# AnQL: SPARQLing Up Annotated RDFS

Nuno Lopes[1], Axel Polleres[1], Umberto Straccia[2], and Antoine Zimmermann[1]

[1] Digital Enterprise Research Institute,
National University of Ireland Galway, Ireland
{nuno.lopes,axel.polleres,antoine.zimmermann}@deri.org
[2] Istituto di Scienza e Tecnologie dell'Informazione (ISTI - CNR), Pisa, Italy
straccia@isti.cnr.it

**Abstract.** Starting from the general framework for Annotated RDFS which we presented in previous work (extending Udrea et al's Annotated RDF), we address the development of a query language – AnQL – that is inspired by SPARQL, including several features of SPARQL 1.1. As a side effect we propose formal definitions of the semantics of these features (subqueries, aggregates, assignment, solution modifiers) which could serve as a basis for the ongoing work in SPARQL 1.1. We demonstrate the value of such a framework by comparing our approach to previously proposed extensions of SPARQL and show that AnQL generalises and extends them.

## Introduction

RDF (Resource Description Framework) [14] 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 to RDF were proposed in order to deal with time [7,19,24], truth or imprecise information [15,22], trust [10,20] and provenance [4]. All these proposals share a common approach of extending the RDF language by attaching meta-information about the RDF graph or triple. RDF Schema (RDFS) [3] is the specification of a restricted vocabulary that allows to deduce further information from existing triples. In our previous work [23], we presented a general extension to RDFS, improving on Udrea et al's Annotated RDF [25], that is capable of encapsulating the mentioned RDF extensions as specific domains for RDF annotations. For this general extension, we present a generic RDFS reasoning procedure which can be formulated independently of the annotation domain by being parameterised with operations any domain needs to provide. An overview of Annotated RDFS is presented in Sect. 1.

SPARQL [21] is the W3C-standardised query language for RDF. In this paper we present an extension of SPARQL for querying annotated RDFS. SPARQL shares similarities with SQL although several features, such as aggregates, nested queries and variable assignments, are still missing from the current SPARQL specification. Our SPARQL extension presented here also deals with these missing features thus going beyond the features of SPARQL, heading towards the currently under development SPARQL 1.1 specification [9]. Our extension of SPARQL, called AnQL, is presented in Sect. 2. Furthermore, Sect. 3 presents a discussion of some of the most important issues with the design of our query language along with the comparison to some of the related works.

P.F. Patel-Schneider et al. (Eds.): ISWC 2010, Part I, LNCS 6496, pp. 518–533, 2010.
© Springer-Verlag Berlin Heidelberg 2010

**Related Work.** The basis for Annotated RDF were first established by Udrea *et al.* [25, 26], in which their query language is restricted to conjunctive queries. SPARQL is compared to the presented conjunctive queries but excludes 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.

In [7], Gutiérrez *et al.* present conjunctive queries with built-in predicates for querying temporal RDF, neither considering full SPARQL. Pugliese *et al.* [19] also have a temporal framework where they only define conjunctive queries, thus ignoring some of the more advanced features of SPARQL. Tappolet and Bernstein [24] present temporal extensions for RDF and SPARQL. A storage format for temporal RDF is presented where each time interval is stored as a named graph. The $\tau$-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. The RDF extensions towards uncertain or fuzzy information [15, 22] so far do not address SPARQL, presenting only extensions for RDFS reasoning but [22] formalises conjunctive queries.

SPARQL extensions towards querying trust have been presented by Hartig [10]. 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 FILTER$s$ or ORDER.

Another extension to query meta-knowledge in RDF, mostly considering provenance and uncertainty is presented by Dividino *et al.* [4]. In this work, the meta-information is stored using named graphs and the syntax and semantics of SPARQL are extended to consider an additional expression that enables querying the named graphs representing the meta-information.

Our present work can also be related to annotated relational databases, especially Green *et al.* [6] 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 residuated lattice in our approach).

## 1 Annotated RDFS

For the sake of making the paper self-contained, we recap essential parts from [23], where we only considered ground graphs, while here we do allow blank nodes as well.

### 1.1 Syntax

Consider pairwise disjoint alphabets $\mathbf{U}$, $\mathbf{B}$, and $\mathbf{L}$ denoting, respectively, *URI references*, *blank nodes* (i.e., *variables*, denoted $x, y, z$)[1] and *Literals*.[2] We call the elements

[1] We will often use the term blank node and variable synonymously in this paper.

[2] We assume $\mathbf{U}, \mathbf{B}$, and $\mathbf{L}$ fixed, and for ease we will denote unions of these sets simply concatenating their names.

in $\mathbf{UBL}$ ($\mathbf{B}$) *terms*. An *RDF triple* is $\tau = (s, p, o) \in \mathbf{UBL} \times \mathbf{U} \times \mathbf{UBL}$.[3] We call $s$ the *subject*, $p$ the *predicate*, and $o$ the *object*. An *annotated triple* is an expression $\tau : \lambda$, where $\tau$ is a triple and $\lambda$ is an *annotation value* (defined below). An *annotated graph* $G$ is a finite set of *annotated triples*. The $universe$ of $G$, $universe(G)$, is the set of elements in $\mathbf{UBL}$ that occur in the triples of $G$. A *vocabulary* is a subset of $\mathbf{UL}$.

As in our previous work, for presentation purposes, we rely on a fragment of RDFS, called $\rho$df [16], that covers essential features of RDFS.[4] $\rho$df is defined as the following subset of the RDFS vocabulary: $\rho\text{df} = \{\mathsf{sp}, \mathsf{sc}, \mathsf{type}, \mathsf{dom}, \mathsf{range}\}$. Informally, $(i)$ $(p, \mathsf{sp}, q)$ means that property $p$ is a *subproperty* of property $q$; $(ii)$ $(c, \mathsf{sc}, d)$ means that class $c$ is a *subclass* of class $d$; $(iii)$ $(a, \mathsf{type}, b)$ means that $a$ is of *type* $b$; $(iv)$ $(p, \mathsf{dom}, c)$ means that the *domain* of property $p$ is $c$; and $(v)$ $(p, \mathsf{range}, c)$ means that the *range* of property $p$ is $c$. Annotations are added to triples to attach meta information such as temporal validity, trust or fuzzy value, provenance.

*Example 1.* For instance, the following annotated triple:

$$(:\text{Alain}, :\text{livesIn}, :\text{Paris}) : \texttt{[1980, 1991]}$$

in a temporal setting [7] has intended meaning "Alain lives in Paris from 1980 to 1991", while in the fuzzy setting [22]:

$$(\text{audiTT}, \mathsf{type}, \text{SportsCar}) : 0.8$$

has intended meaning "AudiTT is a sports car to degree not less than 0.8"; considering provenance as annotations:

$$(\text{Person}, \mathsf{sc}, \text{Agent}) : \{\texttt{http://xmlns.com/foaf/0.1/}\}$$

would mean that the subclass relationship between persons and agents is defined by – or, "belongs to" – the document `http://xmlns.com/foaf/0.1/`.

### 1.2 RDFS Annotation Domains

Consider a lattice $\langle L, \preceq \rangle$. Elements in $L$ are our annotation values. The order $\preceq$ is used to express redundant/entailed/subsumed information. For instance, for temporal intervals, an annotated triple $(s, p, o) : \texttt{[2000,2006]}$ entails $(s, p, o) : \texttt{[2003,2004]}$, as $\texttt{[2003,2004]} \subseteq \texttt{[2000,2006]}$ (here, $\subseteq$ plays the role of $\preceq$). Informally, an interpretation will map statements to elements of the annotation domain. Our semantics generalises the one of standard RDFS by using an algebraic structure that is well-known for Many-Valued FOL [8]. We say that an *annotation domain* for RDFS is a residuated bounded lattice $D = \langle L, \preceq, \wedge, \vee, \otimes, \Rightarrow, \bot, \top \rangle$.[5] 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] As in [16] we allow literals for $s$.

[4] Just as in [16] our annotation framework can be extended to full RDFS, adding additional semantic conditions and respective inference rules [13].

[5] We correct here an imprecision in the definition given in [23], in which we did not mention that the structure should be a residuated lattice.

3. $\Rightarrow$ is the so-called residuum of $\otimes$, *i.e.*, for all $\lambda_1, \lambda_2, \lambda_3$, $\lambda_1 \otimes \lambda_3 \preceq \lambda_2$ iff $\lambda_3 \preceq (\lambda_1 \Rightarrow \lambda_2)$.

*Remark 1.* Note that $\lambda_1 \Rightarrow \lambda_2$ can be determined uniquely as $\lambda_1 \Rightarrow \lambda_2 = \sup\{\lambda \mid \lambda_1 \otimes \lambda \preceq \lambda_2\}$ (see [11]). Furthermore, in the remaining of this paper, we do not use the $\wedge$ which is implicitly defined by the order $\preceq$. For these reasons, we represent a domain succinctly as 6-tuple $\langle L, \preceq, \vee, \otimes, \bot, \top\rangle$.

In what follows we define a *map* as a function $\mu : \mathbf{UBL} \to \mathbf{UBL}$ preserving URIs and literals, *i.e.*, $\mu(t) = t$, for all $t \in \mathbf{UL}$. 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 $\mu$ from $G_1$ to $G_2$, and write $\mu : G_1 \to G_2$, if $\mu$ is such that $\mu(G_1) \subseteq G_2$.

### 1.3 Semantics

Fix an annotation domain $D = \langle L, \preceq, \vee, \otimes, \bot, \top\rangle$. Informally, an interpretation $\mathcal{I}$ will assign to a triple $\tau$ an element of the annotation domain $\lambda \in L$, dictating that under $\mathcal{I}$, the annotation of $\tau$ is greater or equal than (i.e., $\succeq$) $\lambda$. 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 $\Delta_R, \Delta_P, \Delta_C, \Delta_L$ 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 $\mathcal{I}$. These have to satisfy:

1. $\Delta_R$ are the resources (the domain or universe of $\mathcal{I}$);
2. $\Delta_P$ are property names (not necessarily disjoint from $\Delta_R$);
3. $\Delta_C \subseteq \Delta_R$ are the classes;
4. $\Delta_L \subseteq \Delta_R$ are the literal values and contains $\mathbf{L} \cap V$;
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 $t \in \mathbf{UL} \cap V$ 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 $\Delta_R$ to each element in $\mathbf{L}$;

Intuitively, a triple $(s, p, o)\colon \lambda$ is satisfied by an annotated interpretation $\mathcal{I}$ if $(s, o)$ belongs to the extension of $p$ to a "wider" extent than $\lambda$. We formalise this intuition in terms of semantic conditions on the use of the RDFS vocabulary. That is, an interpretation $\mathcal{I}$ is a *model* of an annotated ground graph $G$, denoted $\mathcal{I} \models G$, iff $\mathcal{I}$ is an interpretation over the vocabulary $\rho$df $\cup$ $universe(G)$ that satisfies the conditions in Table 1. Here, considering a set $\Delta \subseteq \Delta_R \cup \Delta_P$, we say that a function $p\colon \Delta \times \Delta \to L$ is sup-$\otimes$ *transitive* (or simply *transitive*) over $\Delta$ iff for all $x, z \in \Delta$, $\sup_{y \in \Delta}\{p(x, y) \otimes p(y, z)\} \preceq p(x, z)$.[6]

Finally, entailment among annotated ground graphs $G$ and $H$ is as usual. Now, $G \models H$, where $G$ and $H$ may contain blank nodes, iff for any grounding $G'$ of $G$ there is a grounding $H'$ of $H$ such that $G' \models H'$.[7]

[6] As $\Delta$ is finite, sup-$\otimes$ transitivity is well defined.

[7] A grounding $G'$ of graph $G$ is obtained by replacing variables in $G$ with terms in $\mathbf{UL}$.

**Table 1.** The conditions for annotated interpretations

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

**Subproperty:**
1. $P[\![\mathsf{sp}^{\mathcal{I}}]\!]$ is sup-$\otimes$ transitive over $\Delta_P$;
2. $P[\![\mathsf{sp}^{\mathcal{I}}]\!](p,q) = \inf_{(x,y)\in\Delta_R\times\Delta_R} P[\![p]\!](x,y) \Rightarrow P[\![q]\!](x,y)$;

**Subclass:**
1. $P[\![\mathsf{sc}^{\mathcal{I}}]\!]$ is sup-$\otimes$ transitive over $\Delta_C$;
2. $P[\![\mathsf{sc}^{\mathcal{I}}]\!](c,d) = \inf_{x\in\Delta_R} C[\![c]\!](x) \Rightarrow C[\![d]\!](x)$;

**Typing I:**
1. $C[\![c]\!](x) = P[\![\mathsf{type}^{\mathcal{I}}]\!](x,c)$;
2. $P[\![\mathsf{dom}^{\mathcal{I}}]\!](p,c) = \inf_{(x,y)\in\Delta_R\times\Delta_R} P[\![p]\!](x,y) \Rightarrow C[\![c]\!](x)$;
3. $P[\![\mathsf{range}^{\mathcal{I}}]\!](p,c) = \inf_{(x,y)\in\Delta_R\times\Delta_R} P[\![p]\!](x,y) \Rightarrow C[\![c]\!](y)$;

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

*Remark 2.* In [16], the authors define two variants of the semantics of $\rho$df: the default one includes reflexivity of the subclass and subproperty relations but in the present paper, we extend the alternative semantics presented in [16, Definition 4] which omits this requirement.

*Remark 3.* Note that we always have that $G \models \tau\colon \bot$. Clearly, triples of the form $\tau\colon \bot$ are uninteresting and, thus, in the following we not consider them as part of the language.

As for the crisp case, it can be shown [23] that *any annotated RDFS graph has a finite model* (modulo Remark 3) and, thus, we do not have to care about consistency.

### 1.4 Deductive System

The important feature of the annotation 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 $\otimes$ and $\vee$ operations has to be provided) and, thus, are amenable to an easy implementation on top of existing systems. Specifically, the rule set contains the rules presented in Table 2. Please note that rule 6 from Table 2 is destructive, *i.e.*, this rule removes the premises as the conclusion is inferred, intuitively meaning that only "maximal" annotations are preserved.

*Remark 4.* We point out that rules $2-5$ from Table 2 can be represented concisely using the following inference rule:

$$(A)\ \frac{\tau_1\colon \lambda_1, ..., \tau_n\colon \lambda_n, \{\tau_1, \ldots \tau_n\} \vdash_{\mathsf{RDFS}} \tau}{\tau\colon \bigotimes_i \lambda_i}$$

**Table 2.** Inference rules for annotated RDFS

1. Simple:
   $(a)$ $\frac{G}{G'}$ for a map $\mu : G' \to G$ $(b)$ $\frac{G}{G'}$ for $G' \subseteq G$
2. Subproperty:
   $(a)$ $\dfrac{(A, \mathsf{sp}, B) \colon \lambda_1, (B, \mathsf{sp}, C) \colon \lambda_2}{(A, \mathsf{sp}, C) \colon \lambda_1 \otimes \lambda_2}$ $(b)$ $\dfrac{(D, \mathsf{sp}, E) \colon \lambda_1, (X, D, Y) \colon \lambda_2}{(X, E, Y) \colon \lambda_1 \otimes \lambda_2}$
3. Subclass:
   $(a)$ $\dfrac{(A, \mathsf{sc}, B) \colon \lambda_1, (B, \mathsf{sc}, C) \colon \lambda_2}{(A, \mathsf{sc}, C) \colon \lambda_1 \otimes \lambda_2}$ $(b)$ $\dfrac{(A, \mathsf{sc}, B) \colon \lambda_1, (X, \mathsf{type}, A) \colon \lambda_2}{(X, \mathsf{type}, B) \colon \lambda_1 \otimes \lambda_2}$
4. Typing:
   $(a)$ $\dfrac{(D, \mathsf{dom}, B) \colon \lambda_1, (X, D, Y) \colon \lambda_2}{(X, \mathsf{type}, B) \colon \lambda_1 \otimes \lambda_2}$ $(b)$ $\dfrac{(D, \mathsf{range}, B) \colon \lambda_1, (X, D, Y) \colon \lambda_2}{(Y, \mathsf{type}, B) \colon \lambda_1 \otimes \lambda_2}$
5. Implicit Typing:
   $(a)$ $\dfrac{(A, \mathsf{dom}, B) \colon \lambda_1, (D, \mathsf{sp}, A) \colon \lambda_2, (X, D, Y) \colon \lambda_3}{(X, \mathsf{type}, B) \colon \lambda_1 \otimes \lambda_2 \otimes \lambda_3}$
   $(b)$ $\dfrac{(A, \mathsf{range}, B) \colon \lambda_1, (D, \mathsf{sp}, A) \colon \lambda_2, (X, D, Y) \colon \lambda_3}{(Y, \mathsf{type}, B) \colon \lambda_1 \otimes \lambda_2 \otimes \lambda_3}$
6. Generalisation:
   $\dfrac{(X, A, Y) \colon \lambda_1, (X, A, Y) \colon \lambda_2}{(X, A, Y) \colon \lambda_1 \vee \lambda_2}$

Essentially, this rule says that if a classical RDFS triple $\tau$ can be inferred by applying a classical RDFS inference rule to triples $\tau_1, \ldots \tau_n$ (denoted $\{\tau_1, \ldots, \tau_n\} \vdash_{\mathsf{RDFS}} \tau$), then the annotation term of $\tau$ will be $\bigotimes_i \lambda_i$, where $\lambda_i$ is the annotation of triple $\tau_i$. It follows immediately that, using rule schema $(A)$, the annotated framework extends to the whole RDFS rule set as well. We also assume that rule schema $(A)$ or rule 6 of Table 2 are not applied if the consequence is of the form $\tau \colon \bot$ (see Remark 3).

Finally, like for the classical case, the *closure* is defined as $cl(G) = \{\tau \colon \lambda \mid G \vdash^* \tau \colon \lambda\}$, where $\vdash^*$ is as $\vdash$ for the annotated framework without rule $(1a)$. Notice 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 $\otimes$ and $\vee$ are polynomially bounded (from a computational complexity point of view, it is as for the classical case, plus the cost of the operations $\otimes$ and $\vee$ in $L$).

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

### 1.5 Examples of Domains

Here, we instantiate the definition with several domains that have been discussed in the literature. The interested reader can find more details about the temporal and fuzzy domains in our previous work [23] and additional information in our accompanying technical report [13]. Furthermore, domains can be combined into a multi-dimensional annotation domain as explained in [23].

**Crisp.** Note that with the domain $D_{01} = \langle \{0,1\}, \leqslant, \max, \min, 0, 1\rangle$, Annotated RDFS turns out to be the same as standard RDFS.

**Fuzzy.** The fuzzy domain has been presented in [15, 22] and to model fuzzy RDFS in our framework is easy: the annotation domain is $D_{[0,1]} = \langle [0,1], \leqslant, \max, \otimes, 0, 1\rangle$ where $\otimes$ is any continuous t-norm on $[0,1]$ and $\vee$ is $\max$.

**Temporal.** For modelling the temporal domain we generalise the notions presented in [7,19,24] and consider that *time points* are elements of xsd:dateTimeStamp [18] value space $\cup \{-\infty, +\infty\}$.[8] A *temporal interval* is a non-empty interval $[\alpha_1, \alpha_2]$, where $\alpha_i$ are time points. An empty interval is denoted as $\emptyset$. We define a partial order on intervals as $I_1 \leqslant I_2$ iff $I_1 \subseteq I_2$ and $L$ as (where $\bot = \{\emptyset\}, \top = \{[-\infty, +\infty]\}$). Therefore, a *temporal term* is a finite set of pairwise disjoint time intervals. Furthermore, on $L$ we define the following partial order:

$$t_1 \preceq t_2 \text{ iff } \forall I_1 \in t_1 \exists I_2 \in t_2, \text{ such that } I_1 \leqslant I_2 \ .$$

The join and t-norm $\otimes$ operators are defined as:

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

*Remark 5.* Although we represent time points as dateTimeStamps, for presentation purposes in this paper we will only use years.

**Provenance.** We also generalise the representation of *provenance* as described, e.g., in [4, 5]. In this case, we start from a countably infinite set of *atomic provenances* $\mathbf{P}$. 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 $\models$. 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 $\models$, t-norm and join are $\wedge$ and $\vee$ respectively. We set $\top$ to *true* and $\bot$ to *false*.

**Trust.** For the trust domain we rely on previous work by Schenk [20] that defines a *bilattice* structure to model some form of *trust*. We can directly use this algebraic structure as an annotation domain in our framework.

## 2 AnQL: Annotated SPARQL

We now present an extension of the SPARQL [21] query language, made for querying annotated graphs, which we call AnQL. For the rest of this section we fix a specific annotation domain, $D = \langle L, \preceq, \vee, \otimes, \bot, \top\rangle$.

[8] Note that we have a continuous set of time points as opposed to Gutiérrez *et al.* [7].

### 2.1 Syntax

A *simple AnQL query* is defined – analogously to a SPARQL query – as a triple $Q = (P, G, V)$ where $P$ is an *annotated graph pattern*, the *dataset* $G$ is an annotated graph and $V$ is a set of variables, called the *result form*. We restrict ourselves to SELECT queries in this work so it is sufficient to consider the result form $V$ as a list of variables to be projected.

*Remark 6.* 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 [21] for details.

Triple patterns in annotated AnQL are defined the same way as in SPARQL. A *triple pattern* is a triple $(s, p, o)$ where $s, o \in \mathbf{UBL}$ and $p \in \mathbf{UB}$. We denote variables from $\mathbf{B}$ in triple patterns by '?' prefixed names.[9] Let $\mathbf{V}$ be a distinct set of variables, called *annotation variables*. For a triple pattern $\tau$ and $\lambda$ either an annotation term from $D$ or an annotation variable, we call $\tau : \lambda$ an *annotated triple pattern*; sets of annotated triple patterns are called *basic annotated patterns* (BAP). An *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 [21], and later on), then $(P \text{ AND } P')$, $(P \text{ OPTIONAL } P')$, $(P \text{ UNION } P')$, $(P \text{ FILTER } R)$ are annotated graph patterns.

*Example 2.* Suppose we are looking for people who live near Paris during some time period and optionally owned a car during that period. This query can be posed as follows:

```
SELECT ?p ?c ?l
WHERE {(?p :basedNear :paris):?l OPTIONAL{(?p :hasCar ?c):?l}}
```

Assuming the following input data:

$$(:\text{alain}, :\text{livesIn}, :\text{paris}) : [2007, 2010]$$
$$(:\text{alain}, :\text{hasCar}, :\text{peugeot}) : [2004, 2009]$$
$$(:\text{alain}, :\text{hasCar}, :\text{renault}) : [2010, 2010]$$
$$(:\text{livesIn}, \mathsf{sp}, :\text{basedNear}) : [-\infty, +\infty]$$

we will get the following answers:

$$\theta_1 = \{?p/:\text{alain}, ?l/[2007, 2010]\}$$
$$\theta_2 = \{?p/:\text{alain}, ?c/:\text{peugeot}, ?l/[2007, 2009]\}$$
$$\theta_3 = \{?p/:\text{alain}, ?c/:\text{renault}, ?l/[2010, 2010]\}\,.$$

The first answer corresponds to the answer in which the OPTIONAL pattern is not satisfied, so we get the annotation value $[2007, 2010]$ that corresponds to the time Alain lives in Paris. In the second and third answers, the OPTIONAL pattern is matched and, in this case, the annotation value is restricted to the time when Alain lives in Paris and has a car. □

[9] Note that we do not consider blank nodes in triple patterns separately, since they can be treated just as other variables.

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 3.* A slightly different query can be people who lived near Paris 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 ?c ?l WHERE {(?p :basedNear :paris):?l
              OPTIONAL {(?p :hasCar ?c):?l2 FILTER (?l2 ⪯ ?l)} }
```

Using the input data from Example 2, we obtain the following answers:

$$\theta_1 = \{?p/\text{:alain}, ?l/[2007, 2010]\}$$
$$\theta_2 = \{?p/\text{:alain}, ?c/\text{:renault}, ?l/[2007, 2010]\}$$

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 [2]. □

### 2.2 Semantics

We denote by $var(P)$ the set of variables and annotation variables present in a BAP $P$. A *substitution* $\theta$ for a BAP $P$ is a mapping with domain $var(P)$ (annotation) variables into (annotation) terms occurring in $G$. We denote the *domain* of a substitution $\theta$, i.e. the variables for which $\theta$ is defined, by $dom(\theta)$. 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 $t_i$. Note that we do not allow any assignment of an annotation variables to $\perp$ (of the domain $D$). An annotation value of $\perp$, although it is a valid answer for any triple, does not provide any additional information and thus is of minor interest.

For a BAP $P$, and a substitution $\theta$ we denote by $\theta(P)$ the triples obtained by replacing the variables in $P$ according to $\theta$. By $G \models \theta(P)$ we denote the fact that $\theta(P)$ is entailed by $G$.

For the extension of the SPARQL relational algebra to the annotated case we introduce – inspired by the definitions in [17] – definitions of compatibility and union of substitutions: given two substitutions $\theta_1$ and $\theta_2$, $\theta_1$ and $\theta_2$ are $\otimes$-*compatible* if and only if $(i)$ $\theta_1(x) = \theta_2(x)$ for any non-annotation variable $x \in dom(\theta_1) \cap dom(\theta_2)$; $(ii)$ $\theta_1(\lambda) \otimes \theta_2(\lambda) \neq \perp$ for any annotation variable $\lambda \in dom(\theta_1) \cap dom(\theta_2)$. Further, for two $\otimes$-*compatible* substitutions $\theta_1$ and $\theta_2$the $\otimes$-*union*, denoted $\theta_1 \otimes \theta_2$, is as $\theta_1 \cup \theta_2$, with the exception that for any annotation variable $\lambda \in dom(\theta_1) \cap dom(\theta_2)$, $\theta_1 \otimes \theta_2(\lambda) = \theta_1(\lambda) \otimes \theta_2(\lambda)$.

We are now ready to present the notion of evaluation for generic AnQL graph patterns. 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*,[10] is recursively defined as:

[10] Strictly speaking, we consider *sequences* of answers – note that SPARQL allows duplicates and imposing and order on solutions, cf. Sect. 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 $\{\ldots\}_{\text{DISTINCT}}$.

$[\![P]\!]_G$ = $\{\theta \mid dom(\theta) = var(P)$ and $G \models \theta(P)\}$
$[\![P_1$ 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$\}$
$[\![P_1$ UNION $P_2]\!]_G$ = $[\![P_1]\!]_G \cup [\![P_2]\!]_G$
$[\![P_1$ FILTER $R]\!]_G$ = $\{\theta \mid \theta \in [\![P_1]\!]_G$ and $R\theta$ is true$\}$
$[\![P_1$ OPTIONAL $P_2[R]]\!]_G$ = $\{\theta \mid$ and $\theta$ 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 $\theta_1$ 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 $\theta_1$ and $\theta_2$ $\otimes$-compatible, $R(\theta_1 \otimes \theta_2)$ is false$\}$

*Remark 7.* For practical convenience, we retain in $[\![\cdot]\!]_G$ only "domain maximal answers". That is, let us define $\theta' \preceq \theta$ iff $(i)$ $\theta' \neq \theta$; $(ii)$ $dom(\theta) = dom(\theta')$; $(iii)$ $\theta(x) = \theta'(x)$ for any non-annotation variable $x$; and $(iv)$ $\theta'(\lambda) \preceq \theta(\lambda)$ for any annotation variable $\lambda$. Then, for any $\theta \in [\![P]\!]_G$ we remove any $\theta' \in [\![P]\!]_G$ such that $\theta' \preceq \theta$.

*Remark 8.* Please note that the cases for the evaluation of the OPTIONAL are compliant with the SPARQL specification [21], covering the notion of unsafe FILTER$s$ as presented in [2]. 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 of not. This is the behaviour demonstrated in Example 2.

The notion of *true filter* is defined as follows: for a FILTER expression $R$, the valuation of $R$ on a substitution $\theta$, denoted $R\theta$ is *true* iff:[11]

(1) $R$ = BOUND$(v)$ with $v \in dom(\theta)$;
(2) $R$ = isBLANK$(v)$ with $v \in dom(\theta)$ and $\theta(v) \in \mathbf{B}$;
(3) $R$ = isIRI$(v)$ with $v \in dom(\theta)$ and $\theta(v) \in \mathbf{U}$;
(4) $R$ = isLITERAL$(v)$ with $v \in dom(\theta)$ and $\theta(v) \in \mathbf{L}$;
(5) $R = (u = v)$ with $u, v \in dom(\theta) \cup \mathbf{UBL}$ and $\theta(u) = \theta(v)$;
(6) $R = (\neg R_1)$ with $\theta(R_1)$ is false;
(7) $R = (R_1 \vee R_2)$ with $\theta(R_1)$ is true or $\theta(R_2)$ is true;
(8) $R = (R_1 \wedge R_2)$ with $\theta(R_1)$ is true and $\theta(R_2)$ is true;
(9) $R = (x \preceq y)$ with $x, y \in dom(\theta) \cup L$ and $\theta(x) \preceq \theta(y)$;
(10) $R = p(\bar{\mathbf{z}})$ with $p(\bar{\mathbf{z}})\theta =$ true iff $p(\theta(\bar{\mathbf{z}})) =$ true, where $p$ is a built-in predicate.

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 $\mathbf{UL}$, $p$ 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.* Consider we want to know where Alain was living before 2009. This query can be expressed in the following way:

```
SELECT ?city
WHERE {(:alain :livesIn ?city):?l FILTER(before(?l, [2009]))}
```

[11] We consider a simple evaluation of filter expressions where the "error" result is ignored, see [21, Sect. 11.3] for details.

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

**Proposition 2.** *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 $\theta$ and $\theta'$ coinciding on $var(P)$.*

### 2.3 Further Extensions of AnQL

In this section we include various features from SPARQL 1.1[12] such as variable assignments, projection (i.e. sub-SELECTs), aggregates and solution modifiers to AnQL. We succinctly present both syntax and semantics of the constructs. The evaluation of a ASSIGN statement is defined as:

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

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 $f(\theta_1(\bar{\mathbf{z}}))$, which is the evaluation of the function $f(\bar{\mathbf{z}})$ with respect to a substitution $\theta_1 \in [\![P]\!]_G$.

*Example 5.* 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. □

*Remark 9.* Note that this definition is more general than "`SELECT` *expr* `AS` *?var*" project expressions in current SPARQL 1.1 [9] 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 [21, Section 9.1]. In case the variable $x$ 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 [12]. Likewise, the SQL-like statement LIMIT$(k)$ can be added straightforwardly.

We can further extend the evaluation of AnQL queries with aggregate functions

$$@ \in \{\mathsf{SUM}, \mathsf{AVG}, \mathsf{MAX}, \mathsf{MIN}, \mathsf{COUNT}, \wedge, \vee, \otimes\}$$

as follows: the evaluation of a GROUPBY statement is defined as:[13]

[12] These features are currently being defined by W3C, see [9] for the latest draft.

[13] In the expression, $@\bar{\mathbf{f}}(\bar{\mathbf{z}}) \; AS \; \bar{\alpha}$ is a concise representation of $n$ aggregations of the form $@_i f_i(\bar{\mathbf{z}}_i) \; AS \; \alpha_i$.

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

where the variables $\alpha_i \notin var(P)$, $\bar{\mathbf{z}}_i \in var(P)$ and none of the GROUPBY variables $\bar{\mathbf{w}}$ are included in the aggregation function variables $\bar{\mathbf{z}}_i$. Here, we denote by $\theta|_{\bar{\mathbf{w}}}$ the restriction of variables in $\theta$ to variables in $\bar{\mathbf{w}}$. 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:

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

*Remark 10.* 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 [9] 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 three aggregation functions, namely $\wedge, \vee$ and $\otimes$, are defined by the annotation domain and thus can be used on any annotation values.

*Remark 11.* 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 withinin any pattern. This may be viewed as syntactic sugar allowing for more concise writing than the current SPARQL 1.1 [9] draft.

*Example 6.* 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 3.** *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.

## 3 Twisting AnQL – Issues and Pitfalls

In this section we discuss some practical issues arising in formulating real-life questions in AnQL like the treatment of non-annotated queries, combination of domains in queries and some domain specific issues while highlighting problems in some related works.

### 3.1 Uniform Treatment of Annotated and Non-annotated Queries

We aim at providing a uniform treatment for queries, *i.e.*, it should be allowed to ask annotated queries against non-annotated graphs and vice-versa. There are two distinct situations where a default value must be determined, viz., in the RDF data or in SPARQL queries. The treatment of non-annotated triples in the data has been discussed in [23] and here we just use the meta-variable $\Omega_D$ to represent the default value for domain $D$. We consider a similar solution for evaluating a SPARQL query over an annotated RDFS dataset. We allow that any non-annotated triple pattern $\tau$ be considered a BAP by assigning it a default annotation. We consider that a graph pattern $P$, is in *Annotated Normal Form* (*ANF*) if it does not contain any non-annotated triple patterns. Any graph pattern $P$ can be transformed into *ANF* by replacing each non-annotated triple pattern $\tau_i \in P$ by using one of the following approaches:

1. adding a single annotation variable for each triple: $\tau_i : \lambda$, where $\lambda$ is a new annotation variable not occurring in $P$; or
2. adding a different annotation variable for each non-annotated triple: $\tau_i : \lambda_i$ s.t. each $\lambda_i$ is a new annotation variable not occurring in P and different from any other generated variable; or
3. adding the $\top$ element from the domain: $\tau_i : \top$.

In later discussions, we will use the meta-variable $\Theta_D$ to represent the default value of domain $D$ assigned to annotations in the query triples.

*Example 7.* For instance, if we again consider the query (excluding the annotation variables) and input data from Example 2, the query would look like:

```
SELECT ?p ?c
WHERE {(?p :basedNear :paris) OPTIONAL{(?p :hasCar ?c)}}
```

Now, given the 3 approaches for transforming this query into ANF we would get the following answers:

| | | |
|---|---|---|
| Approach 1 | $?p/$:alain | - |
| | $?p/$:alain | $?c/$:peugeot |
| | $?p/$:alain | $?c/$:renault |
| Approach 2 | $?p/$:alain | $?c/$:peugeot |
| | $?p/$:alain | $?c/$:renault |
| Approach 3 | $\emptyset$ | |

### 3.2 Querying Multi-dimensional Domains

Similarly to the discussion in the previous subsection, we can encounter mismatches between the Annotated RDFS dataset and the AnQL query. In case the AnQL query contains only variables for the annotations, the query can be answered on any Annotated RDFS 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 RDFS 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 8.* Assuming the following input data:

$$(:\text{alain}, :\text{livesIn}, :\text{paris}) \colon \{\text{ex.org}\}$$

When performing the following query:

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

we would interpret the data to the form:

$$(:\text{alain}, :\text{livesIn}, :\text{paris}) \colon (\{\text{ex.org}, \Omega_{\text{temporal}}\})$$

while the query would be interpreted as:

```
SELECT ?p ?c WHERE  (?p :livesIn ?c):(Θprovenance, [2009, 2010])
```

where $\Omega_{\text{temporal}}$ and $\Theta_{\text{provenance}}$ are annotations corresponding to the default values their respective domains, as discussed in Section 3.1. The semantics of combining different domains into one multi-dimensional domain has been discussed in [23].

### 3.3 Temporal Issues

Let us highlight some specific issues inherent to the temporal domain. Considering queries using Allen's temporal relations [1] (before, after, overlaps, etc.) as allowed in [24], we can pose queries like "find persons who lived in Paris before Alain". this query raises some ambiguity when considering that persons may have lived in the same city at different disjoint intervals. We can model such situations – relying on sets of temporal intervals modelling the temporal domain. Consider the following input data:

$$(:\text{betty}, :\text{livesIn}, :\text{paris}) \colon \{[1990, 1995]\}$$
$$(:\text{alain}, :\text{livesIn}, :\text{paris}) \colon \{[1980, 2000], [2002, 2010]\}$$

Tappolet and Bernstein [24] consider the latter triple as two triples with disjoint intervals as annotations. For the following query in their language $\tau$SPARQL:

```
SELECT ?p WHERE {
[?s1,?e1] ?p :livesIn :paris . [?s2,?e2] :alain :livesIn :paris .
[?s1,?e1] time:intervalBefore [?s2,?e2] }
```

we would get :betty as an answer although Alain was already living in Paris when Betty moved there. 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 :livesIn :paris):?l1 .
   (:alain :livesIn :paris):?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 [24] by use of beforeAny instead. Our report [13] provides more details on this issue.

### 3.4 Constraints vs Filters

Considering the previous section, please note that FILTER*s* do not act as *constraints* over the query. It could be expected that, given the data from the previous section, and for the following query:

```
SELECT ?l1 ?l2 WHERE {(?p :livesIn :paris):?l1 .
   (:alain :livesIn :paris):?l2 }
```

with an additional *constraint* that requires $?l1$ to be "before" $?l2$. We could expect the answer $\{?l1/[1990, 1995], ?l2/[1996, 2000]\}$ that matches the query with regards to the data 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*.

## Conclusions

Based on our previous work on Annotated RDFS [23], we presented a semantics for an extension of the SPARQL query language, AnQL, that enables querying RDF with annotations. Queries are specified with regards to a specific domain, from which we presented some of the more common ones. 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.

A prototype implementation, including the annotated RDFS inferencing and annotated SPARQL query engine is available at `http://anql.deri.org`.

**Acknowledgement.** The work presented in this report has been funded in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Líon-2) and supported by COST Action IC0801 on Agreement Technologies. We thank Jürgen Umbrich for his useful comments.

## References

1. Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM 26(11), 832–843 (1983)
2. Angles, R., Gutierrez, C.: The Expressive Power of SPARQL. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 114–129. Springer, Heidelberg (2008)
3. Brickley, D., Guha, R.V.: RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation (2004), `http://www.w3.org/TR/rdf-schema/`
4. Dividino, R.Q., Sizov, S., Staab, S., Schueler, B.: Querying for Provenance, Trust, Uncertainty and other Meta Knowledge in RDF. Journal of Web Semantics 7(3), 204–219 (2009)
5. Flouris, G., Fundulaki, I., Pediaditis, P., Theoharis, Y., Christophides, V.: Coloring RDF Triples to Capture Provenance. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 196–212. Springer, Heidelberg (2009)
6. Green, T.J., Karvounarakis, G., Tannen, V.: Provenance semirings. In: Proc. of 26th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2010), pp. 31–40 (2007)

7. Gutiérrez, C., Hurtado, C.A., Vaisman, A.A.: Introducing Time into RDF. IEEE Transactions on Knowledge and Data Engineering 19(2), 207–218 (2007)
8. Hájek, P.: Metamathematics of Fuzzy Logic. In: Trends in Logic, Kluwer, Dordrecht (1998)
9. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language. W3C Working Draft (2010), http://www.w3.org/TR/2010/WD-sparql11-query-20100601/
10. Hartig, O.: Querying Trust in RDF Data with tSPARQL. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 5–20. Springer, Heidelberg (2009)
11. Klement, E.P., Mesiar, R., Pap, E.: Triangular Norms. In: Trends in Logic, Kluwer, Dordrecht (2000)
12. Labrador, N.M., Straccia, U.: Monotonic Mappings Invariant Linearisation of Finite Posets. Technical report, Computing Research Repository (2010), Available as CoRR technical report at, http://arxiv.org/abs/1006.2679
13. Lopes, N., Lukácsy, G., Polleres, A., Straccia, U., Zimmermann, A.: A General Framework for Representing, Reasoning and Querying with Annotated Semantic Web Data. Technical report, DERI (2010), http://www.deri.ie/fileadmin/documents/DERI-TR-2010-03-29.pdf
14. Manola, F., Miller, E.: RDF Primer. W3C Recommendation (2004), http://www.w3.org/TR/rdf-primer/
15. Mazzieri, M., Dragoni, A.F.: A Fuzzy Semantics for the Resource Description Framework. In: da Costa, P.C.G., d'Amato, C., Fanizzi, N., Laskey, K.B., Laskey, K.J., Lukasiewicz, T., Nickles, M., Pool, M. (eds.) URSW 2005 - 2007. LNCS (LNAI), vol. 5327, pp. 244–261. Springer, Heidelberg (2008)
16. Muñoz, S., Pérez, J., Gutiérrez, C.: Minimal Deductive Systems for RDF. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 53–67. Springer, Heidelberg (2007)
17. Pérez, J., Arenas, M., Gutiérrez, C.: Semantics and complexity of SPARQL. ACM Transactions on Database Systems 34(3) (2009)
18. Peterson, D., Gao, S., Malhotra, A., Sperberg-McQueen, C.M., Thompson, H.S.: W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. W3C Working Draft (2009), http://www.w3.org/TR/xmlschema11-2/
19. Pugliese, A., Udrea, O., Subrahmanian, V.S.: Scaling RDF with time. In: Proc. of 17th International Conference on World Wide Web (WWW 2008), pp. 605–614 (2008)
20. Schenk, S.: On the Semantics of Trust and Caching in the Semantic Web. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 533–549. Springer, Heidelberg (2008)
21. Seaborne, A., Prud'hommeaux, E.: SPARQL Query Language for RDF. W3C Recommendation (2008), http://www.w3.org/TR/rdf-sparql-query/
22. Straccia, U.: A Minimal Deductive System for General Fuzzy RDF. In: Proc. of 3rd Int. Conference on Web Reasoning and Rule Systems (RR 2009), pp. 166–181 (2009)
23. Straccia, U., Lopes, N., Lukacsy, G., Polleres, A.: A General Framework for Representing and Reasoning with Annotated Semantic Web Data. In: Proc. of 24th AAAI Conference on Artificial Intelligence (AAAI 2010). AAAI Press, Menlo Park (2010)
24. Tappolet, J., Bernstein, A.: Applied Temporal RDF: Efficient Temporal Querying of RDF Data with SPARQL. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 308–322. Springer, Heidelberg (2009)
25. Udrea, O., Recupero, D.R., Subrahmanian, V.S.: Annotated RDF. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 487–501. Springer, Heidelberg (2006)
26. Udrea, O., Recupero, D.R., Subrahmanian, V.S.: Annotated RDF. ACM Transactions on Computational Logic 11(2), 1–41 (2010)