. RDF Schema (RDFS) [10] is the specification of a restricted vocabulary that allows one to deduce further information from existing RDF triples. SPARQL [11] is the W3C-standardised query language for RDF.

In this paper, we present an extension of the RDF model to support meta-information in the form of annotations of triples. We specify the semantics by conservatively extending the RDFS semantics and provide a deductive system for Annotated RDFS. Further, we define a query language that extends SPARQL and include advanced features such as aggregates, nested queries and variable assignments, which are part of the not-yet-standardised SPARQL 1.1 specification. The present paper is based on and ex-

tends two previously published articles introducing Annotated RDFS [12] and AnQL (our SPARQL extension) [13]. In addition to improving the descriptions of this existing body of work, we provide the following novelties:

1. We introduce a use case scenario that better reflects a realistic example of how annotations can be used.
2. We detail three concrete domains of annotations (temporal, fuzzy, provenance) that were only sketched in our previous publications.
3. We present a detailed and systematic approach for combining multiple annotation domains into a new single complex domain; this represents the most significant novel contribution of the paper.
4. We discuss the integration of annotated triples with standard, non-annotated triples, as well as the integration of data using different annotation domains.
5. We describe a prototype implementation.

Section 2 gives preliminary definitions of the RDFS semantics and query answering, restricting ourselves to the sublanguage ρdf. Our extension of RDF is presented in Section 3 together with essential examples of primitive domains. Our extension of SPARQL, is presented in Section 4. Furthermore, Section 5 presents a discussion of important issues with respect to specific domains and their combination. Finally, Section 6 describes our prototype implementation.

* Corresponding author.
E-mail addresses: antoine.zimmermann@insa-lyon.fr (A. Zimmermann), nuno.lopes@deri.org (N. Lopes), axel.polleres@deri.org (A. Polleres), straccia@isti.cnr.it (U. Straccia).

1.1. Related work

The basis for Annotated RDF were first established by Udrea et al. [14,15], where they define triples annotated with values taken from a *finite partial order*. In their work, triples are of the form $(s, p : \lambda, o)$, where the property, rather than the triple is annotated. We instead rely on a richer, not necessarily finite, structure and provide additional inference capabilities to [15], such as a more involved propagation of annotation values through schema triples. For instance, in the temporal domain, from (a, sc, b): [2,6] and (b, sc, c): [3,8], we will infer (a, sc, c): [3,6] (sc is the subclass property). Essentially, Udrea et al. do not provide an operation to combine the annotation in such inferences, while the algebraic structures we consider support such operations. Also, they require specific algorithms, while we show that a simple extension to the classical RDF inference rules is sufficient. The query language presented in this paper consists of conjunctive queries and, while SPARQL's Basic Graph Patterns are compared to their conjunctive queries, they do not consider extending SPARQL with the possibility of querying annotations. Furthermore, OPTIONAL, UNION and FILTER SPARQL queries are not considered which results in a subset of SPARQL that can be directly translated into their previously presented conjunctive query system.

Adding annotations to logical statements was already proposed in the logic programming realm in which Kifer and Subrahmanian [16] present a similar approach, where atomic formulas are annotated with a value taken from a lattice of annotation values, an annotation variable or a complex annotation, i.e., a function applied to annotation values or variables. Similarly, we can relate our work to annotated relational databases, especially Green et al. [17] who provides a similar framework for the relational algebra. After presenting a generic structure for annotations, they focus more specifically on the provenance domain. The specificities of the relation algebra, especially Closed World Assumption, allows them to define a slightly more general structure for annotation domains, namely semiring (as opposed to the residuated lattice in our initial approach [12,13]). In relation to our rule-based RDFS Reasoning, it should be mentioned that Green et al. [17] also provide an algorithm that can decide ground query answers for annotated Datalog, which might be used for RDFS rules; general query answering or materialisation though might not terminate, due to the general structure of annotations, in their case. Karvounarakis et al. [18] extend the work of [17] towards various annotations – not only provenance, but also confidence, rank, etc. – but do not specifically discuss their combinations.

In a generic approach mostly focused on Semantic Web ontologies, Baader et al. [19,20] describe a framework to delimit sub-ontologies according to access restrictions, where access rights are modelled by labels from a finite lattice, attached to ontological axioms. The approach is not tied to a particular ontology language, thereby could be used on RDF datasets. As the algebraic structure is fully compatible with our framework, we can consider that this case is covered by our work. However, the goal of their work is not to provide means to reason and query over the access right annotations but rather to enforce the access rights on subsets of the axioms.

For the Semantic Web, several extensions of RDF were proposed in order to deal with specific domains such as truth of imprecise information [5,21,22,6], time [2–4], trust [7,8] and provenance [9]. These approaches are detailed in the following paragraphs.

Straccia [6], presents Fuzzy RDF in a general setting where triples are annotated with a degree of truth in [0,1]. For instance, "Rome is a big city to degree 0.8" can be represented with (Rome,type,BigCity): 0.8; the annotation domain is [0,1]. For the query language, it formalises conjunctive queries. Other similar approaches for Fuzzy RDF [5,21,22] provide the syntax and seman-

tics, along with RDF and RDFS interpretations of the annotated triples. In [22] the author describes an implementation strategy that relies on translating the Fuzzy triples into plain RDF triples by using reification. However these works focus mostly on the representation format and the query answering problem is not addressed.

Gutiérrez et al. [2] presents the definitions of Temporal RDF, including reduction of the semantics of Temporal RDF graphs to RDF graphs, a sound and complete inference system and shows that entailment of Temporal graphs does not yield extra complexity than RDF entailment. Our Annotated RDFS framework encompasses this work by defining the temporal domain. They present conjunctive queries with built-in predicates as the query language for Temporal RDF, although they do not consider full SPARQL. Pugliese et al. [3] presents an optimised indexing schema for Temporal RDF, the notion of normalised Temporal RDF graph and a query language for these graphs based on SPARQL. The indexing scheme consists of clustering the RDF data based on their temporal distance, for which several metrics are given. For the query language they only define conjunctive queries, thus ignoring some of the more advanced features of SPARQL. Tappolet and Bernstein [4] present another approach to the implementation of Temporal RDF, where each temporal interval is represented as a named graph [23] containing all triples valid in that time period. Information about temporal intervals, such as their relative relations, start and end points, is asserted in the default graph. The τ-SPARQL query language allows to query the temporal RDF representation using an extended SPARQL syntax that can match the graph pattern against the snapshot of a temporal graph at any given time point and allows to query the start and endpoints of a temporal interval, whose values can then be used in other parts of the query.

SPARQL extensions towards querying trust have been presented by Hartig [7]. Hartig introduces a trust aware query language, tSPARQL, that includes a new constructor to access the trust value of a graph pattern. This value can then be used in other statements such as FILTERs or ORDER. Also in the setting of trust management, Schenk [8] defines a *bilattice* structure to model *trust* relying on the dimensions of knowledge and truth. The defined knowledge about trust in information sources can then be used to compute the trust of an inferred statement. An extension towards OWL is presented but there is no query language defined. Finally, this approach is used to resolve inconsistencies in ontologies arising from connecting multiple data sources.

In [9] the authors also present a generic extension of RDF to represent *meta information*, mostly focused on provenance and uncertainty. Such meta information is stored using named graphs and their extended semantics of RDF, denoted RDF^+, assumes a predefined vocabulary to be interpreted as the meta information. However they do not provide an extension of the RDFS inference rules or any operations for combining meta information. The authors also provide an extension of the SPARQL query language, considering an additional expression that enables querying the RDF meta information.

Our initial approach of using residuated lattices as the structure for representing annotations [12,13] was extended to the more general semiring structure by Buneman and Kostylev [24]. This paper also shows that, once the RDFS inferences of an RDF graph have been computed for a specific domain, it is possible to reuse these inferences if the graph is annotated with a different domain. Based on this result the authors define a universal domain which is possible to transform to other domains by applying the corresponding transformations.

Aidan Hogan's thesis [25, Chapter 6] provides a framework for a specific combination of annotations (authoritativeness, rank, blacklisting) within RDFS and (a variant of) OWL 2 RL. This work is orthogonal to ours, in that it does not focus on aspects of query

answering, or providing a generic framework for combinations of annotations, but rather on scalable and efficient algorithms for materialising inferences for the specific combined annotations under consideration.

2. Preliminaries – classical RDF and RDFS

In this section we present notions and definitions that are necessary for our discussions later. First we give a short overview of RDF and RDFS.

2.1. Syntax

Consider pairwise disjoint alphabets **U**, **B**, and **L** denoting, respectively, *URI references*, *blank nodes* and *literals*.[1] We call the elements in **UBL** (**B**) *terms* (*variables*, denoted x, y, z). An *RDF triple* is $\tau = (s, p, o) \in \textbf{UBL} \times \textbf{U} \times \textbf{UBL}$.[2] We call s the *subject*, p the *predicate*, and o the *object*. A *graph* G is a set of triples, the *universe* of G, *universe*(G), is the set of elements in **UBL** that occur in the triples of G, the *vocabulary* of G, $voc(G)$, is *universe*$(G) \cap \textbf{UL}$.

We rely on a fragment of RDFS, called ρdf [26], that covers essential features of RDFS. ρdf is defined as the following subset of the RDFS vocabulary: $\rho\text{df} = \{\text{sp, sc, type, dom, range}\}$. Informally, (i) (p, sp, q) means that property p is a *subproperty* of property q; (ii) (c, sc, d) means that class c is a *subclass* of class d; (iii) (a, type, b) means that a is of *type* b; (iv) (p, dom, c) means that the *domain* of property p is c; and (v) (p, range, c) means that the *range* of property p is c. In what follows we define a *map* as a function $\mu : \textbf{UBL} \to \textbf{UBL}$ preserving URIs and literals, i.e., $\mu(t) = t$, for all $t \in \textbf{UL}$. Given a graph G, we define $\mu(G) = \{(\mu(s), \mu(p), \mu(o)) \mid (s, p, o) \in G\}$. We speak of a map μ from G_1 to G_2, and write $\mu : G_1 \to G_2$, if μ is such that $\mu(G_1) \subseteq G_2$.

2.2. Semantics

An *interpretation* $\mathcal{I}$ over a vocabulary V is a tuple $\mathcal{I} = \langle \Delta_R, \Delta_P, \Delta_C, \Delta_L, P[\![\cdot]\!], C[\![\cdot]\!], \cdot^{\mathcal{I}} \rangle$, where $\Delta_R, \Delta_P, \Delta_C, \Delta_L$ are the interpretation domains of $\mathcal{I}$, which are finite non-empty sets, and $P[\![\cdot]\!], C[\![\cdot]\!], \cdot^{\mathcal{I}}$ are the interpretation functions of $\mathcal{I}$. They have to satisfy:

1. Δ_R are the resources (the domain or universe of $\mathcal{I}$);
2. Δ_P are property names (not necessarily disjoint from Δ_R);
3. $\Delta_C \subseteq \Delta_R$ are the classes;
4. $\Delta_L \subseteq \Delta_R$ are the literal values and contains $\textbf{L} \cap V$;
5. $P[\![\cdot]\!]$ is a function $P[\![\cdot]\!] : \Delta_P \to 2^{\Delta_R \times \Delta_R}$;
6. $C[\![\cdot]\!]$ is a function $C[\![\cdot]\!] : \Delta_C \to 2^{\Delta_R}$;
7. $\cdot^{\mathcal{I}}$ maps each $t \in \textbf{UL} \cap V$ into a value $t^{\mathcal{I}} \in \Delta_R \cup \Delta_P$, and such that $\cdot^{\mathcal{I}}$ is the identity for plain literals and assigns an element in Δ_R to each element in **L**.

An interpretation $\mathcal{I}$ is a *model* of a ground graph G, denoted $\mathcal{I} \models G$, if and only if $\mathcal{I}$ is an interpretation over the vocabulary $\rho\text{df} \cup \textit{universe}(G)$ that satisfies the following conditions:

Simple:
1. For each $(s, p, o) \in G$, $p^{\mathcal{I}_A} \in \Delta_P$ and $(s^{\mathcal{I}_A}, o^{\mathcal{I}_A}) \in P[\![p^{\mathcal{I}_A}]\!]$.

Subproperty:
1. $P[\![\text{sp}^{\mathcal{I}_A}]\!]$ is transitive over Δ_P;
2. if $(p, q) \in P[\![\text{sp}^{\mathcal{I}_A}]\!]$ then $p, q \in \Delta_P$ and $P[\![p]\!] \subseteq P[\![q]\!]$.

Subclass:
1. $P[\![\text{sc}^{\mathcal{I}_A}]\!]$ is transitive over Δ_C;
2. if $(c, d) \in P[\![\text{sc}^{\mathcal{I}_A}]\!]$ then $c, d \in \Delta_C$ and $C[\![c]\!] \subseteq C[\![d]\!]$.

Typing I:
1. $x \in C[\![c]\!]$ if and only if $(x, c) \in P[\![\text{type}^{\mathcal{I}_A}]\!]$;
2. if $(p, c) \in P[\![\text{dom}^{\mathcal{I}_A}]\!]$ and $(x, y) \in P[\![p]\!]$ then $x \in C[\![c]\!]$;
3. if $(p, c) \in P[\![\text{range}^{\mathcal{I}_A}]\!]$ and $(x, y) \in P[\![p]\!]$ then $y \in C[\![c]\!]$.

Typing II:
1. For each $\text{e} \in \rho\text{df}$, $\text{e}^{\mathcal{I}_A} \in \Delta_P$;
2. if $(p, c) \in P[\![\text{dom}^{\mathcal{I}_A}]\!]$ then $p \in \Delta_P$ and $c \in \Delta_C$;
3. if $(p, c) \in P[\![\text{range}^{\mathcal{I}_A}]\!]$ then $p \in \Delta_P$ and $c \in \Delta_C$;
4. if $(x, c) \in P[\![\text{type}^{\mathcal{I}_A}]\!]$ then $c \in \Delta_C$.

Entailment among ground graphs G and H is as usual. Now, $G \models H$, where G and H may contain blank nodes, if and only if for any grounding G' of G there is a grounding H' of H such that $G' \models H'$.[3]

Remark 2.1. In [26], the authors define two variants of the semantics: the default one includes reflexivity of $P[\![\text{sp}^{\mathcal{I}}]\!]$ (resp. $C[\![\text{sc}^{\mathcal{I}}]\!]$) over Δ_P (resp. Δ_C) but we are only considering the alternative semantics presented in [26, Definition 4] which omits this requirement. Thus, we do not support an inference such as $G \models (a, \text{sc}, a)$, which anyway are of marginal interest.

Remark 2.2. In a First-Order Logic (FOL) setting, we may interpret classes as unary predicates, and (RDF) predicates as binary predicates. Then

1. a subclass relation between class c and d may be encoded as the formula $\forall x.c(x) \Rightarrow d(x)$;
2. a subproperty relation between property p and q may be encoded as $\forall x \forall y.p(x, y) \Rightarrow q(x, y)$;
3. domain and range properties may be represented as: $\forall x \forall y.p(x, y) \Rightarrow c(x)$ and $\forall x \forall y.p(x, y) \Rightarrow c(y)$;
4. the transitivity of a property can be represented as $\forall x \forall y \exists z.(p(x, z) \land p(z, y)) \Rightarrow p(x, y)$.

Although this remark is trivial, we will see that it will play an important role in the formalisation of Annotated RDFS.

2.3. Deductive system

In what follows, we provide the sound and complete deductive system for our language derived from [26]. The system is arranged in groups of rules that captures the semantic conditions of models. In every rule, A, B, C, X, and Y are meta-variables representing elements in **UBL** and D, E represent elements in **UL**. The rules are as follows:

1. **Simple:**

$$(a) \quad \frac{G}{G'} \quad \text{for a map} \, \mu : G' \to G \quad (b) \quad \frac{G}{G'} \quad \text{for } G' \subseteq G$$

2. **Subproperty:**

$$(a) \quad \frac{(A, \text{sp}, B), (B, \text{sp}, C)}{(A, \text{sp}, C)} \quad (b) \quad \frac{(D, \text{sp}, E), (X, D, Y)}{(X, E, Y)}$$

3. **Subclass:**

$$(a) \quad \frac{(A, \text{sc}, B), (B, \text{sc}, C)}{(A, \text{sc}, C)} \quad (b) \quad \frac{(A, \text{sc}, B), (X, \text{type}, A)}{(X, \text{type}, B)}$$

---

[1] We assume **U**, **B**, and **L** fixed, and for ease we will denote unions of these sets simply concatenating their names.

[2] As in [26] we allow literals for s.

[3] A grounding G' of graph G is obtained, as usual, by replacing variables in G with terms in **UL**.

4. **Typing:**

$$(a) \quad \frac{(D, \mathsf{dom}, B), (X, D, Y)}{(X, \mathsf{type}, B)} \quad (b) \quad \frac{(D, \mathsf{range}, B), (X, D, Y)}{(Y, \mathsf{type}, B)}$$

5. **Implicit Typing:**

$$(a) \quad \frac{(A, \mathsf{dom}, B), (D, \mathsf{sp}, A), (X, D, Y)}{(X, \mathsf{type}, B)}$$

$$(b) \quad \frac{(A, \mathsf{range}, B), (D, \mathsf{sp}, A), (X, D, Y)}{(Y, \mathsf{type}, B)}$$

A reader familiar with [26] will notice that these rules are as rules 1–5 of [26] (which has 7 rules). We excluded the rules handling reflexivity (rules 6–7) which are not needed to answer queries. Furthermore, as noted in [26], the "Implicit Typing" rules are a necessary addition to the rules presented in [27] for complete RDFS entailment. These represent the case when variable A in (D, sp, A) and (A, dom, B) or (A, range, B), is a property implicitly represented by a blank node.

We denote with $\{\tau_1, \ldots, \tau_n\} \vdash_{\mathsf{RDFS}} \tau$ that the consequence τ is obtained from the premise $\tau_1, \ldots, \tau_n$ by applying one of the inference rules 2–5 above. Note that $n \in \{2, 3\}$. $\vdash_{\mathsf{RDFS}}$ is extended to the set of all RDFS rules as well, in which case $n \in \{1, 2, 3\}$.

If a graph G' can be obtained by recursively applying rules 1-5 from a graph G, the sequence of applied rules is called a *proof*, denoted $G \vdash G'$, of G' from G. The following proposition shows that our proof mechanism is sound and complete w.r.t. the ρdf semantics:

Proposition 2.1 (Soundness and completeness [26]). *Inference* $\vdash$ *based on rules* 1–5 *as of* [26] *and applied to our semantics defined above is sound and complete for* $\models$, *that is,* $G \vdash G'$ *if and only if* $G \models G'$.

Proposition 2.2 [26]. *Assume* $G \vdash G'$ *then there is a proof of* G' *from* G *where the rule* (1a) *is used at most once and at the end.*

Finally, the *closure* of a graph G is defined as $cl(G) = \{\tau \mid G \vdash^* \tau\}$, where $\vdash^*$ is as $\vdash$ except that rule (1a) is excluded. Note that the size of the closure of G is polynomial in the size of G and that the closure is *unique*. Now we can prove that:

Proposition 2.3. $G \vdash G'$ *if and only if* $G' \subseteq cl(G)$ *or* G' *is obtained from* $cl(G)$ *by applying rule* (1a).

2.4. Query answering

Concerning query answering, we are inspired by Gutiérrez et al. [28] and the logic programming setting and we assume that a RDF graph G is *ground*, that is, all blank nodes have been skolemised, i.e., replaced with terms in **UL**.

A *query* is of the rule-like form

$$q(\overline{\mathbf{x}}) \leftarrow \exists \overline{\mathbf{y}}. \varphi(\overline{\mathbf{x}}, \overline{\mathbf{y}}),$$

where $q(\overline{\mathbf{x}})$ is the *head* and $\exists \overline{\mathbf{y}}. \varphi(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ is the *body* of the query, which is a conjunction (we use the symbol "," to denote conjunction in the rule body) of triples $\tau_i (1 \leqslant i \leqslant n)$. $\overline{\mathbf{x}}$ is a vector of variables occurring in the body, called the *distinguished variables*, $\overline{\mathbf{y}}$ are so-called *non-distinguished variables* and are distinct from the variables in $\overline{\mathbf{x}}$, each variable occurring in τ_i is either a distinguished or a non-distinguished variable. If clear from the context, we may omit the existential quantification $\exists \overline{\mathbf{y}}$.

In a query, we allow built-in triples of the form (s, p, o), where p is a built-in predicate taken from a reserved vocabulary and having a *fixed interpretation*. We generalise the built-ins to any n-ary pred-

icate p, where p's arguments may be ρ df variables, values from **UL**, and p has a fixed interpretation. We will assume that the evaluation of the predicate can be decided in finite time. For convenience, we write "functional predicates"[4] as *assignments* of the form $x := f(\overline{\mathbf{z}})$ and assume that the function $f(\overline{\mathbf{z}})$ is safe. We also assume that a non functional built-in predicate $p(\overline{\mathbf{z}})$ should be safe as well.

A query example is:

$$q(x, y) \leftarrow (y, \mathtt{created}, x), (y, \mathtt{type}, \mathtt{Italian}),$$
$$(x, \mathtt{exhibitedAt}, \mathtt{Uffizi}),$$

having intended meaning to retrieve all the artefacts x created by Italian artists y, being exhibited at Uffizi Gallery.

In order to define an *answer* to a query we introduce the following:

Definition 2.1 (*Query instantiation*). Given a vector $\overline{\mathbf{x}} = \langle x_1, \ldots, x_k \rangle$ of variables, a *substitution* over $\overline{\mathbf{x}}$ is a vector of terms $\overline{\mathbf{t}}$ replacing variables in $\overline{\mathbf{x}}$ with terms of **UBL**. Then, given a query $q(\overline{\mathbf{x}}) \leftarrow \exists \overline{\mathbf{y}}. \varphi(\overline{\mathbf{x}}, \overline{\mathbf{y}})$, and two substitutions $\overline{\mathbf{t}}, \overline{\mathbf{t}'}$ over $\overline{\mathbf{x}}$ and $\overline{\mathbf{y}}$, respectively, the *query instantiation* $\varphi(\overline{\mathbf{t}}, \overline{\mathbf{t}'})$ is derived from $\varphi(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ by replacing $\overline{\mathbf{x}}$ and $\overline{\mathbf{y}}$ with $\overline{\mathbf{t}}$ and $\overline{\mathbf{t}'}$, respectively.

Note that a query instantiation is an RDF graph.

Definition 2.2 (*Entailment*). Given a graph G, a query $q(\overline{\mathbf{x}}) \leftarrow \exists \overline{\mathbf{y}}. \varphi(\overline{\mathbf{x}}, \overline{\mathbf{y}})$, and a vector $\overline{\mathbf{t}}$ of terms in *universe*(G), we say that $q(\overline{\mathbf{t}})$ is *entailed* by G, denoted $G \models q(\overline{\mathbf{t}})$, if and only if in any model $\mathcal{I}$ of G, there is a vector $\overline{\mathbf{t}'}$ of terms in *universe*(G) such that $\mathcal{I}$ is a model of the query instantiation $\varphi(\overline{\mathbf{t}}, \overline{\mathbf{t}'})$.

Definition 2.3. If $G \models q(\overline{\mathbf{t}})$ then $\overline{\mathbf{t}}$ is called an *answer* to q. The *answer set* of q w.r.t. G is defined as $ans(G, q) = \{\overline{\mathbf{t}} \mid G \models q(\overline{\mathbf{t}})\}$.

We next show how to compute the answer set. The following can be shown:

Proposition 2.4. *Given a graph* G, $\overline{\mathbf{t}}$ *is an answer to* q *if and only if there exists an instantiation* $\varphi(\overline{\mathbf{t}}, \overline{\mathbf{t}'})$ *that is true in the closure of* G (*i.e., all triples in* $\varphi(\overline{\mathbf{t}}, \overline{\mathbf{t}'})$ *are in* $cl(G)$).

Therefore, we have a simple method to determine $ans(G, q)$. Compute the closure $cl(G)$ of G and store it into a database, e.g., using the method [29]. It is easily verified that any query can be mapped into an SQL query over the underlying database schema. Hence, $ans(G, q)$ can be determined by issuing such an SQL query to the database.

3. RDFS with annotations

This section presents the extension to RDF towards generic annotations. Throughout this paper we will use an RDF dataset describing companies, acquisitions between companies and employment history. This dataset is partially presented in Fig. 1. We consider this data to be annotated with the temporal domain, which intuitively means that the annotated triple is valid in dates contained in the annotation interval (the exact meaning of the annotations will be explained later). Also, the information in this example can be derived from Wikipedia and thus we can consider this data also annotated with the provenance domain (although not explicitly represented in the example). We follow the modelling of employment records proposed by DBpedia, for instance a list of employees of Google is available as members of the class http://dbpedia.org/class/yago/GoogleEmployees. For presentation purposes we use the shorter name googleEmp. We also introduce

---

[4] A predicate $p(\overline{\mathbf{x}}, y)$ is functional if for any $\overline{\mathbf{t}}$ there is *unique* t' for which $p(\overline{\mathbf{t}}, t')$ is true.

(youtubeEmp, sc, googleEmp): [2006, 2011]
(steveChen, type, youtubeEmp): [2005, 2011]
(chadHurley, type, youtubeEmp): [2005, 2010]
(jawedKarim, type, youtubeEmp): [2005, 2011]
(jawedKarim, type, paypalEmp): [2000, 2005]
(paypalEmp, sc, ebayEmp): [2002, 2011]
(chadHurley, type, paypalEmp): [2002, 2005]
(skypeEmp, sc, ebayEmp): [2005, 2011]
(SkypeCollab, sc, EbayCollab): [2005, 2009]
(SkypeCollab, sc, EbayCollab): [2009, 2011]
(niklasZennstrom, ceo, skype): [2003, 2007]
(ceo, sp, worksFor): [−∞, +∞]
(larryPage, worksFor, google): [1998, 2011]
(sergeyBrin, worksFor, google): [1998, 2011]

Fig. 1. Company acquisition dataset example.

SkypeCollab (resp. EbayCollab) to represent Skype's (resp. Ebay's) collaborators.

3.1. Syntax

Our approach is to extend triples with annotations, where an annotation is taken from a specific domain.[5]

An *annotated triple* is an expression $\tau : \lambda$, where τ is a triple and λ is an *annotation value* (defined below). An *annotated graph* is a finite set of annotated triples. The intended semantics of annotated triples depends of course on the meaning we associate to the annotation values. For instance, in a temporal setting [2]

(niklasZennstrom, ceoOf, skype) : [2003, 2007]

has intended meaning "Niklas was CEO of Skype during the period 2003–2007", while in the fuzzy setting [6] (skype, ownedBy, big-Company): 0.3 has intended meaning "Skype is owned by a big company to a degree not less than 0.3".

3.2. RDFS Annotation domains

To start with, let us consider a non-empty set L. Elements in L are our annotation values. For example, in a fuzzy setting, $L = [0, 1]$, while in a typical temporal setting, L may be time points or time intervals. In our annotation framework, an interpretation will map statements to elements of the annotation domain. Our semantics generalises the formulae in Remark 2.2 by using a well known algebraic structure.

We say that an *annotation domain* for RDFS is an idempotent, commutative semi-ring
$$D = \langle L, \oplus, \otimes, \bot, \top \rangle,$$
where $\oplus$ is $\top$-annihilating [24]. That is, for $\lambda, \lambda_i \in L$

1. $\oplus$ is idempotent, commutative, associative;
2. $\otimes$ is commutative and associative;
3. $\bot \oplus \lambda = \lambda, \top \otimes \lambda = \lambda, \bot \otimes \lambda = \bot$, and $\top \oplus \lambda = \top$;
4. $\otimes$ is distributive over $\oplus$, i.e., $\lambda_1 \otimes (\lambda_2 \oplus \lambda_3) = (\lambda_1 \otimes \lambda_2) \oplus (\lambda_1 \otimes \lambda_3)$.

It is well-known that there is a natural partial order on any idempotent semi-ring: an annotation domain $D = \langle L, \oplus, \otimes, \bot, \top \rangle$ induces a partial order $\preceq$ over L defined as:
$$\lambda_1 \preceq \lambda_2 \quad \text{if and only if} \quad \lambda_1 \oplus \lambda_2 = \lambda_2.$$

The order $\preceq$ is used to express redundant/entailed/subsumed information. For instance, for temporal intervals, an annotated triple $(s, p, o) : [2000, 2006]$ entails $(s, p, o) : [2003, 2004]$, as $[2003, 2004] \subseteq [2000, 2006]$ (here, $\subseteq$ plays the role of $\preceq$).

Remark 3.1. In previous work [12,13], an annotation domain was assumed to be a more specific structure, namely a residuated bounded lattice $D = \langle L, \preceq, \wedge, \vee, \otimes, \Rightarrow, \bot, \top \rangle$. That is:

1. $\langle L, \preceq, \wedge, \vee, \bot, \top \rangle$ is a bounded lattice, where $\bot$ and $\top$ are bottom and top elements, and $\wedge$ and $\vee$ are meet and join operators.
2. $\langle L, \otimes, \top \rangle$ is a commutative monoid.
3. $\Rightarrow$ is the so-called residuum of $\otimes$, i.e., for all $\lambda_1, \lambda_2, \lambda_3, \lambda_1 \otimes \lambda_3 \preceq \lambda_2$ if and only if $\lambda_3 \preceq (\lambda_1 \Rightarrow \lambda_2)$.

Note that any bounded residuated lattice satisfies the conditions of an annotation domain. In [24] it was shown that we may use a slightly weaker structure than residuated lattices for annotation domains.

Remark 3.2. Observe that $\langle L, \preceq, \oplus, \bot, \top \rangle$ is a bounded join semi-lattice.

Remark 3.3. Note that the domain $D_{01} = \langle \{0, 1\}, \max, \min, 0, 1 \rangle$ corresponds to the boolean case. In fact, in this case Annotated RDFS will turn out to be the same as classical RDFS.

Remark 3.4. We use $\oplus$ to combine information about the same statement. For instance, in temporal logic, from $\tau : [2000, 2006]$ and $\tau : [2003, 2008]$, we infer $\tau : [2000, 2008]$, as $[2000, 2008] = [2000, 2006] \cup [2003, 2008]$; here, $\cup$ plays the role of $\oplus$. In the fuzzy context, from $\tau : 0.7$ and $\tau : 0.6$, we infer $\tau : 0.7$, as $0.7 = \max(0.7, 0.6)$ (here, max *plays the role of* $\oplus$).

Remark 3.5. We use $\otimes$ to model the "conjunction" of information. In fact, a $\otimes$ is a generalisation of boolean conjunction to the many-valued case. In fact, $\otimes$ satisfies also that

1. $\otimes$ is bounded: i.e., $\lambda_1 \otimes \lambda_2 \preceq \lambda_1$.
2. $\otimes$ is $\preceq$-monotone, i.e., for $\lambda_1 \preceq \lambda_2, \lambda \otimes \lambda_1 \preceq \lambda \otimes \lambda_2$.

For instance, on interval-valued temporal logic, from $(a, sc, b):$ $[2000, 2006]$ and $(b, sc, c): [2003, 2008]$, we will infer $(a, sc, c):$ $[2003, 2006]$, as $[2003, 2006] = [2000, 2006] \cap [2003, 2008]$; here, $\cap$ plays the role of $\otimes$.[6] In the fuzzy context, one may chose any t-norm [30,31], e.g., product, and, thus, from $(a, sc, b): 0.7$ and $(b, sc, c): 0.6$, we will infer $(a, sc, c): 0.42$, as $0.42 = 0.7 \cdot 0.6$ (here, $\cdot$ plays the role of $\otimes$).

Remark 3.6. Observe that the distributivity condition is used to guarantee that, e.g., we obtain the same annotation $\lambda_1 \otimes (\lambda_2 \oplus \lambda_3)$ $= (\lambda_1 \otimes \lambda_2) \oplus (\lambda_1 \otimes \lambda_3)$ of the triple (a, sc, c) that can be inferred from triples $(a, sc, b) : \lambda_1, (b, sc, c) : \lambda_2$ and $(b, sc, c) : \lambda_3$.

Finally, note that, conceptually, in order to build an annotation domain, one has to:

1. determine the set of annotation values L (typically a countable set[7]), identify the top and bottom elements;
2. define suitable operations $\otimes$ and $\oplus$ that acts as "conjunction" and "disjunction" function, to support the intended inference over schema axioms, such as
    "from $(a, sc, b) : \lambda$ and $(b, sc, c) : \lambda'$ infer $(a, sc, c) : \lambda \otimes \lambda'$"
    and
    "from $\tau : \lambda$ and $\tau : \lambda'$ infer $\tau : \lambda \oplus \lambda'$".

---

[5] The readers familiar with the annotated logic programming framework [16], will notice the similarity of the approaches.

[6] As we will see, $\oplus$ and $\otimes$ may be more involved.

[7] Note that one may use XML decimals in [0,1] in place of real numbers for the fuzzy domain.

3.3. Semantics

Fix an annotation domain $D = \langle L, \oplus, \otimes, \bot, \top \rangle$. Informally, an interpretation $\mathcal{I}$ will assign to a triple τ an element of the annotation domain $\lambda \in L$. Formally, an *annotated interpretation* $\mathcal{I}$ over a vocabulary V is a tuple

$$\mathcal{I} = \langle \Delta_R, \Delta_P, \Delta_C, \Delta_L, P[\![\,\cdot\,]\!], C[\![\,\cdot\,]\!], \cdot^{\mathcal{I}} \rangle,$$

where $\Delta_R, \Delta_P, \Delta_C, \Delta_L$ are interpretation domains of $\mathcal{I}$ and $P[\![\,\cdot\,]\!], C[\![\,\cdot\,]\!], \cdot^{\mathcal{I}}$ are interpretation functions of $\mathcal{I}$.

They have to satisfy:

1. Δ_R is a nonempty finite set of resources, called the domain or universe of $\mathcal{I}$;
2. Δ_P is a finite set of property names (not necessarily disjoint from Δ_R);
3. $\Delta_C \subseteq \Delta_R$ is a distinguished subset of Δ_R identifying if a resource denotes a class of resources;
4. $\Delta_L \subseteq \Delta_R$, the set of literal values, Δ_L contains all plain literals in $\mathtt{L} \cap V$;
5. $P[\![\,\cdot\,]\!]$ maps each property name $p \in \Delta_P$ into a function $P[\![p]\!] : \Delta_R \times \Delta_R \to L$, i.e., assigns an annotation value to each pair of resources;
6. $C[\![\,\cdot\,]\!]$ maps each class $c \in \Delta_C$ into a function $C[\![c]\!] : \Delta_R \to L$, i.e., assigns an annotation value representing class membership in c to every resource;
7. $\cdot^{\mathcal{I}}$ maps each $t \in \mathtt{UL} \cap V$ into a value $t^{\mathcal{I}} \in \Delta_R \cup \Delta_P$ and such that $\cdot^{\mathcal{I}}$ is the identity for plain literals and assigns an element in Δ_R to each element in **L**.

An interpretation $\mathcal{I}$ is a *model* of an annotated ground graph G, denoted $\mathcal{I} \models G$, if and only if $\mathcal{I}$ is an interpretation over the vocabulary $\rho\mathrm{df} \cup universe(G)$ that satisfies the following conditions:

Simple:
1. $(s, p, o) : \lambda \in G$ implies $p^{\mathcal{I}} \in \Delta_P$ and $P[\![p^{\mathcal{I}}]\!](s^{\mathcal{I}}, o^{\mathcal{I}}) \succeq \lambda$.

Subproperty:
1. $P[\![\mathsf{sp}^{\mathcal{I}}]\!](p, q) \otimes P[\![\mathsf{sp}^{\mathcal{I}}]\!](q, r) \preceq P[\![\mathsf{sp}^{\mathcal{I}}]\!](p, r)$;
2. $P[\![p^{\mathcal{I}}]\!](x, y) \otimes P[\![\mathsf{sp}^{\mathcal{I}}]\!](p, q) \preceq P[\![q^{\mathcal{I}}]\!](x, y)$.

Subclass:
1. $P[\![\mathsf{sc}^{\mathcal{I}}]\!](c, d) \otimes P[\![\mathsf{sc}^{\mathcal{I}}]\!](d, e) \preceq P[\![\mathsf{sc}^{\mathcal{I}}]\!](c, e)$;
2. $C[\![c^{\mathcal{I}}]\!](x) \otimes P[\![\mathsf{sc}^{\mathcal{I}}]\!](c, d) \preceq P[\![d^{\mathcal{I}}]\!](x)$.

Typing I:
1. $C[\![c]\!](x) = P[\![\mathsf{type}^{\mathcal{I}}]\!](x, c)$;
2. $P[\![\mathsf{dom}^{\mathcal{I}}]\!](p, c) \otimes P[\![p]\!](x, y) \preceq C[\![c]\!](x)$;
3. $P[\![\mathsf{range}^{\mathcal{I}}]\!](p, c) \otimes P[\![p]\!](x, y) \preceq C[\![c]\!](y)$.

Typing II:
1. For each $\mathsf{e} \in \rho\mathrm{df}$, $\mathsf{e}^{\mathcal{I}} \in \Delta_P$;
2. $P[\![\mathsf{sp}^{\mathcal{I}}]\!](p, q)$ is defined only for $p, q \in \Delta_P$;
3. $C[\![\mathsf{sc}^{\mathcal{I}}]\!](c, d)$ is defined only for $c, d \in \Delta_C$;
4. $P[\![\mathsf{dom}^{\mathcal{I}}]\!](p, c)$ is defined only for $p \in \Delta_P$ and $c \in \Delta_C$;
5. $P[\![\mathsf{range}^{\mathcal{I}}]\!](p, c)$ is defined only for $p \in \Delta_P$ and $c \in \Delta_C$;
6. $P[\![\mathsf{type}^{\mathcal{I}}]\!](s, c)$ is defined only for $c \in \Delta_C$.

Intuitively, a triple $(s, p, o) : \lambda$ is satisfied by I if (s, o) belongs to the extension of p to a "wider" extent than λ. Note that the major differences from the classical setting relies on items 5 and 6.

We further note that the classical setting is as the case in which the annotation domain is D_{01} where $L = \{0, 1\}$.

Finally, entailment among annotated ground graphs G and H is as usual. Now, $G \models H$, where G and H may contain blank nodes, if and only if for any grounding G' of G there is a grounding H' of H such that $G' \models H'$.

Remark 3.7. Note that we always have that $G \models \tau :\bot$. Clearly, triples of the form $\tau :\bot$ are uninteresting and, thus, in the following we do not consider them as part of the language.

As for the crisp case, it can be shown that:

Proposition 3.1. *Any Annotated RDFS graph has a finite model.*

Proof 3.1. Let G be an annotated graph over domain d. Let $Lit = L \cap universe(G)$ be the set of literals present in G and $l_0 \in Lit$. We define the interpretation I over V as follows:

1. $\Delta_R = \Delta_P = \Delta_C = Lit = \Delta_L = Lit$;
2. $\forall x, y, p\ P[\![p]\!](x, y) \mapsto \top$;
3. $\forall x, c\ C[\![c]\!](x) \mapsto \top$;
4. (a) $\forall l \in L, l^{\mathcal{I}} = l$,
   (b) $\forall x \in V, l^{\mathcal{I}} = l_0$.

It is easy to see that I satisfies all the conditions of RDF-satisfiability and thus is a model of G.

Therefore, we do not have to care about consistency.

3.4. Examples of primitive domains

To demonstrate the power of our approach, we illustrate its application to some domains: fuzzy [6], temporal [2] and provenance.

3.4.1. The fuzzy domain

To model fuzzy RDFS [6] we may define the annotation domain as $D_{[0,1]} = \langle [0, 1], \max, \otimes, 0, 1 \rangle$ where $\otimes$ is any continuous t-norm on $[0, 1]$.

Example 3.1. Adapting our example of employment records to the fuzzy domain we can state the following: Skype collaborators are also Ebay collaborators to some degree since Ebay possesses 30% of Skype's shares, and also that Toivo is a part-time Skype collaborator:

```
(SkypeCollab, sc, EbayCollab) : 0.3
(toivo, type, SkypeCollab) : 0.5.
```

Then, e.g., under the product t-norm $\otimes$, we can infer the following triple:

```
(toivo, type, EbayCollab) : 0.15.
```

3.4.2. The temporal domain

Most of the semantic information on the Web deals with time in an implicit or explicit way. Social relation graphs, personal profiles, information about various entities continuously evolve and do not remain static. This dynamism can take various forms: certain information is only valid in a specific time interval (e.g., somebody's address), some data talks about events that took place at a specific time point in the past (e.g., beginning of a conference), some data describe eternal truth (e.g., tigers are mammals), or truth that is valid from a certain point of time onwards forever (e.g., Elvis is dead), or creation or change dates of online information items (e.g., the edit history of a wiki page). We believe that treating Web data in a time-sensitive way is one of the biggest steps towards turning the Semantic Web idea into reality.

3.4.2.1. Precise temporal information. For our representation of the temporal domain we aim at using non-discrete time as it is necessary to model temporal intervals with any precision. however, for presentation purposes we will show the dates as years only.

3.4.2.2. Modelling the temporal domain. To start with, *time points* are elements of the value space $\mathbb{Q} \cup \{-\infty, +\infty\}$. A *temporal interval* is a non-empty interval $[\alpha_1, \alpha_2]$, where α_i are time points. An empty interval is denoted as $\emptyset$. We define a partial order on intervals as $I_1 \leqslant I_2$ if and only if $I_1 \subseteq I_2$. The intuition here is that if a triple is true at time points in I_2 and $I_1 \leqslant I_2$ then, in particular, it is true at any time point in $I_1 \neq \emptyset$.

Now, apparently the set of intervals would be a candidate for L, which however is not the case. The reason is that, e.g., in order to represent the upper bound interval of $\tau : [1, 5]$ and $\tau : [8, 9]$ we rather need the union of intervals, denoted $\{[1, 5], [8, 9]\}$, meaning that a triple is true both in the former as well as in the latter interval. Now, we define L as (where $\bot = \{\emptyset\}$, $\top = \{[-\infty, +\infty]\}$)

$L = \{t | t$ is a finite set of disjoint temporal intervals$\} \cup \{\bot, \top\}$.

Therefore, a *temporal term* is an element $t \in L$, i.e., a set of pairwise disjoint time intervals. We allow to write $[\alpha]$ as a shorthand for $[\alpha, \alpha]$, $\tau : \alpha$ as a shorthand of $\tau : \{[\alpha]\}$ and $\tau : [\alpha, \alpha']$ as a shorthand of $\tau : \{[\alpha, \alpha']\}$. Furthermore, on L we define the following partial order:

$t_1 \preceq t_2$ if and only if $\forall I_1 \in t_1 \exists I_2 \in t_2$, such that $I_1 \leqslant I_2$.

Please note that $\preceq$ is the Hoare order on power sets [32], which is a pre-order. For the anti-symmetry property, assume that $t_1 \preceq t_2$ and $t_2 \preceq t_1$: so for $I_1 \in t_1$, there is $I_2 \in t_2$ for which there is $I_3 \in t_1$ such that $I_1 \subseteq I_2 \subseteq I_3$. But, t_1 is maximal and, thus, $I_1 = I_3 = I_2$. So, $t_1 = t_2$ and, thus, $\preceq$ is a partial order. Similarly as for time intervals, the intuition for $\preceq$ is that if a triple is true at time points in intervals in t_2 and $t_1 \preceq t_2$, then, in particular, it is true at any time point in intervals in t_1. Essentially, if $t_1 \preceq t_2$ then a temporal triple $\tau_2 : t_2$ is true to a larger "temporal extent" than the temporal triple $\tau_1 : t_1$. It can also be verified that $\langle L, \preceq, \bot, \top \rangle$ is a bounded lattice. Indeed, to what concerns us, the partial order $\preceq$ induces the following join ($\oplus$) operation on L. Intuitively, if a triple is true at t_1 and also at t_2 then it will be true also for time points specified by $t_1 \oplus t_2$ (a kind of union of time points). As an example, if $\tau : \{[2, 5], [8, 12]\}$ and $\tau : \{[4, 6], [9, 15]\}$ are true then we expect that this is the same as saying that $\tau : \{[2, 6], [8, 15]\}$ is true. The join operator will be defined in such way that $\{[2, 5], [8, 12]\} \oplus \{[4, 6], [9, 15]\} = \{[2, 6], [8, 15]\}$. Operationally, this means that $t_1 \oplus t_2$ will be obtained as follows: (i) take the union of the sets of intervals $t = t_1 \cup t_2$; and (ii) join overlapping intervals in t until no more overlapping intervals can be obtained. Formally,

$t_1 \oplus t_2 = \inf\{t \mid t \succeq t_i, i = 1, 2\}$.

It remains to define the meet $\otimes$ over sets of intervals. Intuitively, we would like to support inferences such as "from $(a, \text{sc}, b) : \{[2, 5], [8, 12]\}$ and $(b, \text{sc}, c) : \{[4, 6], [9, 15]\}$ infer $(a, \text{sc}, b) : \{[4, 5], [9, 12]\}$", where $\{[2, 5], [8, 12]\} \otimes \{[4, 6], [9, 15]\} = \{[4, 5], [9, 12]\}$. We get it by means of

$t_1 \otimes t_2 = \sup\{t \mid t \preceq t_i, i = 1, 2\}$.

Note that here the t-norm used for modelling "conjunction" coincides with the lattice meet operator.

Example 3.2. Using the data from our running example, we can infer that

$(\text{chadHurley}, \text{type}, \text{googleEmp}) : [2006, 2010]$,

where

$\{[2005, 2010]\} \otimes \{[2006, 2011]\} = \{[2006, 2010]\}$.

In [2] are described some further features such as a "Now" time point (which is just a defined time point in D_T) and anonymous time points, allowing to state that a triple is true at some point. Adding anonymous time points would require us to extend the lattice by appropriate operators, e.g. $[4, T] \oplus [T, 8] = [4, 8]$ (where T is an anonymous time point), etc.

3.4.3. Provenance domain

Identifying provenance of triples is regarded as an important issue for dealing with the heterogeneity of Web data, and several proposals have been made to model provenance [33–36]. Typically, provenance is identified by a URI, usually the URI of the document in which the triples are defined or possibly a URI identifying a named graph. However, provenance of inferred triples is an issue that have been little tackled in the literature [37,35]. We propose to address this issue by introducing an annotation domain for provenance.

The intuition behind our approach is similar to the one of [37] and [35] where provenance of an inferred triple is defined as the aggregation of provenances of documents that allow to infer that triple. For instance, if a document d_1 defines $(\text{youtubeEmp}, \text{sc}, \text{googleEmp}) : d_1$ and a second document d_2 defines $(\text{chadHurley}, \text{type}, \text{youtubeEmp}) : d_2$, then we can infer $(\text{chadHurley}, \text{type}, \text{googleEmp}) : d_1 \wedge d_2$.

Such a mechanism makes sense and would fit well as a meet operator, but these approaches do not address the join operation which should take place when identical triples are annotated differently. We improve this with the following formalisation.

3.4.3.1. Modelling the provenance domain. We start from a countably infinite set of *atomic provenances* $\mathbf{P}$ which, in practice, can be represented by URIs. We consider the propositional formulae made from symbols in $\mathbf{P}$ (atomic propositions), logical *or* ($\vee$) and logical *and* ($\wedge$), for which we have the standard entailment $\vDash$. A *provenance value* is an equivalent class for the logical equivalence relation, i.e., the set of annotation values is the quotient set of $\mathbf{P}$ by the logical equivalence. The order relation is $\vDash$, $\otimes$ and $\oplus$ are $\wedge$ and $\vee$, respectively. We set $\top$ to *true* and $\bot$ to *false*.

Example 3.3. Consider the following data:

$(\text{chadHurley}, \text{worksFor}, \text{youtube}) : chad$

$(\text{chadHurley}, \text{type}, \text{Person}) : chad$

$(\text{youtube}, \text{type}, \text{Company}) : chad$

$(\text{Person}, \text{sc}, \text{Agent}) : foaf$

$(\text{worksFor}, \text{dom}, \text{Person}) : workont$

$(\text{worksFor}, \text{range}, \text{Company}) : workont$.

We can deduce that chadHurley is an Agent in two different ways: using the first, fourth and fifth statement or using the second and fourth statement. So, it is possible to infer the following annotated triple:

$(\text{chadHurley}, \text{type}, \text{Agent}) : (chad \wedge foaf \wedge workont)$
$$\vee (chad \wedge foaf)$$

However, since $(chad \wedge foaf \wedge workont) \vee (chad \wedge foaf)$ is logically equivalent to $chad \wedge foaf$, the aggregated inference can be collapsed into:

$(\text{chadHurley}, \text{type}, \text{Agent}) : chad \wedge foaf$.

Intuitively, a URI denoting a provenance can also denote a RDF graph, either by using a named graph approach, or implicitly by getting a RDF document by dereferencing the URI. In this case, we can see the conjunction operation as a union of graphs and disjunction as an intersection of graphs.

3.4.3.2. Comparison with other approaches. [37] does not formalise the semantics and properties of his aggregation operation (simply denoted by ∧) nor the exact rules that should be applied to correctly and completely reason with provenance. Query answering is not tackled either.

The authors of [35] are providing more insight on the formalisation and actually detail the rules by reusing (tacitly) [26]. They also provide a formalisation of a simple query language. However, the semantics they define is based on a strong restriction of ρdf.[8]

As an example, they define the answers to the query (?*x*, type, ?-*y*, ?*c*) as the tuples (X, Y, C) such that there is a triple (X, type, Y, C) which can be inferred from only the application of rules (3a) and (3b). This means that a domain or range assertion would not provide additional answers to that type of query.

Provenance also relates to the named graphs formalism [23] where one can identify distinct graphs with a URI. The name can be seen as an atomic provenance annotation. However, named graphs do not provide operations to combine the provenances. Yet, the formalism could be used as a possible syntactic solution for representing annotated triples.

Finally, none of those papers discuss the possibility of universally true statements (the ⊤ provenance) or the statements from unknown provenance (⊥). They also do not consider mixing non-annotated triples with annotated ones as we do in Section 5.3.

3.5. Deductive system

An important feature of our framework is that we are able to provide a deductive system in the style of the one for classical RDFS. Moreover, *the schemata of the rules are the same for any annotation domain* (only support for the domain dependent ⊗ and ⊕ operations has to be provided) and, thus, are amenable to an easy implementation on top of existing systems. The rules are arranged in groups that capture the semantic conditions of models, A, B, C, X and Y are meta-variables representing elements in **UBL** and D, E represent elements in **UL**. The rule set contains two rules (1a) and (1b), that are the same as for the crisp case, while rules (2a)–(5b) are the annotated rules homologous to the crisp ones. Finally, rule (6) is specific to the annotated case.

Please note that rule (6) is destructive i.e., this rule removes the premises as the conclusion is inferred. We also assume that a rule is not applied if the consequence is of the form $\tau : \perp$ (see Remark 3.7). It can be shown that:

1. Simple:

    (a) $\dfrac{G}{G'}$ for a map $\mu : G' \to G$

    (b) $\dfrac{G}{G'}$ for $G' \subseteq G$

2. Subproperty:

    (a) $\dfrac{(A, \text{sp}, B) : \lambda_1, (B, \text{sp}, C) : \lambda_2}{(A, \text{sp}, C) : \lambda_1 \otimes \lambda_2}$

    (b) $\dfrac{(D, \text{sp}, E) : \lambda_1, (X, D, Y) : \lambda_2}{(X, E, Y) : \lambda_1 \otimes \lambda_2}$

3. Subclass:

    (a) $\dfrac{(A, \text{sc}, B) : \lambda_1, (B, \text{sc}, C) : \lambda_2}{(A, \text{sc}, C) : \lambda_1 \otimes \lambda_2}$

    (b) $\dfrac{(A, \text{sc}, B) : \lambda_1, (X, \text{type}, A) : \lambda_2}{(X, \text{type}, B) : \lambda_1 \otimes \lambda_2}$

4. Typing:

    (a) $\dfrac{(D, \text{dom}, B) : \lambda_1, (X, D, Y) : \lambda_2}{(X, \text{type}, B) : \lambda_1 \otimes \lambda_2}$

    (b) $\dfrac{(D, \text{range}, B) : \lambda_1, (X, D, Y) : \lambda_2}{(Y, \text{type}, B) : \lambda_1 \otimes \lambda_2}$

5. Implicit Typing:

    (a) $\dfrac{(A, \text{dom}, B) : \lambda_1, (D, \text{sp}, A) : \lambda_2, (X, D, Y) : \lambda_3}{(X, \text{type}, B) : \lambda_1 \otimes \lambda_2 \otimes \lambda_3}$

    (b) $\dfrac{(A, \text{range}, B) : \lambda_1, (D, \text{sp}, A) : \lambda_2, (X, D, Y) : \lambda_3}{(Y, \text{type}, B) : \lambda_1 \otimes \lambda_2 \otimes \lambda_3}$

6. Generalisation:

    $$\dfrac{(X, A, Y) : \lambda_1, (X, A, Y) : \lambda_2}{(X, A, Y) : \lambda_1 \oplus \lambda_2}$$

Proposition 3.2 (Soundness and completeness). *For an annotated graph, the proof system $\vdash$ is sound and complete for $\models$, that is, (1) if $G \vdash \tau : \lambda$ then $G \models \tau : \lambda$ and (2) if $G \models \tau : \lambda$ then there is $\lambda' \succeq \lambda$ with $G \vdash \tau : \lambda'$.*

We point out that rules 2–5 can be represented concisely using the following inference rule:

(AG) $\dfrac{\tau_1 : \lambda_1, \ldots, \tau_n : \lambda_n, \{\tau_1, \ldots \tau_n\} \vdash_{\text{RDFS}} \tau}{\tau : \otimes_i \lambda_i}$.

Essentially, this rule says that if a classical RDFS triple τ can be inferred by applying a classical RDFS inference rule to triples $\tau_1, \ldots \tau_n$ (denoted $\{\tau_1, \ldots, \tau_n\} \vdash_{\text{RDFS}} \tau$), then the annotation term of τ will be $\otimes_i \lambda_i$, where λ_i is the annotation of triple τ_i. It follows immediately that, using rule (AG), in addition to rules (1) and (6) from the deductive system above, it is easy to extend these rules to cover complete RDFS.

Finally, like for the classical case, the *closure* is defined as $cl(G) = \{\tau : \lambda \mid G \vdash^* \tau : \lambda\}$, where $\vdash^*$ is as $\vdash$ without rule (1a). Note again that the size of the closure of G is polynomial in $|G|$ and can be computed in polynomial time, provided that the computational complexity of operations ⊗ and ⊕ are polynomially bounded (from a computational complexity point of view, it is as for the classical case, plus the cost of the operations ⊗ and ⊕ in L). Eventually, similar propositions as Propositions 2.2 and 2.3 hold.

Example 3.4. As an example, consider the following triples from Fig. 1:

(youtubeEmp, sc, googleEmp) : $[2006, 2011]$
(chadHurley, worksFor, youtubeEmp) : $[2005, 2010]$

we infer the following triple:

(chadHurley, type, googleEmp) : $[2006, 2010]$

3.6. Query answering

Informally, queries are as for the classical case where triples are replaced with annotated triples in which *annotation variables*

---

[8] Remember that ρdf is already a restriction of RDFS.

(taken from an appropriate alphabet and denoted Λ) may occur. We allow built-in triples of the form (s,p,o), where p is a built-in predicate taken from a reserved vocabulary and having a *fixed interpretation* on the annotation domain d, such as $(\lambda,\preceq,l)$ stating that the value of lambda has to be $\preceq$ than the value $l \in L$. We generalise the built-ins to any n-ary predicate p, where p's arguments may be annotation variables, ρdf variables, domain values of d, values from **UL**, and p has a fixed interpretation. We will assume that the evaluation of the predicate can be decided in finite time. As for the crisp case, for convenience, we write "functional predicates" as *assignments* of the form $x := f(\overline{z})$ and assume that the function $f(\overline{z})$ is safe. We also assume that a non functional built-in predicate $p(\overline{z})$ should be safe as well.

For instance, informally for a given time interval $[t_1, t_2]$, we may define $x := length([t_1, t_2])$ as true if and only if the value of x is $t_2 - t_1$.

Example 3.5. Considering our dataset from Fig. 1 as input and the query asking for people that work for Google between 2002 and 2011 and the temporal term at which this was true:

$$q(x, \Lambda) \leftarrow (x, \mathtt{worksFor}, \mathtt{google}) : \Lambda',$$
$$\Lambda := (\Lambda' \wedge [2002, 2011]),$$

will get the following answers:

$\langle \mathtt{steveChen}, [2006, 2011] \rangle$
$\langle \mathtt{chadHurley}, [2006, 2010] \rangle$
$\langle \mathtt{jawedKarim}, [2006, 2011] \rangle$
$\langle \mathtt{larryPage}, [2002, 2011] \rangle$
$\langle \mathtt{sergeyBrin}, [2002, 2011] \rangle$.

Formally, an *annotated query* is of the form

$$q(\overline{x}, \overline{\Lambda}) \leftarrow \exists \overline{y} \exists \Lambda'. \varphi(\overline{x}, \overline{\Lambda}, \overline{y}, \overline{\Lambda}')$$

in which $\varphi(\overline{x}, \overline{\Lambda}, \overline{y}, \overline{\Lambda}')$ is a conjunction (as for the crisp case, we use "," as conjunction symbol) of annotated triples and built-in predicates, $\overline{x}$ and $\overline{\Lambda}$ are the distinguished variables, $\overline{y}$ and $\overline{\Lambda}'$ are the vectors of *non-distinguished variables* (existential quantified variables), and $\overline{x}, \overline{\Lambda}, \overline{y}$ and $\overline{\Lambda}'$ are pairwise disjoint. Variables in $\overline{\Lambda}$ and $\overline{\Lambda}'$ can only appear in annotations or built-in predicates. The query head contains at least one variable.

Given an annotated graph G, a query $q(\overline{x}, \overline{\Lambda}) \leftarrow \exists \overline{y} \exists \Lambda'. \varphi(\overline{x}, \overline{\Lambda}, \overline{y}, \overline{\Lambda}')$, a vector $\overline{t}$ of terms in *universe*(G) and a vector $\overline{\Lambda}$ of annotated terms in L, we say that $q(\overline{t}, \overline{\Lambda})$ is *entailed* by G, denoted $G \models q(\overline{t}, \overline{\Lambda})$, if and only if in any model I of G, there is a vector $\overline{t}'$ of terms in *universe*(G) and a vector $\overline{\Lambda}'$ of annotation values in L such that I is a model of $\varphi(\overline{t}, \overline{\Lambda}, \overline{t}', \overline{\Lambda}')$. If $G \models q(\overline{t}, \overline{\Lambda})$ then $\langle \overline{t}, \overline{\Lambda} \rangle$ is called an *answer* to q. The *answer set* of q w.r.t. G is ($\preceq$ extends to vectors point-wise)

$$ans(G, q) = \{ \langle \overline{t}, \overline{\lambda} \rangle \mid G \models q(\overline{t}, \overline{\lambda}), \overline{\lambda} \neq \perp \text{ and}$$
$$\text{for any } \overline{\lambda}' \neq \overline{\lambda} \text{ such that } G \models q(\overline{t}, \overline{\lambda}'), \overline{\lambda}' \preceq \overline{\lambda} \text{ holds} \}.$$

That is, for any tuple $\overline{t}$, the vector of annotation values $\overline{\lambda}$ is as large as possible. This is to avoid that redundant/subsumed answers occur in the answer set. The following can be shown:

Proposition 3.3. *Given a graph G, $\langle \overline{t}, \overline{\lambda} \rangle$ is an answer to q if and only if $\exists \overline{y} \exists \Lambda'. \varphi(\overline{t}, \overline{\lambda}, \overline{y}, \overline{\Lambda}')$ is true in the closure of G and λ is $\preceq$-maximal.*[9]

Therefore, we may devise a similar query answering method as for the crisp case by computing the closure, store it into a database and then using SQL queries with the appropriate support of built-in predicates and domain operations.

---

[9] $\exists \overline{y} \exists \Lambda'. \varphi(\overline{t}, \overline{\lambda}, \overline{y}, \overline{\lambda}')$ is true in the closure of G if and only if for some $\overline{t}', \overline{\lambda}'$ for all triples in $\varphi(\overline{t}, \overline{\lambda}, \overline{t}', \overline{\lambda}')$ there is a triple in $cl(G)$ that subsumes it and the built-in predicates are true, where an annotated triple $\tau : \lambda_1$ subsumes $\tau : \lambda_2$ if and only if $\lambda_2 \preceq \lambda_1$.

3.7. Queries with aggregates

As next, we extend the query language by allowing so-called aggregates to occur in a query. Essentially, aggregates may be like the usual SQL aggregate functions such as SUM, AVG, MAX, MIN. But, we have also domain specific aggregates such as $\oplus$ and $\otimes$.

The following examples present some queries that can be expressed with the use of built-in queries and aggregates.

Example 3.6. Using a built-in aggregate we can pose a query that, for each employee, retrieves his maximal time of employment for any company in the following way:

$$q(x, maxL) \quad \leftarrow \quad (x, \mathtt{worksFor}, y) : \lambda,$$
$$maxL := maxlength(\lambda).$$

Here, the *maxlength* built-in predicate returns, given a set of temporal intervals, the maximal interval in the set.

Example 3.7. Suppose we are looking for employees that work for some companies for a certain time period. We would like to know the average length of their employment. Then such a query will be expressed as

$$q(x, avgL) \quad \leftarrow \quad (x, \mathtt{worksFor}, y) : \lambda,$$
$$\mathsf{GroupedBy}(x),$$
$$avgL := \mathsf{AVG}[length(\lambda)]$$

Essentially, we group by the employee, compute for each employee the time he worked for a company by means of the built-in function *length*, and compute the average value for each group. That is, $g = \{\langle t, t_1 \rangle, \ldots, \langle t, t_n \rangle\}$ is a group of tuples with the same value t for employee x, and value t_i for y, where each length of employment for t_i is l_i (computed as $length(\cdot)$), then the value of *avgL* for the group G is $(\sum_i l_i)/n$.

Formally, let @ be an aggregate function with @ $\in$ {SUM, AVG, MAX, MIN, COUNT, $\oplus$, $\otimes$} then a query with aggregates is of the form

$$q(\overline{x}, \overline{\Lambda}, \alpha) \quad \leftarrow \quad \exists \overline{y} \exists \Lambda'. \varphi(\overline{x}, \overline{\Lambda}, \overline{y}, \overline{\Lambda}'),$$
$$\mathsf{GroupedBy}(\overline{w}),$$
$$\alpha := @[f(\overline{z})]$$

where $\overline{w}$ are variables in $\overline{x}, \overline{y}$ or $\overline{\Lambda}$ and each variable in $\overline{x}$ and $\overline{\Lambda}$ occurs in $\overline{w}$ and any variable in $\overline{z}$ occurs in $\overline{y}$ or $\overline{\Lambda}'$.

From a semantics point of view, we say that I is a model of (*satisfies*) $q(\overline{t}, \overline{\lambda}, a)$, denoted $I \models q(\overline{t}, \overline{\lambda}, a)$ if and only if

$a = @ [a_1, \ldots, a_k]$ where $g = \{\langle \overline{t}, \overline{\lambda}, \overline{t}'_1, \overline{\lambda}'_1 \rangle, \ldots, \langle \overline{t}, \overline{\lambda}, \overline{t}'_k, \overline{\lambda}'_k \rangle\}$,
is a group of k tuples with identical projection
on the variables in $\overline{w}, \varphi(\overline{t}, \overline{\lambda}, \overline{t}'_r, \overline{\lambda}'_r)$ is true in I
and $a_r = f(\overline{t})$ where $\overline{t}$ is the projection of $\langle \overline{t}'_r, \overline{\lambda}'_r \rangle$
on the variables $\overline{z}$.

Now, the notion of $G \models q(\overline{t}, \overline{\Lambda}, a)$ is as usual: any model of G is a model of $q(\overline{t}, \overline{\Lambda}, a)$.

Eventually, we further allow to order answers according to some ordering functions.

Example 3.8. Consider Example 3.7. We additionally would like to order the employee according to the average length of employment. Then such a query will be expressed as

$$q(x, avgL) \quad \leftarrow \quad (x, \mathsf{worksFor}, y) : \lambda,$$
$$\mathsf{GroupedBy}(x),$$
$$avgL := \mathsf{AVG}[length(\lambda)],$$
$$\mathsf{OrderBy}(avgL)$$

Formally, a query with ordering is of the form

$$q(\overline{x}, \overline{\Lambda}, z) \leftarrow \exists \overline{y} \exists \Lambda'.\varphi(\overline{x}, \overline{\Lambda}, \overline{y}, \overline{\Lambda}'), \quad OrderBy(z)$$

or, in case grouping is allowed as well, it is of the form

$$\begin{aligned}
q(\overline{x}, \overline{\Lambda}, z, \alpha) \quad \leftarrow \quad & \exists \overline{y} \exists \Lambda'.\varphi(\overline{x}, \overline{\Lambda}, \overline{y}, \overline{\Lambda}'), \\
& GroupedBy(\overline{w}), \\
& \alpha := [f(\overline{z})], \\
& OrderBy(z)
\end{aligned}$$

From a semantics point of view, the notion of $G \models q(\overline{t}, \overline{\Lambda}, z, a)$ is as before, but the notion of answer set has to be enforced with the fact that the answers are now ordered according to the assignment to the variable z. Of course, we require that the set of values over which z ranges can be ordered (like string, integers, reals). In case the variable z is an annotation variable, the order is induced by $\preceq$. In case, $\preceq$ is a partial order then we may use some linearisation method for posets, such as [38]. Finally, note that the additional of the SQL-like statement LIMIT(k) can be added straightforwardly.

4. AnQL: Annotated SPARQL

Our introduced query language so far allows for conjunctive queries. Languages like SQL and SPARQL allow to pose more complex queries including built-in predicates to filter solutions, advanced features such as negation or aggregates. In this section we will present an extension of the SPARQL [11] query language, called AnQL, that enables querying annotated graphs. We will begin by presenting some preliminaries on SPARQL.

4.1. SPARQL

SPARQL [11] is the W3C recommended query language for RDF. A *SPARQL query* is defined by a triple $Q = (P, G, V)$, where p is a *graph pattern* and the *dataset G* is an RDF graph and V is the *result form*. We will restrict ourselves to SELECT queries in this work so it is sufficient to consider the result form V as a list of variables.

Remark 4.1. Note that, for presentation purposes, we simplify the notion of datasets by excluding named graphs and thus GRAPH queries. Our definitions can be straightforwardly extended to named graphs and we refer the reader to the SPARQL W3C specification [11] for details.

We base our semantics of SPARQL on the semantics presented by Pérez et al. [39], extending the multiset semantics to lists, which are considered a multiset with "default" ordering. RDF triples, possibly with variables in subject, predicate or object positions, are called *triple patterns*. In the basic case, graph patterns are sets of triple patterns, also called *Basic Graph Patterns* (BGP). Let **U**, **B**, **L** be defined as before and let **V** denote a set of variables, disjoint from **UBL**. We further denote by *var(P)* the set of variables present in a graph pattern p.

Definition 4.1 (*Solution [11, Section 12.3.1]*). Given a graph G and a BGP p, a *solution* θ for p over G is a mapping over a subset V of *var(P)*, i.e., $\theta : V \rightarrow term(G)$ such that $G \models P\theta$ where $P\theta$ represents the triples obtained by replacing the variables in graph pattern p according to θ, and whre $G \models P\theta$ means that any triple in $P\theta$ is entailed by G. We call V the *domain* of θ, denoted by *dom(θ)*. For convenience, sometimes we will use the notation $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ to indicate that $\theta(x_i) = t_i$, i.e., variable x_i is assigned to term t_i.

Two mappings θ_1 and θ_2 are considered *compatible* if for all $x \in dom(\theta_1) \cap dom(\theta_2), \theta_1(x) = \theta_2(x)$. We call the evaluation of a BGP p over a graph G, denoted $[\![P]\!]_G$, the set of solutions.

Remark 4.2. Note that variables in the domain of θ play the role of distinguished variables in conjunctive queries and there are no non-distinguished variables.

The notion of solution for BGPs is the same as the notion of answers for conjunctive queries:

Proposition 4.1. *Given a graph G and a BGP p, then the solutions of p are the same as the answers of the query $q(var(P)) \leftarrow P$ (where var(P) is the vector of variables in p), i.e., $ans(G, q) = [\![P]\!]_G$.*

We present the syntax of SPARQL based on [39] and present *graph patterns* similarly. A *triple pattern* (s, p, o) is a graph pattern where $s, o \in$ **ULV** and $p \in$ **UV**.[10] Sets of triple patterns are called *Basic Graph Patterns* (BGP). A generic *graph pattern* is defined in a recursive manner: any BGP is a graph pattern; if p and P' are graph patterns, R is a filter expression (see [11]), then $(P$ AND $P'), (P$ OPTIONAL $P'), (P$ UNION $P'), (P$ FILTER $R)$ are graph patterns. As noted in Remark 4.1 we do not consider GRAPH patterns.

Evaluations of more complex patterns including FILTERs, OPTIONAL patterns, AND patterns, UNION patterns, etc. are defined by an algebra that is built on top of this *basic graph pattern matching* (see [11,39]).

Definition 4.2 (*SPARQL Relational Algebra*). Let Ω_1 and Ω_2 be sets of mappings:

$$\begin{aligned}
\Omega_1 \bowtie \Omega_2 &= \{\theta_1 \cup \theta_2 \mid \theta_1 \\
&\in \Omega_1, \theta_2 \in \Omega_2, \theta_1 \text{ and } \theta_2 \text{ compatible}\} \\
\Omega_1 \cup\!\!\!\cup \Omega_2 &= \{\theta \mid \theta \in \Omega_1 \text{ or } \theta \in \Omega_2\} \\
\Omega_1 - \Omega_2 &= \{\theta_1 \in \Omega_1 \mid \text{ forall } \theta_2 \in \Omega_2, \\
&\quad \theta_1 \text{ and } \theta_2 \text{ not compatible}\} \\
\Omega_1 \,\sqsupset\!\!\!\bowtie \Omega_2 &= (\Omega_1 \bowtie \Omega_2) \cup\!\!\!\cup (\Omega_1 - \Omega_2)
\end{aligned}$$

Definition 4.3 (*Evaluation [39, Definition 2.2]*). Let $\tau = (s, p, o)$ be a triple pattern, P, P_1, P_2 graph patterns and G an RDF graph, then the evaluation $[\![\cdot]\!]_G$ is recursively defined as follows:

$$\begin{aligned}
[\![t]\!]_G &= \{\theta \mid dom(\theta) = var(P) \text{ and } G \models \tau\theta\} \\
[\![P_1 \text{ AND } P_2]\!]_G &= [\![P_1]\!]_G \bowtie [\![P_2]\!]_G \\
[\![P_1 \text{ UNION } P_2]\!]_G &= [\![P_1]\!]_G \cup\!\!\!\cup [\![P_2]\!]_G \\
[\![P_1 \text{ OPTIONAL } P_2]\!]_G &= [\![P_1]\!]_G \,\sqsupset\!\!\!\bowtie [\![P_2]\!]_G \\
[\![P \text{ FILTER } R]\!]_G &= \{\theta \in [\![P]\!]_G \mid R\theta \text{ is true}\}
\end{aligned}$$

Let R be a FILTER[11] expression, $u, v \in$ V $\cup$ UBL. The valuation of R on a substitution θ, written $R\theta$, is *true* if:

(1) $R = $ BOUND(v) with $v \in dom(\theta)$;
(2) $R = $ isBLANK(v) with $v \in dom(\theta)$ and $\theta(v) \in$ B;
(3) $R = $ isIRI(v) with $v \in dom(\theta)$ and $\theta(v) \in$ U;
(4) $R = $ isLITERAL(v) with $v \in dom(\theta)$ and $\theta(v) \in$ L;
(5) $R = (u = v)$ with $u, v \in dom(\theta) \cup$ UBL $\wedge \ \theta(u) = \theta(v)$;
(6) $R = (\neg R_1)$ with $R_1\theta$ is false;
(7) $R = (R_1 \vee R_2)$ with $R_1\theta$ is true or $R_2\theta$ is true;
(8) $R = (R_1 \wedge R_2)$ with $R_1\theta$ is true and $R_2\theta$ 9m.

$R\theta$ yields an error (denoted ε), if:

(1) $R = $ isBLANK(v), $R = $ isIRI(v), or $R = $ isLITERAL(v) and $v \notin dom(\theta) \cup T$;

---

[10] We do not consider blank nodes in triple patterns since they can be considered as variables.

[11] For simplicity, we will omit from the presentation FILTERs such as comparison operators ('<', '>', '⩽', '⩾'), data type conversion and string functions and refer the reader to [11, Section 11.3] for details.

(2) $R = (u = v)$ with $u \notin dom(\theta) \cup T$ or $v \notin dom(\theta) \cup T$;
(3) $R = (\neg R_1)$ and $R_1\theta = \varepsilon$;
(4) $R = (R_1 \vee R_2)$ and $(R_1\theta \neq \top$ and $R_2\theta \neq \top)$ and $(R_1\theta = \varepsilon$ or $R_2\theta = \varepsilon)$;
(5) $R = (R1 \wedge R2)$ and $R_1\theta = \varepsilon$ or $R_2\theta = \varepsilon$.

Otherwise $R\theta$ is *false*.

In order to make the presented semantics compliant with the SPARQL specification [11], we need to introduce an extension to consider unsafe FILTERs (also presented in [40]):

Definition 4.4 (*OPTIONAL* with *FILTER* Evaluation). Let P_1, P_2 be graph patterns R a FILTER expression. A mapping θ is in $[\![P_1 \text{ OPTIONAL } (P_2 \text{ FILTER } R)]\!]_{DS}$ if and only if:

- $\theta = \theta_1 \cup \theta_2$, s.t. $\theta_1 \in [\![P_1]\!]_G, \theta_2 \in [\![P_2]\!]_G$ are compatible and $R\theta$ is true, or
- $\theta \in [\![P_1]\!]_G$ and $\forall \theta_2 \in [\![P_2]\!]_G, \theta$ and θ_2 are not compatible, or

- $\theta \in [\![P_1]\!]_G$ and $\forall \theta_2 \in [\![P_2]\!]_G$ s.t. θ and θ_2 are compatible, and $R\theta_3$ is false for $\theta_3 = \theta \cup \theta_2$.

4.2. AnQL

We are now ready to extend SPARQL for querying Annotated RDF. We call the novel query language AnQL. For the rest of this section we fix a specific annotation domain, $D = \langle L, \oplus, \otimes, \bot, \top \rangle$, as defined in Section 3.2.

4.2.1. Syntax

We take inspiration on the notion of conjunctive annotated queries discussed in Section 3.6. A *simple AnQL query* is defined – analogously to a SPARQL query – as a quadruple $Q = (P, G, V, A)$ with the differences that (1) G is an Annotated RDF graph; (2) we allow annotated graph patterns as presented in Definition 4.5 and (3) A is the set of annotation variables taken from an infinite set **A** (distinct from **V**). We further denote by $avar(P)$ the set of annotation variables present in a graph pattern p.

Definition 4.5 (*Annotated graph pattern*). Let λ be an annotation value from L or an annotation variable from **A**. We call λ an *annotation label*. Triple patterns in annotated AnQL are defined the same way as in SPARQL. For a triple pattern τ, we call $\tau : \lambda$ an *annotated triple pattern* and sets of annotated triple patterns are called *Basic Annotated Patterns* (BAP). A generic *annotated graph pattern* is defined in a recursive manner: any BAP is an annotated graph pattern; if p and P' are annotated graph patterns, R is a filter expression (see [11]), then $(P \text{ AND } P'), (P \text{ OPTIONAL } P'), (P \text{ UNION } P'), (P \text{ FILTER } R)$ are annotated graph patterns.

Example 4.1. Suppose we are looking for Ebay employees during some time period and that optionally owned a car during that period. This query can be posed as follows:

```
SELECT ?p ?l ?c WHERE {
    (?p type ebayEmp) : ?l
    OPTIONAL{(?p hasCar ?c) : ?l}
}
```

Assuming our example dataset from Fig. 1 extended with the following triples:

(toivo, type, paypalEmp) : [2000, 2009]
(toivo, hasCar, peugeot) : [1999, 2005]
(toivo, hasCar, renault) : [2005, 2010]

we will get the following answers:

$\theta_1 = \{?p/\text{toivo}, ?l/[2002, 2009]\}$
$\theta_2 = \{?p/\text{toivo}, ?l/[2002, 2005], ?c/\text{peugeot}\}$
$\theta_3 = \{?p/\text{toivo}, ?l/[2005, 2009], ?c/\text{renault}\}.$

The first answer corresponds to the answer in which the OPTIONAL pattern is not satisfied, so we get the annotation value [2002,2009] that corresponds to the time toivo is an Ebay employee. In the second and third answers, the OPTIONAL pattern is also matched and, in this case, the annotation value is restricted to the time when Toivo is employed by Paypal and has a car.

Note that – as we will see – this first query will return as a result for the annotation variable the periods where a car was owned.

Example 4.2. A slightly different query can be the employees of Ebay during some time period and optionally owned a car at some point during their stay. This query – which will rather return the time periods of employment – can be written as follows:

```
SELECT ?p ?l ?c WHERE {
    (?p type ebayEmp) : ?l
    OPTIONAL {(?p hasCar ?c) : ?l2
    FILTER (?l2 ⪯ ?l)}
}
```

Using the input data from Example 4.1, we obtain the following answers:

$\theta_1 = \{?p/\text{toivo}, ?l/[2002, 2009]\}$
$\theta_2 = \{?p/\text{toivo}, ?l/[2002, 2009], ?c/\text{renault}\}$

In this example the FILTER behaves as in SPARQL by removing from the answer set the mappings that do not make the FILTER expression true.

This query also exposes the issue of unsafe filters, noted in [40] and we presented the semantics to deal with this issue in Definition 4.4.

4.2.2. Semantics

We are thus ready to define the semantics of AnQL queries by extending the notion of SPARQL BGP matching. As for the SPARQL query language, we are going to define the notion of solutions for BAP as the equivalent notion of answers set of annotated conjunctive queries. Just as matching BGPs against RDF graphs is at the core of SPARQL semantics, matching BAPs against Annotated RDF graphs is the heart of the evaluation semantics of AnQL.

We extend the notion of *substitution* to include a substitution of annotation variables in which we do not allow any assignment of an annotation variable to $\bot$ (of the domain d). An annotation value of $\bot$, although it is a valid answer for any triple, does not provide any additional information and thus is of minor interest. Furthermore this would contribute to increasing the number of answers unnecessarily.

Definition 4.6 (*BAP evaluation*). Let p be a BAP and G an Annotated RDF graph. We define *evaluation* $[\![P]\!]_G$ as the list of substitutions that are *solutions* of p, i.e., $[\![P]\!]_G = \{\theta \mid G \models \theta(P)\}$, and where $G \models \theta(P)$ means that any annotated triple in $\theta(P)$ is entailed by G.

As for SPARQL, we have:

Proposition 4.2. *Given an annotated graph G and a BAP p, the solutions of p are the same as the answers of the annotated query $q(var(P)) \leftarrow P$ (where var(P) is the vector of variables in p), i.e., $ans(G, q) = [\![P]\!]_G$.*

For the extension of the SPARQL relational algebra to the annotated case we introduce – inspired by the definitions in [39] – definitions of compatibility and union of substitutions:

Definition 4.7 ($\otimes$-*compatibility*). Two substitutions θ_1 and θ_2 are $\otimes$-*compatible* if and only if (i) θ_1 and θ_2 are compatible for all the non-annotation variables, i.e., $\theta_1(x) = \theta_2(x)$ for any non-annotation variable $x \in dom(\theta_1) \cap dom(\theta_2)$; and (ii) $\theta_1(\lambda) \otimes \theta_2(\lambda) \neq \bot$ for any annotation variable $\lambda \in dom(\theta_1) \cap dom(\theta_2)$.

Definition 4.8 ($\otimes$-*union of substitutions*). Given two $\otimes$-compatible substitutions θ_1 and θ_2, the $\otimes$-*union* of θ_1 and θ_2, denoted $\theta_1 \otimes \theta_2$, is as $\theta_1 \cup \theta_2$, with the exception that any annotation variable $\lambda \in dom(\theta_1) \cap dom(\theta_2)$ is mapped to $\theta_1(\lambda) \otimes \theta_2(\lambda)$.

We now present the notion of evaluation for generic AnQL graph patterns. This consists of an extension of Definition 4.3:

Definition 4.9 (*Evaluation, extends [39, Definition 2]*). Let p be a BAP, P_1, P_2 annotated graph patterns, G an annotated graph and R a filter expression, then the evaluation $[\![\cdot]\!]_G$, i.e., set of *answers*,[12] is recursively defined as:

$[\![P]\!]_G = \{\theta \mid dom(\theta) = var(P) \text{ and } G \models \theta(P)\}$,
$[\![P_1 \text{ AND } P_2]\!]_G = \{\theta_1 \otimes \theta_2 \mid \theta_1 \in [\![P_1]\!]_G, \theta_2 \in [\![P_2]\!]_G, \theta_1$ and $\theta_2 \otimes -compatible\}$,
$[\![P_1 \text{ UNION } P_2]\!]_G = [\![P_1]\!]_G \cup [\![P_2]\!]_G$,
$[\![P_1 \text{ FILTER } R]\!]_G = \{\theta \mid \theta \in [\![P_1]\!]_G \text{ and } R\theta \text{ is true}\}$,
$[\![P_1 \text{ OPTIONAL } P_2[R]]\!]_G = \{\theta \mid \text{ and } \theta \text{ meets one of the following conditions:}$

1. $\theta = \theta_1 \otimes \theta_2$ if $\theta_1 \in [\![P_1]\!]_G, \theta_2 \in [\![P_2]\!]_G, \theta_1$ and $\theta_2 \otimes$-compatible, and $R\theta$ is true;
2. $\theta = \theta_1 \in [\![P_1]\!]_G$ and $\forall \theta_2 \in [\![P_2]\!]_G$ such that θ_1 and $\theta_2 \otimes$-compatible, $R(\theta_1 \otimes \theta_2)$ is true, and for all annotation variables $\lambda \in dom(\theta_1) \cap dom(\theta_2), \theta_2(\lambda) \prec \theta_1(\lambda)$;
3. $\theta = \theta_1 \in [\![P_1]\!]_G$ and $\forall \theta_2 \in [\![P_2]\!]_G$ such that θ_1 and $\theta_2 \otimes$-compatible, $R(\theta_1 \otimes \theta_2)$ is false.

Let R be a FILTER expression and $x, y \in \text{A} \cup \text{L}$, in addition to the FILTER expressions presented in Definition 4.3 we further allow the expressions presented next. The valuation of R on a substitution θ, denoted $R\theta$ is true if:[13]

(9) $R = (x \preceq y)$ with $x, y \in dom(\theta) \cup \text{L} \wedge \theta(x) \preceq \theta(y)$;

(10) $R = p(\overline{z})$ with $p(\overline{z})\theta = \text{true}$ if and only if $p(\theta(\overline{z})) = \text{true}$, where p is a built-in predicate.
   Otherwise $R\theta$ is false.

In the FILTER expressions above, a built-in predicate p is any n-ary predicate p, where p's arguments may be variables (annotation and non-annotation ones), domain values of d, values from **UL**, p has a fixed interpretation and we assume that the evaluation of the predicate can be decided in finite time. Annotation domains may define their own built-in predicates that range over annotation values as in the following query:

Example 4.3. Consider our example dataset from Fig. 1 and that we want to know where `chadHurley` was working before 2005. This query can be expressed in the following way:

```
SELECT ?city WHERE {
    (chadHurley worksFor ?comp) : ?l
    FILTER(before(?l, [2005]))
}
```

Remark 4.3. For practical convenience, we retain in $[\![\cdot]\!]_G$ only "domain maximal answers". That is, let us define $\theta' \preceq \theta$ if and only if (i) $\theta' \neq \theta$; (ii) $dom(\theta) = dom(\theta')$; (iii) $\theta(x) = \theta'(x)$ for any non-annotation variable x; and (iv) $\theta'(\lambda) \preceq \theta(\lambda)$ for any annotation variable λ. Then, for any $\theta \in [\![P]\!]_G$ we remove any $\theta' \in [\![P]\!]_G$ such that $\theta' \preceq \theta$.

Remark 4.4. Please note that the cases for the evaluation of the OPTIONAL are compliant with the SPARQL specification [11], covering the notion of unsafe FILTERs as presented in [40]. However, there are some peculiarities inherent to the annotated case. More specifically case 2 introduces the side effect that annotation variables that are compatible between the mappings may have different values in the answer depending if the OPTIONAL is matched or not. This is the behaviour demonstrated in Example 4.1.

The following proposition shows that we have a conservative extension of SPARQL.

Proposition 4.3. *Let $Q = (P, G, V)$ be a SPARQL query over an RDF graph G. Let G' be obtained from G by annotating triples with $\top$. Then $[\![P]\!]_G$ under SPARQL semantics is in one-to-one correspondence to $[\![P]\!]_{G'}$ under AnQL semantics such that for any $\theta \in [\![P]\!]_G$ there is a $\theta' \in [\![P]\!]_{G'}$ with θ and θ' coinciding on var(P).*

4.2.3. Further Extensions of AnQL

In this section we will present extensions of Definition 4.9 to include variable assignments, aggregates and solution modifiers. These are extensions similar to the ones presented in Section 3.7.

Definition 4.10. Let p be an annotated graph pattern and G an annotated graph, the evaluation of an ASSIGN statement is defined as:

$$[\![P \text{ ASSIGN } f(\overline{z}) \text{ AS } z]\!]_G = \{\theta \mid \theta_1 \in [\![P]\!]_G,$$
$$\theta = \theta_1[z/f(\theta_1(\overline{z}))]\}$$

where

$$\theta[z/t] = \begin{cases} \theta \cup \{z/t\} & \text{if } z \notin dom(\theta) \\ (\theta \setminus \{z/t'\}) \cup \{z/t\} & \text{otherwise}. \end{cases}$$

Essentially, we assign to the variable z the value $f(\theta_1(\overline{z}))$, which is the evaluation of the function $f(\overline{z})$ with respect to a substitution $\theta_1 \in [\![P]\!]_G$.

Example 4.4. Using a built-in function we can retrieve for each employee the length of employment for any company:

```
SELECT ?x ?y ?z WHERE {
    (?x worksFor ?y):?l
    ASSIGN length(?l) AS ?z
}
```

Here, the *length* built-in predicate returns, given a set of temporal intervals, the overall total length of the intervals.

---

[12] Strictly speaking, we consider *sequences* of answers – note that SPARQL allows duplicates and imposes an order on solutions, cf. Section 4.2.3 below for more discussion – but we stick with set *notation* representation here for illustration. Whenever we mean "real" sets where duplicates are removed we write $\{\dots\}_{\text{DISTINCT}}$.
[13] We consider a simple evaluation of filter expressions where the "error" result is ignored, see [11, Section 11.3] for details.

Remark 4.5. Note that this definition is more general than "SELECT *expr* AS *?var*" project expressions in current SPARQL 1.1 [41] due to not requiring that the assigned variable be unbound.

We introduce the ORDERBY clause where the evaluation of a $[\![P \text{ ORDERBY } ?x]\!]_G$ statement is defined as the ordering of the solutions – for any $\theta \in [\![P]\!]_G$ – according to the values of $\theta(?x)$. Ordering for non-annotation variables follows the rules in [11, Section 9.1].

Similarly to ordering in the query answering setting, we require that the set of values over which x ranges can be ordered and some linearisation method for posets may be applied if necessary, such as [38]. We can further extend the evaluation of AnQL queries with aggregate functions

$$@ \in \{\text{SUM, AVG, MAX, MIN, COUNT}, \oplus, \otimes\}$$

as follows:

Definition 4.11. The evaluation of a GROUPBY statement is defined as[14]:

$$[\![P \text{ GROUPBY}(\overline{w}) \; \overline{@f}(\overline{z}) \; AS\overline{\alpha}]\!]_G = \{\theta \mid \theta_1 \text{in } [\![P]\!]_G,$$
$$\theta = \theta_1|_{\overline{w}}[\alpha_i/@_i f_i(\theta_i(\overline{z}_i))]\}_{\text{DISTINCT}}$$

where the variables $\alpha_i \notin var(P), \overline{z}_i \in var(P)$ and none of the GROUPBY variables $\overline{w}$ are included in the aggregation function variables $\overline{z}_i$. Here, we denote by $\theta|_{\overline{w}}$ the restriction of variables in θ to variables in $\overline{w}$. Using this notation, we can also straightforwardly introduce projection, i.e., sub-SELECTs as an algebraic operator in the language covering another new feature of SPARQL 1.1:

$$[\![\text{SELECT } \overline{V} \; \{P\}]\!]_G = \{\theta \mid \theta_1 \text{ in } [\![P]\!]_G, \theta = \theta_1|_{\overline{v}}\}.$$

Remark 4.6. Please note that the aggregator functions have a domain of definition and thus can only be applied to values of their respective domain. For example, SUM and AVG can only be used on numeric values, while MAX, MIN are applicable to any total order. Resolution of type mismatches for aggregates is currently being defined in SPARQL 1.1 [41] and we aim to follow those, as soon as the language is stable. The COUNT aggregator can be used for any finite set of values. The last two aggregation functions, namely $\oplus$ and $\otimes$, are defined by the annotation domain and thus can be used on any annotation variable.

Remark 4.7. Please note that, unlike the current SPARQL 1.1 syntax, assignment, solution modifiers (ORDER BY, LIMIT) and aggregation are stand-alone operators in our language and do not need to be tied to a sub-SELECT but can occur nested within any pattern. This may be viewed as syntactic sugar allowing for more concise writing than the current SPARQL 1.1 [41] draft.

Example 4.5. Suppose we want to know, for each employee, the average length of their employments with different employers. Then such a query will be expressed as:

```
SELECT ?x ?avgL WHERE {
    (?x worksFor ?y) : ?l
    GROUPBY(?x)
    AVG(length(?l)) AS ?avgL
}
```

Essentially, we group by the employee, compute for each employee the time he worked for a company by means of the built-in function *length*, and compute the average value for each group.

That is, if $g = \{\langle t, t_1 \rangle, \ldots, \langle t, t_n \rangle\}$ is a group of tuples with the same value t for employee x, and value t_i for y, where each length of employment for t_i is l_i (computed as $length(\cdot)$), then the value of *avgL* for the group G is $(\sum_i l_i)/n$.

Proposition 4.4. *Assuming the built-in predicates are computable in finite time, the answer set of any AnQL is finite and can also be computed in finite time.*

This proposition can be demonstrated by induction over all the constructs we allow in AnQL.

4.3. Constraints vs. filters

Please note that FILTERs do not act as *constraints* over the query. Given the data from our dataset example and for the following query:

```
SELECT ?l1 ?l2 WHERE {
    (?p type youtubeEmp) : ?l1.
    (steveChen type youtubeEmp) : ?l2
}
```

with an additional *constraint* that requires $?l1$ to be "before" $?l2$, we could expect the answer

$$\{?l1/[2005, 2010], ?l2/[2011, 2011]\}.$$

This answer matches the following triples of our dataset:

$$(\texttt{steveChen, type, youtubeEmp}) : [2005, 2011]$$
$$(\texttt{chadHurley, type, youtubeEmp}) : [2005, 2010]$$

and satisfies the proposed *constraint*. However, we require maximality of the annotation values in the answers, which in general, do not exist in presence of *constraints*. For this reason, we do not allow general *constraints*.

4.4. Union of annotations

The SPARQL UNION operator may also introduce some discussion when considering shared annotations between graph patterns. Take for example the following query:

```
SELECT ?l WHERE {
    {(chadHurley type youtubeEmp) : ?l1}
    UNION
    (chadHurley type paypalEmp) : ?l
}
```

and assume our dataset from Fig. 1 as input. Considering the temporal domain, the intuitive meaning of the query is "retrieve all time periods when chadHurley was an employee of Youtube or PayPal". In the case of UNION patterns the two instances of the variable $?l$ are treated as two different variables. If the intended query would rather require treating both instances of the variable $?l$ as the same, for instance to retrieve the time periods when chadHurley was an employee of either Youtube or PayPal but assuming we may not have information for one of the patterns, the query should rather look like:

```
SELECT ?l WHERE {
    {(chadHurley type youtubeEmp):?l1}
    UNION
    {(chadHurley type paypalEmp):?l2}
    ASSIGN ?l1 ∨ ?l2 as ?l
}
```

---

[14] In the expression, $\overline{@f}(\overline{z})$ *AS* $\overline{\alpha}$ is a concise representation of n aggregations of the form $@_i f_i(\overline{z}_i)$ *AS* α_i.
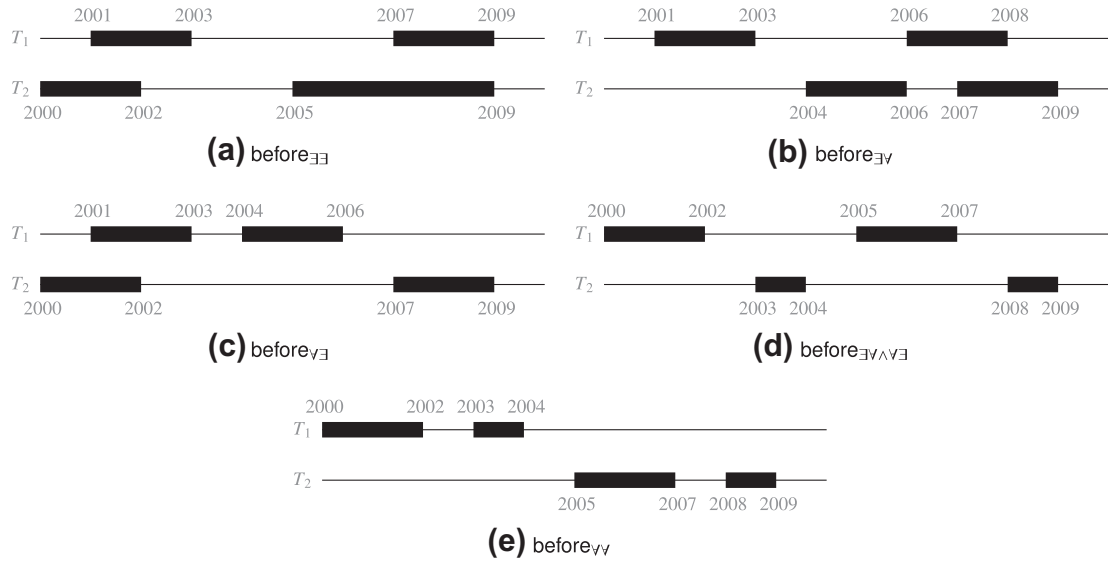
Fig. 2. Temporal relations.

where ∨ represents the domain specific built-in predicate for union of annotations.

5. On primitive domains and their combinations

In this section we discuss some practical issues related to (i) the representation of the temporal domain (Section 5.1); (ii) the combination of several domains into one compound domain (Section 5.2); (iii) the integration of differently annotated triples or non-annotated triples in the data or query (Section 5.3).

5.1. Temporal issues

Let us highlight some specific issues inherent to the temporal domain. Considering queries using Allen's temporal relations [42] (before, after, overlaps, etc.) as allowed in [4], we can pose queries like "find persons who were employees of PayPal before `toivo`". This query raises some ambiguity when considering that persons may have been employed by the same company at different disjoint intervals. We can model such situations – relying on sets of temporal intervals modelling the temporal domain. Consider our dataset triples from Fig. 1 extended with the following triple:

(`toivo`, type, `paypalEmp`) : {[1999, 2004], [2006, 2008]}.

Tappolet and Bernstein [4] consider this triple as two triples with disjoint intervals as annotations. For the following query in their language τSPARQL:

```
SELECT ?p WHERE {
    [?s1,?e1] ?p type youtubeEmp.
    [?s2,?e2] chadHurley type youtubeEmp.
    [?s1,?e1] time:intervalBefore [?s2,?e2]
}
```

We would get `chadHurley` as an answer although `toivo` was already working for PayPal when `chadHurley` started. This is one possible interpretation of "before" over a set of intervals. In AnQL we could add different domain specific built-in predicates, representing different interpretations of "before". For instance, we could define binary built-ins (i) beforeAny (?A1, ?A2) which is true if there exists *any* interval in annotation ?A1 before an interval

in ?A2, or, respectively, a different built-in beforeAll (?A1, ?A2) which is only true if *all* intervals in annotation ?A1 are before any interval in ?A2. Using the latter, an AnQL query would look as follows:

```
SELECT ?p WHERE {
    (?p type youtubeEmp):?l1.
    (toivo type youtubeEmp):?l2.
    FILTER(beforeAll(?l1,?l2))
}
```

This latter query gives no result, which might comply with people's understanding of "before" in some cases, while we also have the choice to adopt the behaviour of [4] by use of beforeAny instead.

More formally, if we consider an Allen relation **r** that holds between individual intervals, we can define a relation $\overline{r}$ over sets of intervals in five different ways:

Definition 5.1. Let T_1 and T_2 be two non-empty sets of disjoint intervals. We define the following relations:

- $\overline{r}_{\exists\exists} = \{\langle T_1, T_2\rangle \mid \exists t_1 \in T_1, \exists t_2 \in T_2 \text{ such that } \langle t_1, t_2\rangle \in r\}$;
- $\overline{r}_{\exists\forall} = \{\langle T_1, T_2\rangle \mid \exists t_1 \in T_1, \forall t_2 \in T_2 \text{ such that } \langle t_1, t_2\rangle \in r\}$;
- $\overline{r}_{\forall\exists} = \{\langle T_1, T_2\rangle \mid \forall t_1 \in T_1, \exists t_2 \in T_2 \text{ such that } \langle t_1, t_2\rangle \in r\}$;
- $\overline{r}_{\exists\forall\wedge\forall\exists} = \overline{r}_{\exists\forall} \cap \overline{r}_{\forall\exists}$;
- $\overline{r}_{\forall\forall} = \{\langle T_1, T_2\rangle \mid \forall t_1 \in T_1, \forall t_2 \in T_2 \text{ such that } \langle t_1, t_2\rangle \in r\}$.

These relations can be compared with each others according to set inclusion, as shown in Fig. 3. We illustrate them by the following examples, taking the Allen relation before:

Example 5.1. Fig. 2 is an example of time intervals that make each of the relations introduced in Definition 5.1 true for the *before* Allen relation.

It should be noticed that if one sticks to one choice of quantifier, the resulting set of relations does not form a proper relation algebra. Indeed, it is easy to see that, in the first 3 cases, the relations are not disjoint. For instance, two sets of intervals can be involved in both a $\overline{before}_{\exists\exists}$ and an $\overline{after}_{\exists\exists}$ relation. On the other hand, the last 4 cases are incomplete, that is, there are pairs of sets of intervals that cannot be related with any of the $\overline{r}_{\exists\forall}, \overline{r}_{\forall\exists}, \overline{r}_{\forall\forall}$ or $\overline{r}_{\exists\forall\wedge\forall\exists}$.
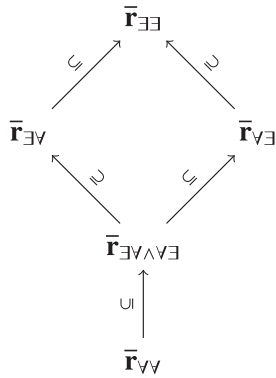
Fig. 3. Hierarchy of relations.

5.2. Extensions to multiple domains

Since annotations in our framework can range over different domains in different applications, one may be interested in combining several annotation domains such as annotating triples with a temporal term and a truth degree or degree of trust, etc. In [12], we proposed an approach for easily combining multiple domains, based on the pointwise extension of domain operators to a product of domains. Here, we criticise this approach and propose a revised approach that better fits the intuition.

5.2.1. Former approach and criticism

The approach described in [12] is the following. In general, assuming having domains $D_1, \ldots, D_n$, where $D_i = \langle L_i, \oplus_i, \otimes_i, \bot_i, \top_i \rangle$, we may build the domain $D = D_1 \times \cdots \times D_n = \langle L, \oplus, \otimes, \bot, \top \rangle$, where $L = L_1 \times \cdots \times L_n$, $\bot = \langle \bot_1, \ldots, \bot_n \rangle$, $\top = \langle \top_1, \ldots, \top_n \rangle$ and the meet and join operations $\otimes$ and $\oplus$ are extended pointwise to L, e.g., $\langle \lambda_1, \ldots, \lambda_n \rangle \otimes \langle \lambda'_1, \ldots, \lambda'_n \rangle' = \langle \lambda_1 \otimes \lambda'_1, \ldots, \lambda_n \otimes \lambda'_n \rangle$. For instance,

$$(\text{SkypeCollab}, \text{sc}, \text{EbayCollab}) : \langle [2009, 2011], 0.3 \rangle$$

may indicate that during 2009–2011, the collaborators of Skype were also considered collaborators of Ebay to degree 0.3 (here we combine a temporal domain and a fuzzy domain). The interesting point of our approach is that the rules of the deductive systems need not be changed, nor the query answering mechanism (except to provide the support to compute $\otimes$ and $\oplus$ accordingly).

The problem with this approach is that the annotations are dealt with independently from each others. As a result, e.g., the truth value 0.3 does not apply to the time range [2009, 2011]. This problem is made very apparent when one observes the unexpected consequences of our $\oplus$ operator on such a combination:

$$(\text{SkypeCollab}, \text{sc}, \text{EbayCollab}) : \langle [2005, 2009], 1 \rangle$$
$$(\text{SkypeCollab}, \text{sc}, \text{EbayCollab}) : \langle [2009, 2011], 0.3 \rangle$$

Applying the point-wise operation $\oplus$, this leads to the conclusion:

$$(\text{SkypeCollab}, \text{sc}, \text{EbayCollab}) : \langle [2005, 2011], 1 \rangle.$$

This defies the intuition that, between 2005 and 2009, Skype collaborators where also Ebay employees (collaborate to degree 1), but from 2009 to 2011 Skype collaborators were Ebay collaborators to the degree 0.3. The pointwise aggregation does not follow this intuition and levels up everything. In the example above, we would like to say that the fuzzy value itself has a duration, so that the temporal interval corresponds more to an annotation of a quadruple. Note that this problem is not specific to the combination of time and fuzziness. We observe a similar issue when combining provenance, for instance, with other domains:

$$(\text{skypeEmp}, \text{sc}, \text{ebayEmp}) : \langle [2005, 2009], \texttt{wikipedia} \rangle$$
$$(\text{skypeEmp}, \text{sc}, \text{ebayEmp}) : \langle [1958, 2012], \texttt{wrong} \rangle$$

Using a point-wise aggregation method, the result would be:

$$(\text{skypeEmp}, \text{sc}, \text{ebayEmp}) : \langle [1958, 2012],$$
$$\texttt{wikipedia} \vee \texttt{wrong} \rangle$$

which entails:

$$(\text{skypeEmp}, \text{sc}, \text{ebayEmp}) : \langle [1958, 2012], \texttt{wikipedia} \rangle.$$

Again, the problem is that provenance here does not define the provenance of the temporal annotation and the temporal annotation is not local to a certain provenance.

In order to match the intuition, we devise a systematic construction that defines a new compound domain out of two existing domains.

5.2.2. Improved formalisation

In this section, we propose a generic construction that builds an annotation domain by combining two predefined domains in a systematic way. To achieve this, we will assume the existence of two annotation domains $D_1 = \langle L_1, \oplus_1, \otimes_1, \bot_1, \top_1 \rangle$ and $D_2 = \langle L_2, \oplus_2, \otimes_2, \bot_2, \top_2 \rangle$ which will be instantiated in examples with the temporal domain for D_1 (abbreviated D_t) and either the fuzzy domain (D_f) or the provenance domain (D_p) for D_2. We denote the temporal and fuzzy combination *time + fuzzy*, and the temporal and provenance combination *time + provenance*.

5.2.2.1. Intuition and desired properties.
In our former approach, we remarked that some information is lost in the join operation. Considering time + fuzzy, we see that the join should represent temporary *changes* in the degree of truth of the triple. Yet, it is clear that representing such changes cannot be done with a simple pair (intervals, value). So, as a first extension of our previous naïve solution, we suggest using sets of pairs of primitive annotations, as exemplified below

$$(\text{SkypeCollab}, \text{sc}, \text{EbayCollab}) : \{\langle [2005, 2009], 1 \rangle,$$
$$\langle [2009, 2011], 0.3 \rangle\}$$

Starting from this, we devise an annotation domain that correctly matches the intuitive meaning of the compound annotations. The annotated triple above can be interpreted as follows: for each pair in the annotation, for each time point in the temporal value of the pair, the triple holds to at least the degree given by the fuzzy value of the pair. The time + provenance combination is interpreted analogously, except that the triple holds (at least) in the context given by the provenance value of the pair.

This interpretation of the compound annotations implies that multiple sets of pairs can convey the exact same information. For example, the following time + fuzzy annotated triples are equivalent:

$$(\text{SkypeCollab}, \text{sc}, \text{EbayCollab}) : \quad \{\langle [2005, 2009], 1 \rangle\}$$
$$(\text{SkypeCollab}, \text{sc}, \text{EbayCollab}) : \quad \{\langle [2005, 2009], 0.3 \rangle,$$
$$\langle [2005, 2009], 1 \rangle\}$$

From this observation, we postulate the following desired property:

Property 1. *For all* $x \in L_1, y, y' \in L_2$ *and for all ρdf triples τ, $\tau : \{\langle x, y \rangle, \langle x, y' \rangle\}$ is semantically equivalent to $\tau : \{\langle x, y \oplus_2 y' \rangle\}$.*

Consequently, it is always possible to assign a unique element $y \in L_2$ to a given element of L_1. Thus, an arbitrary set of pairs in $L_1 \times L_2$ is equivalently representable as a partial mapping from L_1 to L_2. Additionally, given a certain time interval, we can easily compute the maximum known degree to which a time + fuzzy annotated triple holds. For instance, with the annotation

$\{\langle[2005,2009],0.3\rangle, \langle[2008,2011],1\rangle\}$, we can assign the degree 1 to any subset of [2008,2011]; the degree 0.3 to any subset of [2005,2009] which is not contained in [2008,2009]; the degree 0 to any other temporal value.

This remark justifies that we can consider a compound annotation A as a *total* function from L_1 and L_2. From now on, whenever A is a finite set of pairs, we will denote by $\overline{A}$ the function that maps elements of L_1 to an element of L_2 that, informally, minimally satisfies the constraints imposed by the pairs in A. This is formalised below. For instance, if $A = \{\langle[2005,2009],0.3\rangle, \langle[2008,2011],1\rangle\}$, then:

$$\overline{A}(x) = \begin{cases} 1 & \text{if } x \subseteq [2008,2011] \\ 0.3 & \text{if } x \subseteq [2005,2009] \quad \text{and } x \nsubseteq [2008,2009] \\ 0 & \text{otherwise} \end{cases}$$

Whereas in this example, for the time + fuzzy domain the value of $\overline{A}$ for a particular interval seems to follow quite intuitively, let us next turn to the less obvious combination of time + provenance. Here, we postulate that the following triples

$\tau : \{\quad \langle[2005,2009], \texttt{wikipedia}\rangle,$
$\qquad \langle[2008,2011], \texttt{wrong}\rangle\}$

$\tau : \{\quad \langle[2005,2007], \texttt{wikipedia}\rangle,$
$\qquad \langle[2007,2009], \texttt{wikipedia}\rangle,$
$\qquad \langle[2008,2011], \texttt{wrong}\rangle\}$

$\tau : \{\quad \langle[2005,2008], \texttt{wikipedia}\rangle,$
$\qquad \langle[2008,2009], \texttt{wikipedia} \vee \texttt{wrong}\rangle,$
$\qquad \langle[2009,2011], \texttt{wrong}\rangle\},$

represent in fact equivalent annotations. Let us check the intuition behind this on a particular interval, [2005,2009], which for the first triple has unambiguously associated the provenance value `wikipedia`. Considering the second annotated triple, we observe that the provenance `wikipedia` can likewise be associated with the interval [2005,2009] because this provenance is associated with two intervals that – when joined – cover the time span [2005,2009]. In the case of the last annotated triple, the provenance `wikipedia∨wrong` means that the triple holds in `wikipedia` as well as in `wrong` (notice that $x \vee y$ means that the assertion holds in x and in y likewise, see Section 3.4.3 for details). Intuitively, we expect for the last triple that the provenance associated with the joined interval [2005,2009] is obtained from applying the meet operator over the respective provenance annotations `wikipedia` (for the partial interval [2005,2008]) and `wikipedia∨wrong` (for the partial interval [2008,2009]), i.e., $(\texttt{wikipedia}\vee\texttt{wrong}) \wedge \texttt{wikipedia}$ which – again – is equivalent to `wikipedia` in the provenance domain. Besides, considering now the interval [2005,2011], the triple is true in either `wikipedia.org` or `wrong`, which is modelled as $(\texttt{wikipedia}\wedge\texttt{wrong})$ in the provenance domain. Let us cast this intuition into another property we want to ensure on the function $\overline{A}$:

Property 2. *Given a set of annotation pairs A, for all $x_0 \in L_1$ whenever $\exists J \subseteq A$ with $x_0 \preceq_1 \bigoplus_{1\,\langle x,y \rangle \in J} x$, we have $\overline{A}(x_0) \succeq_2 \bigotimes_{2\,\langle x,y \rangle \in J} y$.*

Our goal in what follows is to characterise the set of functions associated with a finite set of pairs, that is $\{\overline{A} \mid A \subseteq L_1 \times L_2\}$, in a manner such that Property 1 and Property 2 are satisfied.

5.2.2.2. Formalisation. As mentioned before, a compound annotation can be seen as a function that maps values of the first domain to values of the second domain. In order to get the desired properties above established, we restrict this function to a particular type of functions that we call *quasihomomorphism* because it closely resembles a semiring homomorphism.

Definition 5.2 (*Quasihomomorphism*). Let f be a function from $D_1 = \langle L_1, \oplus_1, \otimes_1, \bot_1, \top_1 \rangle$ to $D_2 = \langle L_2, \oplus_2, \otimes_2, \bot_2, \top_2 \rangle$. f is a *quasihomomorphism* of domains if and only if for all $x,y \in L_1$: (i) $f(x \oplus_1 y) \succeq_2 f(x) \otimes_2 f(y)$ and (ii) $f(x \otimes_1 y) \succeq_2 f(x) \oplus_2 f(y)$.

We now use quasihomomorphisms to define – on an abstract level – a compound domain of annotations.

Definition 5.3 (*Compound annotation domain*). Given two primitive annotation domains D_1 and D_2, the *compound annotation domain* of D_1 and D_2 is the tuple $\langle L_{12}, \oplus_{12}, \otimes_{12}, \bot_{12}, \top_{12} \rangle$ defined as follows:

- L_{12} is the set of quasihomomorphisms from D_1 to D_2;
- $\bot_{12}$ is the function defined such that for all $x \in L_1, \bot_{12}(x) = \bot_2$;

- $\top_{12}$ is the function defined such that for all $x \in L_1, \top_{12}(x) = \top_2$;
- for all $\lambda, \mu \in L_{12}$, for all $x \in L_1, (\lambda \oplus_{12} \mu)(x) = \lambda(x) \oplus_2 \mu(x)$;
- for all $\lambda, \mu \in L_{12}$, for all $x \in L_1, (\lambda \otimes_{12} \mu)(x) = \lambda(x) \otimes_2 \mu(x)$;

This definition yields again a valid RDF annotation domain, as stated in the following proposition:

Proposition 5.1. $\langle L_{12}, \oplus_{12}, \otimes_{12}, \bot_{12}, \top_{12} \rangle$ *is an idempotent, commutative semiring and $\oplus_{12}$ is $\top_{12}$-annihilating.*

Remark 5.1. It can be noticed that, *a priori*, the order of the primitive annotations matters to the definition of compound annotations.

Quasihomomorphisms are abstract values that may not be representable syntactially. By analogy with XML datatypes [43], we can say that they represent the *value space* of the compound domain. In the following, we want to propose a finite representation of some of these functions. Indeed, as we have seen in the examples above, we intend to represent compound annotations just as finite sets of pairs of primitive annotations. Thus, continuing the analogy, the *lexical space* is merely containing finite sets of pairs of primitive annotation values. To complete the definition, we just have to define a mapping from such finite representation to a corresponding quasihomomorphism. That is, we have to define the *lexical-to-value mapping*.

Consider again the (primitive) domains D_1 and D_2 and let $A \subseteq L_1 \times L_2$ be a finite set of pairs of primitive annotations. We define the function $\overline{A} : D_1 \to D_2$ as follows[15]:

$$\forall z \in L_1, \overline{A}(z) = \texttt{lub}\left\{ \bigotimes_{\langle x,y \rangle \in J} 2\, y \mid J \subseteq A \text{ and } z \preceq_1 \bigoplus_{\langle x,y \rangle \in J} 1\, x \right\}.$$

Theorem 5.2. *If $A \subseteq L_1 \times L_2$ is a finite set of pairs of primitive annotations, then $\overline{A}$ is a quasihomomorphism.*

The proof is mostly a sequence of manipulation of notations with little subtlety, so we refer the reader to A for details.

Now, we know that we can translate an arbitrary finite set of pairs of primitive annotations into a compound annotation. However, using arbitrary sets of pairs is problematic in practice for two reasons: (1) several sets of pairs have equivalent meaning,[16] that is, the function induced by the two sets are identical; (2) the approach does neither gives a programmatic way of computing the operations $(\otimes_{12}, \oplus_{12})$ on compound annotations, nor gives us a tool to finitely represent the results of these operations.

Thus, we next turn towards how to choose a canonical finite representative for a finite set annotation pairs. To this end, we need

---

[15] Note that as D_2 is an annotation domain, the **lub** operation is well defined.
[16] Particularly, we note that there can still be an infinite set of finite representations of the same compound annotation.

a normalising function $N : 2^{L_1 \times L_2} \to 2^{L_1 \times L_2}$ such that for all $A, A' \subseteq L_1 \times L_2, \overline{A} = \overline{A'}$ if and only if $N(A) = N(A')$. This will in turn also allow us to define the operations $\oplus_{12}$ and $\otimes_{12}$ over the set of normalised annotations.

5.2.2.3. Normalisation. We propose a normalisation algorithm based on two main operations:

Saturate: informally, the saturate function increases the size of a set of pairs of annotations by adding any redundant pairs that "result from the application of $\otimes$ and $\oplus$ to values existing in the initial pairs";

Reduce: takes the output of the saturation step and removes "subsumed" pairs.

In particular, the Saturate algorithm is adding pairs of annotations to the input such that in the end, all primitive annotations that can be produced by the use of existing values and operators $\otimes$ and $\oplus$ appear in the output. The algorithm for Saturate, Reduce and Normalise are given in Algorithms 1–3, respectively.

If the operations $\otimes_1$ and $\otimes_2$ are idempotent, Algorithm 1 ensures that given a value $x \in L_1$ that is the result of using operators $\otimes_1$ and $\oplus_1$ on any number of primitive annotations of L_1 appearing in A, then there exists $y \in L_2$ such that $\langle x, y \rangle$ exists in the output of Saturate. Similarly, given $y \in L_2$ that can be obtained from combinations of values of L_2 appearing in A and operators $\otimes_2$ and $\oplus_2$, then there exists $x \in L_2$ such that $\langle x, y \rangle$ exists in the output of Saturate.

Algorithm 1. Saturate (A)

Input: $A \subseteq L_1 \times L_2$ finite
Output: Saturate(A)
    $R := \emptyset$;
    forall $X \subseteq 2^A$ **do**
        $R := R \cup \{\langle \bigoplus_{J \in X} \bigotimes_{\langle x,y \rangle \in J} x, \bigotimes_{J \in X} \bigoplus_{\langle x,y \rangle \in J} y \rangle\}$;
        $R := R \cup \{\langle \bigotimes_{J \in X} \bigoplus_{\langle x,y \rangle \in J} x, \bigoplus_{J \in X} \bigotimes_{\langle x,y \rangle \in J} y \rangle\}$;
    return R;

Example 5.2. Consider the following time + fuzzy annotation:

$$\{\langle [2000, 2005], 0.7 \rangle, \langle [2002, 2008], 0.5 \rangle\}$$

Application of the function saturate gives the following result:

$$\{\langle [2000, 2005], 0.7 \rangle, \langle [2002, 2008], 0.5 \rangle,$$
$$\langle [2000, 2008], 0.35 \rangle, \langle [2002, 2005], 0.7 \rangle,$$
$$\langle [2000, 2005], 0.49 \rangle, \langle [2002, 2005], 0.49 \rangle\}.$$

Now we notice that this can introduce redundant information, which should be eliminated. This is the goal of the function Reduce which is defined by Algorithm 2.

Algorithm 2: Reduce(A)

Input: $A \subseteq L_1 \times L_2$ finite and saturated
Output: Reduce(A)
    while $\exists \langle x, y \rangle \in A, \exists \langle x', y' \rangle \in A \setminus \{\langle x, y \rangle\}$ such that
    $x \preceq_1 x'$ **and** $y \preceq_2 y'$ **do**
        $R := R \setminus \{\langle x, y \rangle\}$;
    while $\exists \langle x, y \rangle \in A$ such that $x = \perp_1$ **or** $y = \perp_2$ **do**
        $R := R \setminus \{\langle x, y \rangle\}$;
    return R;

Example 5.3. Considering Example 5.2 the output of the Saturate algorithm above, the Reduce function gives the following result:

$$\{\langle [2000, 2005], 0.7 \rangle, \langle [2002, 2008], 0.5 \rangle,$$
$$\langle [2000, 2008], 0.35 \rangle\}$$

Algorithm 3. Normalise(A)

Input: $A \subseteq L_1 \times L_2$ finite
Output: Normalise(A)
    return Reduce(Saturate(A));

Example 5.4. Consider the following time + provenance annotation:

$$\{\langle [1998, 2006], \texttt{wikipedia} \rangle, \langle [2001, 2011], \texttt{wrong} \rangle\},$$

which normalises to:

$$\{\langle [1998, 2011], \texttt{wikipedia} \wedge \texttt{wrong} \rangle,$$
$$\langle [1998, 2006], \texttt{wikipedia} \rangle, \langle [2001, 2011], \texttt{wrong} \rangle,$$
$$\langle [2001, 2006], \texttt{wikipedia} \vee \texttt{wrong} \rangle\}$$

Note that the pair $\langle [2001, 2006], \texttt{wikipedia} \vee \texttt{wrong} \rangle$ is introduced by Line 6 of Algorithm 1 and is not discarded during the reduction phase.

The following property can be shown:

Proposition 5.3. *If $D_1 = \langle L_1, \oplus_1, \otimes_1, \perp_1, \top_1 \rangle$ is a lattice then, for all $A \subseteq L_1 \times L_2$ finite, $\overline{A} = $ Normalise(A).*

Notice that we must impose that the first primitive domain of annotation is a lattice for the normalisation to work, that is, we need that $z \preceq_1 x$ and $z \preceq_1 y$ iff $z \preceq_1 x \otimes_1 y$. Details of the proof can be found in A.

The following theorem shows that the normalisation is actually unique up to equivalence of the corresponding functions.

Theorem 5.4. *If $D_1 = \langle L_1, \oplus_1, \otimes_1, \perp_1, \top_1 \rangle$ is a lattice then, for all $A, B \subseteq L_1 \times L_2$ finite then $\overline{A} = \overline{B} \iff$ Normalise(A) = Normalise(B).*

Again, to improve readability, we put the proof in A.

5.2.2.4. Operations on normalised annotations. We can now present the operations $\oplus_{12}$ and $\otimes_{12}$ on normalised finite sets of pairs.

- $A \oplus_{12} B = $ Normalise$(A \cup B)$;

- $A \otimes_{12} B = $ Normalise$(\{\langle x \otimes_1 x', y \otimes_2 y' \rangle \mid \langle x, y \rangle, \langle x', y' \rangle \in A \times B\})$.

Finally, with the proposed representation and operations, we devised a systematic approach to compute combination of domains using existing primitive domains. This implies that an implementation would not need to include operators that are specific to a given combination, as long as programmatic modules exist for the primitive annotation domains.

5.2.3. Discussion

Our definition of a compound annotation domain is, to the best of our knowledge, a novelty in settings involving annotations: previous work on Annotated RDF [15,12], annotated logic programmes [16] or annotated database relations [17] have not addressed this issue. We present in this section some considerations with respect to the chosen approach.

1. The normalisation algorithm is not optimised and would prove inefficient if directly implemented "as is". In this part, we have provided a working solution for normalising compound annotation as a mere proof of existence of such a solution. By observing the examples that we provide for the time + fuzzy domain, it seems that the cost of normalising can be reduced significantly with appropriate strategies.
2. As indicated by Theorem 5.4, we only ensure that the normalisation is feasible for a combination of annotation domains where at least one is a lattice. Whether a normalisation function exists in the more general case of two commutative, idempotent, $\top$-annihilating semirings is an open question.
3. The method we provide defines a new domain of annotation in function of existing domains, such that it is possible to reason and to query triples annotated with pairs of values. This does not mean that it is possible to reason with a combination of triples annotated with the values of the first domain, and triples annotated with values of the second domain. For instance, reasoning with a combination of temporally annotated triples and fuzzy annotated triples does not boil down to reasoning over time + fuzzy-annotated triples. The next section discusses this issues and how non-annotated triples can be combined with annotated triples.

5.3. Integrating differently annotated triples in data and queries

While our approach conservatively extends RDFS, we would like to be able to seamlessly reason with and query together annotated triples and non-annotated triples. Since non-annotated triples can be seen as triples annotated with boolean values, we can generalise this issue to reasoning and querying graphs annotated with distinct domains. For instinct, let us assume that a dataset provides temporally annotated triples, another one contains fuzzy-annotated triples and yet another is a standard RDF dataset. We want to provide a uniform treatment of all these datasets and even handle the merge of differently annotated triples. Moreover, we expect to allow multiple annotation domains in AnQL queries.

5.3.1. Multiple annotation domains in the data
Consider the following example:

$(\text{chadHurley}, \text{type}, \text{googleEmp}) : [2006, 2010]$

$(\text{chadHurley}, \text{type}, \text{googleEmp}) : 0.7$

$(\text{googleEmp}, \text{sc}, \text{Person}) : 0.97$

We can assume that the subclass relation has been determined by ontology matching algorithms, which typically return confidence measures in the form of a number between 0 and 1. Consider as well the following example queries:

Example 5.5.

```
SELECT ?a WHERE {
    (chadHurley type googleEmp):?a
}
```

Example 5.6.

```
SELECT ?a WHERE {
    (chadHurley type Person):?a
}
```

We propose two alternative approaches to deal with multiple annotation domains. The first one simply segregates the domains of annotations, such that no inferences are made across differently annotated triples. The second one takes advantage of the compound domain approach defined in Section 5.2.

5.3.2. Segregation of domains
With this approach, distinct domains are not combined during reasoning, such that the first annotated triple together with the third one would not produce new results. The query from Example 5.5 would have the following answers: $\{?a/[2006, 2010]\}, \{?a/0.7\}$. The query from Example 5.6 would have the answer $\{?a/0.679\}$ (under product t-norm $\otimes$).

The main advantage is that query answering is kept very straightforward. Moreover, it is possible to combine different annotation domains within the query by simply joining results from the segregated datasets. The drawback is that reasoning would not complement non-annotated knowledge with annotated one and vice versa.

5.3.3. Using compound domains
The principle of this approach is to assume that two primitive annotation values from distinct domains actually represent a pair with an implicit default value for the second element. The default value can be domain dependent or generic, such as using $\top$ or $\bot$ systematically. An example of domain specific default is found in [2] where the value $[-\infty, \text{Now}]$ is used to fill the missing annotations in standard, non-annotated RDF. It can be noticed that using $\bot$ as a default would boil down to having segregated datasets, as in the previous approach. The use of $\top$ has the advantage of being generic and allows one to combine knowledge from differently annotated sources in inferences. So, the query in Example 5.5 has answer $\{?a/\{\langle[2006, 2010], 1\rangle, \langle[-\infty, +\infty], 0.7\rangle\}\}$, while Example 5.6 has the answer $\{?a/\{\langle[2006, 2010], 0.97\rangle, \langle[-\infty, +\infty], 0.679\rangle\}\}$.

The main advantage is the possibility to infer new statements by combining various annotated or non-annotated triples. The drawbacks are that (i) our combination approach is, so far, limited to the case where one domain is a lattice; (ii) if triples with a new annotation domain are added, then it adds one dimension to the answers, which obliges to recompute existing answers; (iii) the combination of more than two domains may be particularly complex and possibly non-commutative.

5.3.4. Multiple annotation domains in the query
When dealing with multiple domains in the query, we face a similar choice as in the data, but we are also offered the option to replace the default value with a variable. If segregation of domains has been chosen, then distinct domains in the query are only used to match the corresponding data, but it is still possible to combine the results from differently annotated sources. For instance:

```
SELECT ?e ?c ?t ?f WHERE {
    {(?c sc ebayEmp):?f}
  UNION
    {(?e type ?c):?t.}
  FILTER{?t ⪯_t [2005,2011] OR ?f ⪯_f 0.5}
```

This query can be executed even on a dataset that does not include fuzzy value. The fuzzy-annotated triple pattern can be simply ignored and the temporally annotated pattern evaluated. As mentioned in Definition 4.9, the comparators $\preceq_t$ and $\preceq_f$ return false whenever at least one of the operands is an unbound variable, so

that the `FILTER` is not influenced by a missing type of annotations.

In the case of the second approach using compound domains, the choices are as follow:

1. add a single fresh annotation variable for all triples in the query that are missing a value for an annotation domain; or
2. add a different fresh annotation variable for each triple in the query; or
3. add a constant annotation such as ⊤ to all missing annotation values.

In later discussions, we will use the meta-variable Θ_D to represent the default value of domain d assigned to annotations in the query triples.

Example 5.7. For instance, if we again consider the query (excluding the annotation variables) and input data from Example 4.1, the query would look like:

```
SELECT ?p ?c WHERE {
    (?p type :ebayEmp)
    OPTIONAL{(?p :hasCar ?c)}
}
```

Now, given the above three approaches for transforming this query we would get the following answers:

Approach 1	$?p$/toivo	-
	$?p$/toivo	$?c$/peugeot
	$?p$/toivo	$?c$/renault
Approach 2	$?p$/toivo	$?c$/peugeot
	$?p$/toivo	$?c$/renault
Approach 3		$\emptyset$

5.3.5. Querying multi-dimensional domains

Similarly to the discussion in the previous subsection, we can encounter mismatches between the Annotated RDF dataset and the AnQL query. In case the AnQL query contains only variables for the annotations, the query can be answered on any Annotated RDF dataset. From a user perspective, the expected answers may differ from the actual annotation domain in the dataset, e.g., the user may be expecting temporal intervals in the answers when the answers actually contain a fuzzy value. For this reason some built-in predicates to determine the type of annotation should be introduced, like isTEMPORAL, isFUZZY, etc.

If the AnQL query contains annotation values and the Annotated RDF dataset contains annotations from a different domain, one option is to not provide any answers. Alternatively, we can consider combining the domain of the query with the domain of the annotation into a multi-dimensional domain, as illustrated in the next example.

Example 5.8. Assuming the following input data:

$(\text{chadHurley}, \text{type}, \text{youtubeEmp}) : \text{chad}$

When performing the following query:

```
SELECT ?p ?c WHERE {
    (?p type ?c):[2009, 2010]
}
```

we would interpret the data to the form:

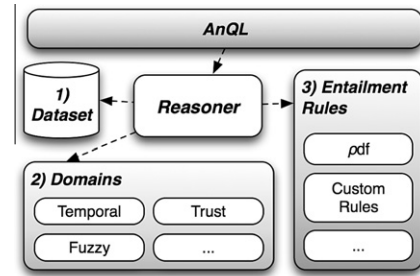$(\text{chadhurley}, \text{type}, \text{youtubeEmp}) : \langle \text{chad}, \Omega_{\text{temporal}} \rangle$

Fig. 4. Annotated RDF implementation schema.

while the query would be interpreted as:

```
SELECT ?p ?c WHERE {
    (?p type ?c):⟨Θ_provenance, [2009, 2010]⟩
}
```

where Ω_{temporal} and $\Theta_{\text{provenance}}$ are annotations corresponding to the default values of their respective domains, as discussed in Section 5.3.4. The semantics of combining different domains into one multi-dimensional domain has been discussed in Section 5.2.

6. Implementation notes

Our prototype implementation is split into two distinct modules: one that implements the Annotated RDFS inferencing and the second module is an implementation of the AnQL query language that relies on the first module to retrieve the data. Our prototype implementation is based on SWI-Prolog's Semantic Web library [44] and we present the architecture of the implementation in Fig. 4.

Our Annotated RDFS module consists of a bottom-up reasoner used to calculate the closure of a given RDF dataset (1). The variable components comprise (2) the specification of the given annotation domain; and (3) the ruleset describing the inference rules and the way the annotation values should be propagated. For (1) we do not suggest a special RDF serialisation for temporal triples but rely on existing proposals using reification [2]. Annotation domains in (2) are to be specified by appropriate lattice operations and describing default annotations for non-annotated triples.

The rules in (3) are specified using a high-level language to specify domain independent rules that abstracts from peculiarities of the reification syntax. For example the following rule provides *subclass inference* in the RDFS ruleset:

```
rdf(O, rdf:type, C2, V) <==
    rdf(O, rdf:type, C1, V1),
    rdf(C1, rdfs:subClassOf, C2, V2),
    infimum(V1, V2, V).
```

(**2**) and (**3**) are independent of each other: it is possible to combine arbitrary rulesets and domains (see above).

The AnQL module also implemented in Prolog relies on the SPARQL implementation provided by the ClioPatria Semantic Web Server.[17] For the AnQL implementation, the domain specification needs to be extended with the grammar rules to parse an annotation value and any built-in functions specific to the domain.

More information and downloads of the prototype implementation can be found at http://anql.deri.org/.

6.1. Implementation of specific domains

For example, for the fuzzy domain the default value is considered to be 1 and the $\otimes$ and $\oplus$ operations are, respectively, the *min* and *max* operations. The AnQL grammar rules consist simply of calling the parser predicate that parses a decimal value.

As for the temporal domain, we are representing triple annotations as *ordered* list of disjoint time intervals. This implies some additional care in the construction of the $\otimes$ and $\oplus$ operations. For the representation of $-\infty$ and $+\infty$ we are using the `inf` and `sup` Prolog atoms, respectively. Concrete time points are represented as integers and we use a standard constraint solver over finite domains (CLPFD) in the $\otimes$ and $\oplus$ operations. The default value for non-annotated triples is `[inf,sup]`. The $\otimes$ operation is implemented as the recursive intersection of all the elements of the annotation values, i.e., temporal intervals. The $\oplus$ operation is handled by constructing CLPFD expressions that evaluate the union of all the temporal intervals. Again, the AnQL grammar rules take care of adapting the parser to the specific domain and we have defined the domain built-in operations described in Section 5.1.

6.2. Use-case example: sensor data

As a use-case for Annotated RDF and AnQL, we present the scenario of exposing sensor readings as RDF data. Representing sensor data as RDF, more specifically as Annotated RDF, enables not only a precise and correct representation for sensor data but also the possibility of interlinking the data with other existing sources on the Web.

Consider the scenario in which each person is assigned a sensor tag (mode) to use in a building that is equipped with several sensor base-stations (that will be responsible for recording the presence of tags). Whenever sensor modes are detected in the proximity of a base-station, sensor readings are created. Normally this sensor reading will contain the time of the reading, the identifier of the base-station and the tag. For our example we used datasets publicly available, that represent movements of persons in a conference. For our test purposes we used a subset of the dataset available at http://people.openpcd.org/meri/openbeacon/sputnik/data/24c3/ with a 1 h time frame.

For the specific Annotated RDF domain, we can take as starting point the temporal domain, where each triple is annotated with a temporal validity. Conceptually, a temporally annotated triple would look like the following:

```
(tag4302 locatedIn room103):
    [2010-07-28T16:52:00Z, 2010-07-28T14:59:00Z]
```

stating that the tag represented by the URI `tag4302` was in the room identified by `room103` during the specified time period. For the URIs we can define a domain vocabulary or rely on an already existing vocabulary.

Since a sensor mode can, at any time, be discovered by several base-stations the issue arrises of how to detect which base-station it is closer to. This can be viewed as a data cleanup process that can be achieved as a post-processing step over the stored data. In our specific experiment, the sensor readings were of the following format:

```
2010-10-11 14:57:51 10.254.2.15 4302 83
2010-10-11 14:57:51 10.254.3.1 4302 83
2010-10-11 14:57:51 10.254.2.6 4302 83
```

where the columns represent respectively: (1) `timestamp` when the record was created; (2) `ip` address of the base station; (3) `tag` identifier and; (4) `ssi`. The `ssi` represents the signal strength of the response from the tag. Each base station registers each tag at the same timestamp with different signal strengths, which can be interpreted as the lower the signal strength value is, the closer the tag is to the base station. This value can then be used in the data cleanup process to discard the base station records in which the tag is furthest from.

In the data cleanup process we start by grouping all the `ips` (with the lowest `ssi`) for a given `timestamp` and `tag`. After this step we can merge all records that share the `tag` and `ip` and have consecutive `timestamp` into a single interval.

6.3. Concrete syntax for Annotated RDFS

In this paper, we stay agnostic with respect to the concrete syntax used to serialise annotated graphs. However, we present here a suggested syntax based on N-Quads [45]. Other possibilities, for instance using named graphs, could be exploited as well, but we believe the following proposal is more straightforward.

We first present how primitive domains are represented, then extend it to combination of domains. In the format N-Quads, the data structure is organised in quadruples:

$$< subject >< predicate >< object >< context > .$$

where `<subject>` `<predicate>` `<object>`. form a valid N-Triple [46] statement. The element `<context>` is an optional context value that belongs to the set **UBL**. In the case of Annotated RDFS, `<context>` will be a typed literal that denotes the annotated value. The datatype of the literal represents the domain of annotation. Consequently, in order to define a concrete representation of an annotation domain, one must at least define its associated datatype, which has three components [47]:

- a lexical space, that is, a set of Unicode character strings (e.g., the set of finite sequences of digits);
- a value space, that is, the set of values represented by the annotations (e.g., the set of time points);
- a lexical-to-value mapping which maps each member of the lexical space to a value in the value space.

In addition to these components, which are mandated by the XML Schema Datatypes specification, the specification of a concrete domain must provide the operations $\otimes$ and $\oplus$, from which the partial order $\preceq$, the upper and lower bounds are also derived. We give as an example the specification of the fuzzy domain. The lexical space, value space and lexical-to-value mapping are the same as the XSD datatype *xsd:decimal*. However, it is not possible to simply reuse this datatype, as it does not specify which operations should be used. Since several t-norms can be used as an $\otimes$ operation, it is necessary to mint distinct URIs for distinct fuzzy semi-rings. Let us call `ex:fuzzy-min` the fuzzy domain with the min t-norm. A fuzzy-annotated triple would look as follows:[18]

```
<ex> <type> <EbayEmp> "0.3"^^ ex:fuzzy-min.
```

With this approach, our framework is made compatible with RDF stores that understand N-Quads. A typical store would simply consider distinct annotation values as different contexts. However, a store implementing Annotated RDFS would be able to easily recognise the annotation domains of the data and load an appropriate reasoning engine dynamically. The reasoner could simply reason separately with different annotation domains.

---

[18] For brevity, we simplify the URIs. A real N-Quads document should have complete URIs all the time.

As compound domains are annotation domains as well, it is possible to use the same approach by providing a distinct lexical/value space for each pair of domains. As an example, one could imagine the following triple occurring in an N-Quads document[19]:

```
<s> <p> <o> "(0.3,[2001,2003])"^^:fuzzy-time.
```

Yet, the purpose of Section 5.2 was to define the combination of domains in function of the primitive domains. Thus, a smart implementation of Annotated RDFS should be able to deal with arbitrary combinations dynamically when they occur in the data. However, this means that there should be a generic syntax for writing compound annotations. To that extent, we propose to extend the N-Quads syntax to a generic tuple syntax where each element of the tuple is in **UBL**. The example above would then look as follows:

```
<s> <p> <o> "0.3"^^:fuzzy "[2001,2003]"^^:time.
```

Few RDF stores are able to parse arbitrary tuples with the notable exception of YARS [48].

7. Conclusion

In this paper we have presented a generalised RDF annotation framework that conservatively extends the RDFS semantics, along with an extension of the SPARQL query language to query annotated data. The framework presented here is generic enough to cover other proposals for RDF annotations and their query languages. Our approach extends the classical case of RDFS reasoning with features of different annotation domains, such as temporality, fuzzyness, trust, etc. and presents a uniform and programatic way to combine any annotation domains.

Furthermore, we presented a semantics for an extension of the SPARQL query language, AnQL, that enables querying RDF with annotations. Queries exemplified in related literature for specific extensions of SPARQL can be expressed in AnQL. Noticeably, our semantics goes beyond the expressivity of the current SPARQL specification and includes some features from SPARQL 1.1 such as aggregates, variable assignments and sub-queries. We also described our implementation of AnQL based on constraint logic programming techniques along with a practical experiment for representing sensor data as Annotated RDF.

Acknowledgement

Appendix A. Proofs of theorems and propositions

A.1. Proof of Theorem 5.2

We start by proving that for all $z, z' \in L_1, \overline{A}(z \oplus_1 z') \succeq_2 \overline{A}(z) \otimes_2 \overline{A}(z')$.

Proof Appendix A. 1. Let $z, z' \in L_1$. In order to prove the proposition, we introduce the notation $K_z^A =_{def} \{J \subseteq A \mid z \preceq_1 \bigoplus_{1 \langle x,y \rangle \in J} x\}$. The property that we want to prove can be rewritten:

$$\text{lub}\left\{\bigotimes_{2 \langle x,y \rangle \in J} y \mid J \in K_{z \oplus_1 z'}^A\right\} \succeq_2$$
$$\text{lub}\left\{\bigotimes_{2 \langle x,y \rangle \in J} y \mid J \in K_z^A\right\} \otimes_2 \text{lub}\left\{\bigotimes_{2 \langle x,y \rangle \in J'} y \mid J' \in K_{z'}^A\right\}.$$

Let us introduce two intermediary lemmas:

Lemma Appendix A. 1. *For all* $z, z' \in L_1, K_{z \oplus_1 z'}^A = K_z^A \cap K_{z'}^A$.

Proof Appendix A. 2. We simply prove each inclusion separately:

- $\subseteq$: let $J \in K_{z \oplus_1 z'}^A$, which implies that $z \oplus_1 z' \preceq_1 \bigoplus_{1 \langle x,y \rangle \in J} x$. So, $z \preceq_1 \bigoplus_{1 \langle x,y \rangle \in J} x$ and $z' \preceq_1 \bigoplus_{1 \langle x,y \rangle \in J} x$, that is $J \in K_z^A$ and $J \in K_{z'}^A$.
- $\supseteq$: let $J \in K_z^A \cap K_{z'}^A$, which implies that $z \preceq_1 \bigoplus_{1 \langle x,y \rangle \in J} x$ and $z' \preceq_1 \bigoplus_{1 \langle x,y \rangle \in J} x$, so $z \oplus_1 z' \preceq_1 \bigoplus_{1 \langle x,y \rangle \in J} x$ by definition of $\oplus_1$. Consequently, $J \in K_{z \oplus_1 z'}^A$.

Lemma Appendix A. 2. *For all* $z, z' \in L_1, K_{z \oplus_1 z'}^A = \{J \cup J' \mid (J, J') \in K_z^A \times K_{z'}^A\}$.

Proof Appendix A. 3. Again, we prove each inclusion separately:

- $\subseteq$: trivial since $J \in K_{z \oplus_1 z'}^A$ implies that $J \in K_z^A \cap K_{z'}^A$ and $J = J \cup J$.
- $\supseteq$: let $J \in K_z^A$ and $J' \in K_{z'}^A$. Clearly, $\bigoplus_{1 \langle x,y \rangle \in J} x \preceq_1 \bigoplus_{1 \langle x,y \rangle \in J \cup J'} x$. Consequently, $J \cup J' \in K_z^A$. Symmetrically, we prove that $J \cup J' \in K_{z'}^A$.

This allows us to rewrite the problem into:

$$\text{lub}\{\bigotimes_{2 \langle x,y \rangle \in J \cup J'} y \mid (J, J') \in K_z^A \times K_{z'}^A\} \succeq_2$$
$$\text{lub}\{\bigotimes_{2 \langle x,y \rangle \in J} y \mid J \in K_z^A\} \otimes_2 \text{lub}\{\bigotimes_{2 \langle x,y \rangle \in J'} y \mid J' \in K_{z'}^A\}$$

which is more concisely written:

$$\bigoplus_{2 (J,J') \in K_z^A \times K_{z'}^A} \left(\bigotimes_{2 \langle x,y \rangle \in J \cup J'} y\right) \succeq_2 \bigoplus_{2 J \in K_z^A} \left(\bigotimes_{2 \langle x,y \rangle \in J} y\right) \otimes_2 \bigoplus_{2 J' \in K_{z'}^A} \left(\bigotimes_{2 \langle x,y \rangle \in J'} y\right).$$

This is easily established by distributivity of $\otimes_2$ over $\oplus_2$ in the right hand side and by remarking that [20]

$$\bigoplus_{2 J' \in K_{z'}^A} \left(\bigoplus_{2 J \in K_z^A} (\bigotimes_{2 \langle x,y \rangle \in J} y \otimes_2 \bigotimes_{2 \langle x,y \rangle \in J'} y)\right) \preceq_2 \bigoplus_{2 (J,J') \in K_z^A \times K_{z'}^A} \left(\bigotimes_{2 \langle x,y \rangle \in J \cup J'} y\right).$$

The second part of the proof demonstrates that for all $z, z' \in L_1, \overline{A}(z \otimes_1 z') \succeq_2 \overline{A}(z) \oplus_2 \overline{A}(z')$.

Proof Appendix A. 4. Before giving the main arguments for the proof, we rewrite the goal as follows: [21]

$$\text{lub}\{\bigotimes_{2 \langle x,y \rangle \in J} y \mid J \in K_z^A\} \oplus_2 \text{lub}\{\bigotimes_{2 \langle x,y \rangle \in J'} y \mid J' \in K_{z'}^A\}$$
$$\preceq_2 \text{lub}\{\bigotimes_{2 \langle x,y \rangle \in J} y \mid J \in K_{z \otimes_1 z'}^A\},$$

---

[19] This is again a fictitious example for the sake of illustration. A more realistic syntax for the representation of time would use XSD dateTime literals.

[20] Note that if $\otimes_2$ is idempotent, then the inequality becomes an equality.
[21] We here reuse the notation K_z^A introduced above.

which again can be made more concise with the following notation:

$$\bigoplus_{2 \atop J \in K_z^A} \left(\bigotimes_{2 \atop \langle x,y\rangle \in J} y\right) \oplus_2 \bigoplus_{2 \atop J \in K_{z'}^A} \left(\bigotimes_{2 \atop \langle x,y\rangle \in J'} y\right) \preceq_2 \bigoplus_{2 \atop J \in K_{z\otimes_1 z'}^A} \left(\bigotimes_{2 \atop \langle x,y\rangle \in J} y\right).$$

Associativity of $\oplus_2$ simplifies the equation further:

$$\bigoplus_{2 \atop J \in K_z^A \cup K_{z'}^A} \left(\bigotimes_{2 \atop \langle x,y\rangle \in J} y\right) \preceq_2 \bigoplus_{2 \atop J \in K_{z\otimes_1 z'}^A} \left(\bigotimes_{2 \atop \langle x,y\rangle \in J} y\right).$$

We established the result by first proving the following lemma:

Lemma Appendix A. 3. *For all $z, z' \in L_1$, $K_z^A \cup K_{z'}^A \subseteq K_{z\otimes_1 z'}^A$.*

Proof Appendix A. 5. Let $J \in K_z^A$. It holds that $z \preceq_1 \bigoplus_{1 \atop \langle x,y\rangle \in J} x$ and $z \otimes_1 z' \preceq_1 z$. So $J \in K_{z\otimes_1 z'}^A$. Idem for any $J \in K_{z'}^A$. This allows us now to easily see that

$$\bigoplus_{2 \atop J \in K_z^A \cup K_{z'}^A} \left(\bigotimes_{2 \atop \langle x,y\rangle \in J} y\right) \preceq_2 \bigoplus_{2 \atop J \in K_{z\otimes_1 z'}^A} \left(\bigotimes_{2 \atop \langle x,y\rangle \in J} y\right).$$

Notice that the opposite inequality does not hold in general.

A.2. Proof of Theorem 5.4

In order to prove the theorem, we first demonstrate the following proposition:

Proposition Appendix A. 4. *If $D_1 = \langle L_1, \oplus_1, \otimes_1, \bot_1, \top_1 \rangle$ is a lattice then, for all $A \subseteq L_1 \times L_2$ finite, $\overline{A} = \overline{\text{Normalise}(A)}$.*

Let us assume that D_1 is a lattice. We show the proposition by proving that at each step of the saturation and reduction, the set R is such that $\overline{R} = \overline{A}$. This is trivially true at the initialisation of Saturate. Now let us assume that R satisfies this property at a certain step of the execution. We start by ensuring that

$$\overline{R} = \overline{R \cup \{\langle \bigoplus_{1 \atop J \in X} \bigotimes_{1 \atop \langle x,y\rangle \in J} x, \bigotimes_{2 \atop J \in X} \bigoplus_{2 \atop \langle x,y\rangle \in J} y\rangle\}}$$

and

$$\overline{R} = \overline{R \cup \{\langle \bigotimes_{1 \atop J \in X} \bigoplus_{1 \atop \langle x,y\rangle \in J} x, \bigoplus_{2 \atop J \in X} \bigotimes_{2 \atop \langle x,y\rangle \in J} y\rangle\}}.$$

We can decompose further the proof by simply showing that, given $\langle a,b\rangle, \langle c,d\rangle \in R$,

$$\overline{R} = \overline{R \cup \{\langle a\otimes_1 c, b\oplus_2 d\rangle\}}$$

and

$$\overline{R} = \overline{R \cup \{\langle a\oplus_1 c, b\otimes_2 d\rangle\}}.$$

To structure the proof better, we split the proof into several lemmas corresponding to each of the aforementioned steps.

Lemma Appendix A. 5. *Let $z \in L_1$. The equality $\overline{R \cup \{\langle a\otimes_1 c, b\oplus_2 d\rangle\}}(z) = \overline{R}(z)$ holds.*

Proof Appendix A. 6. If the pair $\langle a\otimes_1 c, b\oplus_2 d\rangle$ already belongs to R, the equality is trivial. Let us assume $\langle a\otimes_1 c, b\oplus_2 d\rangle \notin R$ so that we can easily distinguish between sets that include $\langle a\otimes_1 c, b\oplus_2 d\rangle$ and sets that do not. Using the definition of $\overline{R \cup \{\langle a\otimes_1 c, b\oplus_2 d\rangle\}}(z)$, we can write:

$$\overline{R \cup \{\langle a\otimes_1 c, b\oplus_2 d\rangle\}}(z) = \overline{R}(z) \oplus_2$$
$$\text{lub}\{(\bigotimes_{2 \atop \langle x,y\rangle \in J} y)\otimes_2(b\oplus_2 d) \mid J \subseteq R \text{ and}$$
$$z \preceq_1 (\bigoplus_{1 \atop \langle x,y\rangle \in J} x)\oplus_1(a\otimes_1 c)\}.$$

Further, due to distributivity,

$$(\bigotimes_{2 \atop \langle x,y\rangle \in J} y)\otimes_2(b\oplus_2 d) = ((\bigotimes_{2 \atop \langle x,y\rangle \in J} y)\otimes_2 b)\oplus_2((\bigotimes_{2 \atop \langle x,y\rangle \in J} y)\otimes_2 d).$$

We can therefore rewrite the previous equality to:

$$\overline{R \cup \{\langle a\otimes_1 c, b\oplus_2 d\rangle\}}(z) = \overline{R}(z)$$
$$\oplus_2 \text{lub}\{(\bigotimes_{2 \atop \langle x,y\rangle \in J} y)\otimes_2 b \mid J\subseteq R \text{ and } z\preceq_1(\bigoplus_{1 \atop \langle x,y\rangle \in J} x)\oplus_1(a\otimes_1 c)\}$$
$$\oplus_2 \text{lub}\{(\bigotimes_{2 \atop \langle x,y\rangle \in J} y)\otimes_2 d \mid J\subseteq R \text{ and } z\preceq_1(\bigoplus_{1 \atop \langle x,y\rangle \in J} x)\oplus_1(a\otimes_1 c)\}.$$

Additionally, since D_1 is a lattice, we have

$$(\bigoplus_{1 \atop \langle x,y\rangle \in J} x)\oplus_1(a\otimes_1 c) = ((\bigoplus_{1 \atop \langle x,y\rangle \in J} x)\oplus_1 a)\otimes_1((\bigoplus_{1 \atop \langle x,y\rangle \in J} x)\oplus_1 c).$$

So $z\preceq_1(\bigoplus_{1 \atop \langle x,y\rangle \in J} x)\oplus_1(a\otimes_1 c)$ implies $z\preceq_1(\bigoplus_{1 \atop \langle x,y\rangle \in J} x)\oplus_1 a$ and $z\preceq_1(\bigoplus_{1 \atop \langle x,y\rangle \in J} x)\oplus_1 c$. This means that $J\cup\{\langle a,b\rangle\} \in \{K\subseteq R \mid z\preceq_1 \bigotimes_{1 \atop \langle x,y\rangle \in J} x\}$ so necessarily, $(\bigotimes_{2 \atop \langle x,y\rangle} y)\otimes_2 b\preceq_2\overline{R}(z)$. Analogically, we conclude that $(\bigotimes_{2 \atop \langle x,y\rangle \in J} y)\otimes_2 d\preceq_2\overline{R}(z)$ and generalising to any suitable J, we conclude, using the equation above, that $\overline{R \cup \{\langle a\otimes_1 c, b\oplus_2 d\rangle\}}(z) = \overline{R}(z)$.

Now let us prove this second equality:

Lemma Appendix A. 6. *Let $z \in L_1$. The equality $\overline{R \cup \{\langle a\oplus_1 c, b\otimes_2 d\rangle\}}(z) = \overline{R}(z)$ holds.*

Proof Appendix A. 7. We apply a similar method as for Lemma 5 to get to the following equality:

$$\overline{R \cup \{\langle a\oplus_1 c, b\otimes_2 d\rangle\}}(z) = \overline{R}(z) \oplus_2$$
$$\text{lub}\{(\bigotimes_{2 \atop \langle x,y\rangle \in J} y)\otimes_2(b\otimes_2 d) \mid J\subseteq R \text{ and}$$
$$z\preceq_1(\bigoplus_{1 \atop \langle x,y\rangle \in J} x)\oplus_1(a\oplus_2 c)\}.$$

This means that $J\cup\{\langle a,b\rangle,\langle c,d\rangle\} \in \{K\subseteq R \mid z\preceq_1 \bigoplus_{1 \atop \langle x,y\rangle \in K} x\}$, which implies that $(\bigotimes_{2 \atop \langle x,y\rangle \in J} y)\otimes_2(b\otimes_2 d)\preceq_2\overline{R}(z)$. Generalising this to any suitable J, we obtain the equality.

Now, let us prove that the reduce algorithm preserves the quasi homomorphism.

Lemma Appendix A. 7. $\overline{A} = \overline{\text{Reduce}(A)}$.

Proof Appendix A. 8. Let $\langle a,b\rangle \in R$ such that there exists $\langle a',b'\rangle \in R$ such that $a \preceq_1 a'$ and $b \preceq_2 b'$. Using the same approach as in Lemma 5, we obtain the following equality:

$$\overline{R}(z) = \overline{R \setminus \{\langle a,b\rangle\}}(z) \oplus_2$$
$$\text{lub}\{(\bigotimes_{2 \atop \langle x,y\rangle \in J} y)\otimes_2 b \mid J\subseteq R \setminus \{\langle a,b\rangle\} \text{ and}$$
$$z\preceq_1(\bigoplus_{1 \atop \langle x,y\rangle \in J} x)\oplus_1 a\}.$$

From the hypothesis, we have that $z\preceq_1(\bigoplus_{1 \atop \langle x,y\rangle \in J} x)\oplus_1 a'$ for any appropriate J. Moreover, for the same J, we have $(\bigotimes_{2 \atop \langle x,y\rangle \in J} y)\otimes_2 b\preceq_2 b'$. Generalising to all adequate J, we entail that:

$$\overline{R \setminus \{\langle a,b \rangle\}}(z) \succeq_2 \text{lub}\{(\bigotimes_{\langle x,y \rangle \in J} y) \otimes_2 b \mid J \subseteq R \setminus \{\langle a,b \rangle\}\text{and}$$

$$z \preceq_1 (\bigoplus_{\langle x,y \rangle \in J} x) \oplus_1 a\}$$

and, thus, $\overline{R}(z) = \overline{R \setminus \{\langle a,b \rangle\}}(z)$. Similarly, every pair $\langle \perp_1, y \rangle$ or $\langle x, \perp_2 \rangle$ does not affect the function $\overline{R}$.

Now, the proof of Proposition 4 follows from an inductive application of Lemmas 5–7. Therefore, $\overline{A} = \overline{\text{Normalise}(A)}$ holds.

A.3. Proof of the theorem

The implication $\Leftarrow$ is a direct consequence of Proposition 4.

Let us prove the other direction. Let A and b two finite sets of pairs of primitive annotations in $L_1 \times L_2$ such that $\overline{A} = \overline{B}$. For $A \subseteq L_1 \times L_2$ and $x \in L_1$, let $K_x^A = \{J \subseteq A \mid x \preceq_1 \bigoplus_{\langle \alpha, \beta \rangle \in J} \alpha\}$. We also remind that:

$$\overline{A} : L_1 \to L_2$$
$$x \mapsto \bigoplus_{J \in K_x^A} \bigotimes_{\langle \alpha, \beta \rangle \in J} \beta$$

Moreover, we introduce the following new notation:

$$\widetilde{A} : L_1 \to L_1$$
$$x \mapsto \bigotimes_{J \in K_x^A} \bigoplus_{\langle \alpha, \beta \rangle \in J} \alpha$$

We establish the proof through the support of several intermediary lemmas.

Lemma Appendix A. 8. *If $\langle x,y \rangle \in A$ then $y \preceq_2 \overline{A}(x)$.*

Proof Appendix A. 9. Let $\langle x,y \rangle \in A$. We remark that $x \preceq_1 \bigoplus_{\langle \alpha, \beta \rangle \in \{\langle x,y \rangle\}} \alpha$ and $\{\langle x,y \rangle\} \subseteq A$, so:

$$y = \bigotimes_{\langle \alpha, \beta \rangle \in \{\langle x,y \rangle\}} \beta \preceq_2 \bigoplus_{J \in K_x^A} \bigotimes_{\langle \alpha, \beta \rangle \in J} \beta = \overline{A}(x).$$

Lemma Appendix A. 9. *If $\langle x,y \rangle \in A$ then $x \preceq_1 \widetilde{A}(x)$.*

Proof Appendix A. 10. Let $\langle x,y \rangle \in A$. For all $J \in K_x^A$, $x \preceq_1 \bigoplus_{\langle \alpha, \beta \rangle \in J} \alpha$ so, since D_1 is a lattice, $x \preceq_1 \bigotimes_{J \in K_x^A} \bigoplus_{\langle \alpha, \beta \rangle \in J} \alpha$, that is, $x \preceq_1 \widetilde{A}(x)$.

Lemma Appendix A. 10. *If $D_1 = \langle L_1, \oplus_1, \otimes_1, \perp_1, \top_1 \rangle$ is a lattice, then a quasihomomorphism is an antitone function, with respect to the orders induced by $\oplus_1$ and $\oplus_2$.*

Proof Appendix A. 11. Assume that D_1 is a lattice. Let f be a quasihomomorphism. Let $x, x' \in L_1$ be two annotation values such that $x \preceq_1 x'$. Then $f(x) = f(x \otimes x') \succeq_2 f(x) \oplus_2 f(x')$ and, thus, $f(x) \succeq_2 f(x')$.

Using the previous lemmas, we can prove two additional lemmas that will bring us to the final proof:

Lemma Appendix A. 11. *If $\langle x,y \rangle \in \text{Normalise}(A)$ then $x = \widetilde{A}(x)$ and $y = \overline{A}(x)$.*

Proof Appendix A. 12. Let $\langle x,y \rangle \in \text{Normalise}(A)$. Consequently, $\langle x,y \rangle \in \text{Saturate}(A)$. Moreover, by definition of Saturate, the pair $\langle \widetilde{A}(x), \overline{A}(x) \rangle$ must exist in Saturate(A). Additionally, from Lemmas 8 and 9, we have that $x \preceq_1 \widetilde{A}(x)$ and $y = \overline{A}(x)$, which implies that $\langle x,y \rangle$ should be eliminated by the Reduce algorithm during normalisation, unless $x = \widetilde{A}(x)$ and $y = \overline{A}(x)$.

Lemma Appendix A. 12. *For all $x \in L_1$, there exists $u \in L_1$ such that $\langle u, \overline{A}(x) \rangle \in \text{Normalise}(A)$ and $x \preceq_1 u$.*

Proof Appendix A. 13. Let $x \in L_1$. Again, $\langle \widetilde{A}(x), \overline{A}(x) \rangle \in \text{Saturate}(A)$. Then, due to the reduction algorithm, there must exist $\langle u,v \rangle \in \text{Normalise}(A)$ such that $u \succeq_1 \widetilde{A}(x)$ and $v \succeq_2 \overline{A}(x)$. We can consider the following assertions:

D_1 is a lattice	(by hypothesis)	(H1)
$\widetilde{A}(x) \preceq_1 u$	(due to Reduce)	(R1)
$\overline{A}(x) \preceq_2 v$	(due to Reduce)	(R2)
$\overline{A}$ is antitone	(from (H1) and Lemma 10)	(A1)
$x \preceq_1 \widetilde{A}(x)$	(from Lemma 9)	(A2)
$\overline{A}(x) \succeq_2 \overline{A}(\widetilde{A}(x))$	(from (A1) and (A2))	(A3)
$\overline{A}(\widetilde{A}(x)) \succeq_2 \overline{A}(u)$	(from (R1) and (A1))	(A4)
$\overline{A}(u) \succeq_2 v$	(from Lemma 8)	(A5)
$\overline{A}(x) = v$	(from (A3), (A4), (A5) and (R2))	(C1)

Assertions (C1) and (C2) establish the lemma.

Proof Appendix A. 14 [Of Theorem 5.4]. Let $\langle x,y \rangle \in \text{Normalise}(A)$. From Lemma 11, we know that $x = \widetilde{A}(x)$ and $y = \overline{A}(x)$. Moreover, from the hypothesis of the theorem, $\overline{A}(x) = \overline{B}(x)$. Hence, due to Lemma 12, there exists $u \in L_1$ such that $\langle u,y \rangle \in \text{Normalise}(B)$ and $x \preceq_1 u$. By using the same reasoning, we can infer that there exists $v \in L_1$ such that $\langle v,y \rangle \in \text{Normalise}(A)$ and $u \preceq_1 v$. But due to Lemma 11, we have that $y = v$ and therefore $\langle x,y \rangle \in \text{Normalise}(B)$.

The situation is symmetrical with respect to A and b, so finally $\text{Normalise}(A) = \text{Normalise}(B)$.

References

[1] F. Manola, E. Miller, RDF Primer, W3C Recommendation, World Wide Web consortium. Available from: <http://www.w3.org/TR/rdf-primer/> (February 10, 2004).

[2] C. Gutiérrez, C.A. Hurtado, A.A. Vaisman, Introducing Time into RDF, IEEE Transactions on Knowledge and Data Engineering 19 (2) (2007) 207–218.

[3] A. Pugliese, O. Udrea, V.S. Subrahmanian, Scaling RDF with time, in: J. Huai, R. Chen, H.-W. Hon, Y. Liu, W.-Y. Ma, A. Tomkins, X. Zhang (Eds.), Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21–25, 2008, ACM, 2008, pp. 605–614.

[4] J. Tappolet, A. Bernstein, Applied Temporal RDF: efficient temporal querying of RDF data with SPARQL, in: Aroyo et al. [49], pp. 308–322.

[5] M. Mazzieri, A.F. Dragoni, A fuzzy semantics for the resource description framework, in: Uncertainty Reasoning for the Semantic Web I, ISWC International Workshops, URSW 2005–2007, Revised Selected and Invited Papers, No. 5327 in Lecture Notes in Computer Science, Springer, 2008, pp. 244–261.

[6] U. Straccia, A minimal deductive system for general fuzzy RDF, in: A. Polleres, T. Swift (Eds.), Web Reasoning and Rule Systems, Third International Conference, RR 2009, Chantilly, VA, USA, October 25–26, 2009, Proceedings, Vol. 5837 of Lecture Notes in Computer Science, Springer, 2009, pp. 166–181.

[7] O. Hartig, Querying trust in RDF data with tSPARQL, in: Aroyo et al. [49], pp. 5–20.

[8] S. Schenk, On the semantics of trust and caching in the Semantic Web, in: Proc. of 7th International Semantic Web Conference (ISWC'2008), 2008, pp. 533–549.

[9] R.Q. Dividino, S. Sizov, S. Staab, B. Schueler, Querying for provenance, trust, uncertainty and other meta knowledge in RDF, Journal of Web Semantics 7 (3) (2009) 204–219.

[10] D. Brickley, R. Guha, RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation, World Wide Web consortium. Available from: <http://www.w3.org/> (February 10, 2004).

[11] A. Seaborne, E. Prud'hommeaux, SPARQL Query Language for RDF, W3C Recommendation, World Wide Web consortium. Available from: <http://www.w3.org/TR/rdf-sparql-query/> (January 15, 2008).

[12] U. Straccia, N. Lopes, G. Lukacsy, A. Polleres, A general framework for representing and reasoning with annotated Semantic Web data, in: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10), AAAI Press, 2010, pp. 1437–1442.

[13] N. Lopes, A. Polleres, U. Straccia, A. Zimmermann, AnQL: SPARQLing Up Annotated RDF, in: Proceedings of the International Semantic Web Conference (ISWC-10), No. 6496 in Lecture Notes in Computer Science, Springer-Verlag, 2010, pp. 518–533.

[14] O. Udrea, D.R. Recupero, V.S. Subrahmanian, Annotated RDF, in: The Semantic Web: Research and Applications, 3rd European Semantic Web Conference, ESWC 2006, No. 4011 in Lecture Notes in Computer Science, Springer, 2006, pp. 487–501.

[15] O. Udrea, D.R. Recupero, V.S. Subrahmanian, Annotated RDF, ACM Transactions on Computational Logic 11 (2) (2010) 1–41.

[16] M. Kifer, V. Subrahmanian, Theory of generalized annotated logic programming and its applications, Journal of Logic Programming 12 (1992) 335–367.

[17] T.J. Green, G. Karvounarakis, V. Tannen, Provenance Semirings, in: L. Libkin (Ed.), Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11–13, 2007, Beijing, China, ACM Press, 2007, pp. 31–40.

[18] G. Karvounarakis, Z.G. Ives, V. Tannen, Querying data provenance, in: A.K. Elmagarmid, D. Agrawal (Eds.), SIGMOD Conference, ACM, 2010, pp. 951–962.

[19] F. Baader, M. Knechtel, R. Peñaloza, A generic approach for large-scale ontological reasoning in the presence of access restrictions to the ontology's axioms, in: Bernstein et al. [50], pp. 49–64.

[20] M. Knechtel, R. Peñaloza, A generic approach for correcting access restrictions to a consequence, in: L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije, H. Stuckenschmidt, L. Cabral, T. Tudorache (Eds.), The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30 – June 3, 2010, Proceedings, Part I, vol. 6088 of Lecture Notes in Computer Science, Springer, 2010, pp. 167–182.

[21] M. Mazzieri, A.F. Dragoni, A fuzzy semantics for Semantic Web languages, in: P.C.G. da Costa, K.B. Laskey, K.J. Laskey, M. Pool (Eds.), ISWC–URSW, 2005, pp. 12–22.

[22] M. Mazzieri, A fuzzy RDF semantics to represent trust metadata, in: 1st Workshop on Semantic Web Applications and Perspectives (SWAP2004), Ancona, Italy, 2004, pp. 83–89.

[23] J.J. Carroll, C. Bizer, P.J. Hayes, P. Stickler, Named graphs, Journal of Web Semantics 3 (4) (2005) 247–267.

[24] P. Buneman, E. 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.

[25] A. Hogan, Exploiting RDFS and OWL for Integrating Heterogeneous, Large-Scale, Linked Data Corpora, Ph.D. thesis, Digital Enterprise Research Institute, National University of Ireland, Galway. Available from: <http://aidanhogan.com/docs/thesis/>; defended (2011).

[26] S. Muñoz, J. Pérez, C. Gutiérrez, Minimal deductive systems for RDF, in: E. Franconi, M. Kifer, W. May (Eds.), The Semantic Web: Research and Applications, 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3–7, 2007, Proceedings, vol. 4519 of Lecture Notes in Computer Science, Springer, 2007, pp. 53–67.

[27] P. Hayes, RDF semantics, W3C Recommendation, World Wide Web consortium. Available from: <http://www.w3.org/TR/rdf-mt/> (February 10, 2004).

[28] C. Gutiérrez, C. Hurtado, A.O. Mendelzon, Foundations of Semantic Web Databases, in: A. Deutsch (Ed.), Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 14–16, 2004, Paris, France, ACM, 2004, pp. 95–106.

[29] G. Ianni, T. Krennwallner, A. Martello, A. Polleres, Dynamic querying of mass-storage RDF data with rule-based entailment regimes, in: Bernstein et al. [50], pp. 310–327.

[30] P. Hájek, Metamathematics of Fuzzy Logic, Trends in Logic, KluwerAcademic Publisher, 1998.

[31] E.P. Klement, R. Mesiar, E. Pap, Triangular Norms, Trends in Logic – Studia Logica Library, Kluwer Academic Publishers, 2000.

[32] S. Abramsky, A. Jung, Domain theory, in: S. Abramsky, D.M. Gabbay, T.S.E. Maibaum (Eds.), Handbook of Logic in Computer Science – Vol. 3: Semantic Structures, Oxford University Press, 1994, pp. 1–168.

[33] L. Ding, T. Finin, Y. Peng, P.P. da Silva, D.L. McGuinness, Tracking RDF graph provenance using RDF molecules, Tech. rep., Knowledge System Lab (2005). <ftp://ftp.ksl.stanford.edu/pub/KSL_Reports/KSL-05-06.pdf>.

[34] J. Carroll, C. Bizer, P.J. Hayes, P. Stickler, Named graphs, provenance and trust, in: A. Ellis, T. Hagino (Eds.), Proceedings of the 14th International Conference on World Wide Web, WWW 2005, Chiba, Japan, May 10–14, 2005, ACML Press, 2005, pp. 613–622.

[35] G. Flouris, I. Fundulaki, P. Pediaditis, Y. Theoharis, V. Christophides, Coloring RDF Triples to Capture Provenance, in: Bernstein et al. [50], pp. 196–212.

[36] O. Hartig, Provenance information in the Web of data, in: C. Bizer, T. Heath, T. Berners-Lee, K. Idehen (Eds.), Linked Data on the Web (LDOW 2009), Proceedings of the WWW2009 Workshop on Linked Data on the Web, Madrid, Spain, April 20, 2009, vol. 538 of CEUR Workshop Proceedings, CEUR, 2009.

[37] R. Delbru, A. Polleres, G. Tummarello, S. Decker, Context dependent reasoning for semantic documents in sindice, in: A. Fokoue, Y. Guo, J.H.T. Liebig (Eds.), 4th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2008), 2008.

[38] N.M. Labrador, U. Straccia, Monotonic mappings invariant linearisation of finite posets, Tech. rep., Computing Research Repository, available as CoRR Technical report at http://arxiv.org/abs/1006.2679 (2010).

[39] J. Pérez, M. Arenas, C. Gutiérrez, Semantics and complexity of SPARQL, ACM Transactions on Database Systems 34 (3) (2009) 16:1–16:45.

[40] R. Angles, C. Gutierrez, The expressive power of SPARQL, in: A.P. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T.W. Finin, K. Thirunarayan (Eds.), International Semantic Web Conference, vol. 5318, Springer, 2008, pp. 114–129.

[41] S. Harris, A. Seaborne, SPARQL 1.1 Query Language, W3C Working Draft, W3C (2010). <http://www.w3.org/TR/2010/WD-sparql11-query-20100601/>.

[42] J.F. Allen, Maintaining knowledge about temporal intervals, Communications of the ACM 26 (11) (1983) 832–843.

[43] D. Peterson, S.S. Gao, A. Malhotra, C.M. Sperberg-McQueen, H.S. Thompson, W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes, W3C Working Draft, World Wide Web consortium. Available from: <http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203/> (December 3, 2009).

[44] J. Wielemaker, Z. Huang, L. van der Meij, SWI-Prolog and the Web, Theory and Practice of Logic Programming 8 (3) (2008) 363–392.

[45] R. Cyganiak, A. Harth, A. Hogan, N-Quads: Enxtending N-Triples with Context (2009). Available from: <http://sw.deri.org/2008/07/n-quads/>.

[46] N-Triples (2001). Available from: <http://www.w3.org/2001/sw/RDFCore/ntriples/>.

[47] A.M.P. Biron, XML Schema Part 2: Datatypes, second ed., W3C recommendation, World Wide Web Consortium, 2004.

[48] A. Harth, J. Umbrich, A. Hogan, S. Decker, Yars2: A federated repository for querying graph structured data from the web, in: K. Aberer, K.-S. Choi, N.F. Noy, D. Allemang, K.-I. Lee, L.J.B. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, P. Cudré-Mauroux (Eds.), ISWC/ASWC, vol. 4825 of Lecture Notes in Computer Science, Springer, 2007, pp. 211–224.

[49] A. Bernstein, D.R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, K. Thirunarayan (Eds.), The Semantic Web – ISWC 2009, 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25–29, 2009. Proceedings, vol. 5823 of Lecture Notes in Computer Science, Springer, 2009.

[50] L. Aroyo, P. Traverso, F. Ciravegna, P. Cimiano, T. Heath, E. Hyvönen, R. Mizoguchi, E. Oren, M. Sabou, E.P.B. Simperl (Eds.), The Semantic Web: Research and Applications, 6th European Semantic Web Conference, ESWC 2009, Heraklion, Crete, Greece, May 31 – June 4, 2009, Proceedings, vol. 5554 of Lecture Notes in Computer Science, Springer, 2009.