Incomplete Information in RDF*

Charalampos Nikolaou and Manolis Koubarakis

National and Kapodistrian University of Athens
{charnik,koubarak}@di.uoa.gr

Abstract. We extend RDF with the ability to represent property values that exist, but are unknown or partially known, using constraints. Following ideas from the incomplete information literature, we develop a semantics for this extension of RDF, called RDF$^{\mathrm{i}}$, and study SPARQL query evaluation in this framework.

1 Introduction

Incomplete information has been studied in-depth in relational databases [6,3] and knowledge representation. It is also an important issue in Semantic Web frameworks such as RDF, description logics, and OWL 2 especially given that all these systems rely on the Open World Assumption (OWA). Making the OWA means that we cannot capture negative information implicitly, i.e., if a formula ϕ is not entailed by our knowledge base, we cannot assume its negation as in the Closed World Assumption (CWA). In the context of the Web, incomplete information has recently been studied in detail for XML [2]. There have also been some recent papers in the area of Semantic Web.

[4] introduces the concept of *anonymous timestamps* in general temporal RDF graphs, i.e., graphs containing quads of the form $(s, p, o)[t]$ where t is a timestamp (a natural number) or an anonymous timestamp x stating that the triple (s, p, o) is valid in some unknown time point x. [5] extends the concept of general temporal RDF graphs of [4] so that one is allowed to express temporal constraints involving anonymous timestamps using a formula ϕ which is a conjunction of order constraints $x_1\ OP\ x_2$ where OP is an arithmetic comparison operator. [5] calls c-temporal graphs the resulting pairs (G, ϕ) where G is a general temporal RDF graph and ϕ is a conjunction of constraints. [5] defines a semantics for c-temporal graphs and studies the relevant problem of entailment.

More recently, [1] examines the question of whether SPARQL is an appropriate language for RDF given the OWA typically associated with the framework. It defines a *certain answer* semantics for SPARQL query evaluation based on well-known ideas from incomplete information research. According to this semantics, if G is an RDF graph then evaluating a SPARQL query q over G is defined as evaluating q over all graphs $H \supseteq G$ that are possible extensions of G according to the OWA, and then taking the intersection of all answers. [1] shows

* This work was supported by the European FP7 project TELEIOS (257662) and the Greek NSRF project SWeFS (180).

that if we evaluate a monotone graph pattern (e.g., one using only the operators AND, UNION, and FILTER) using the well-known W3C semantics, we get the same result we would get if we used the certain answer semantics. The converse also holds, thus monotone SPARQL graph patterns are exactly the ones that have this nice property. However, OPTIONAL (OPT) is not a monotone operator and the two semantics do not coincide for it. [1] defines the notion of weak monotonicity that appears to capture the intuition behind OPT and shows that the fragment of SPARQL consisting of the well-designed graph patterns defined originally in [16] is weakly monotone.

1.1 Contributions

In this paper we continue the line of research started by [4,5,1] and study in a general way an important kind of incomplete information that has so far been ignored in the context of RDF. Our contributions are the following.

First, we extend RDF with the ability to define a new kind of literals for each datatype. These literals will be called *e-literals* ("e" comes from the word "existential") and can be used to represent values of properties that *exist but are unknown or partially known*. In the proposed extension of RDF, called RDF$^{\text{i}}$ (where "i" stands for "incomplete"), e-literals are allowed to appear only in the object position of triples. RDF$^{\text{i}}$ allows partial information regarding property values represented by e-literals to be expressed by a quantifier-free formula of a first-order *constraint language* $\mathcal{L}$. Thus, RDF$^{\text{i}}$ extends the concept of an RDF graph to the concept of an RDF$^{\text{i}}$ *database* which is a pair (G, ϕ) where G is an RDF graph possibly containing triples with e-literals in their object positions, and ϕ is a quantifier-free formula of $\mathcal{L}$. [12] motivates the need for introducing RDF$^{\text{i}}$ by concentrating on the representation of incomplete spatial knowledge.

Following ideas from the incomplete information literature [6,3], we develop a semantics for RDF$^{\text{i}}$ databases and SPARQL query evaluation. The semantics defines the set of possible RDF graphs corresponding to an RDF$^{\text{i}}$ database and the fundamental concept of certain answer for SPARQL query evaluation over an RDF$^{\text{i}}$ database. We transfer the well-known concept of *representation system* from [6] to the case of RDF$^{\text{i}}$, and show that CONSTRUCT queries without blank nodes in their templates and using only operators AND, UNION, and FILTER or the restricted fragment of graph patterns corresponding to the well-designed patterns of [1] can be used to define a representation system for RDF$^{\text{i}}$. On our way to show these results, we also show some interesting monotonicity properties for CONSTRUCT queries.

We define the fundamental concept of certain answer to SPARQL queries over RDF$^{\text{i}}$ databases and present an algorithm for its computation. Finally, we present preliminary complexity results for computing certain answers by considering equality, temporal, and spatial constraint languages $\mathcal{L}$ and the class of CONSTRUCT queries of our representation system. Our results show that the data complexity of evaluating a query of this class over RDF$^{\text{i}}$ databases increases from LOGSPACE (the upper bound for evaluating queries from this class over RDF graphs [16]) to coNP-complete for the case of equality and temporal

constraints. This result is in line with similar complexity results for querying incomplete information in relational databases [3,7]. The same coNP-completeness bound is shown for the case of spatial constraints on rectangles in $\mathbb{Q}^2$ [7]. For topological constraints over more general spatial regions (regular closed subsets of $\mathbb{Q}^2$), the best upper bound that we can show is EXPTIME. It is an open problem whether a better complexity bound can be achieved in this case.

The paper is organized as follows. Section 2 presents the properties that we expect constraint languages to have so that they can be used in RDF$^{\text{i}}$, and defines some useful constraint languages. Section 3 introduces RDF$^{\text{i}}$, while Sections 4 and 5 define its semantics and the evaluation of SPARQL queries over RDF$^{\text{i}}$ databases. Section 6 presents fragments of SPARQL that can be used to define a representation system for RDF$^{\text{i}}$. Section 7 gives an algorithm for computing the certain answer for SPARQL queries over RDF$^{\text{i}}$ databases and presents complexity results. Sections 8 and 9 discuss related and future work. Proofs of results and some technical details are omitted due to space; they may be found in the full version of the paper [11].

2 Constraint Languages

We consider many-sorted first-order languages, structures, and theories. Every language $\mathcal{L}$ is interpreted over a *fixed* structure, called the *intended structure*, which is denoted by $\mathbf{M}_{\mathcal{L}}$. If $\mathbf{M}_{\mathcal{L}}$ is a structure then $Th(\mathbf{M}_{\mathcal{L}})$ denotes the theory of $\mathbf{M}_{\mathcal{L}}$. For every language $\mathcal{L}$, we distinguish a class of quantifier free formulae called $\mathcal{L}$-*constraints*. The atomic formulae of $\mathcal{L}$ are included in the class of $\mathcal{L}$-constraints. There are also two distinguished $\mathcal{L}$-constraints *true* and *false* with obvious semantics. Every first-order language $\mathcal{L}$ we consider has a distinguished equality predicate, denoted by EQ, with the standard semantics. The class of $\mathcal{L}$-constraints is assumed to: *a*) contain all formulae t_1 EQ t_2 where t_1, t_2 are terms of $\mathcal{L}$, and *b*) be *weakly closed under negation*, i.e., the negation of every $\mathcal{L}$-constraint is equivalent to a disjunction of $\mathcal{L}$-constraints.

The full version of the paper [11] defines formally various constraint languages that allow us to explore the scope of modeling possibilities that RDF$^{\text{i}}$ offers, especially in temporal and spatial domains. These languages are ECL, diPCL, dePCL, PCL, TCL and RCL and are only briefly defined here. ECL is the first order language of equality constraints of the form x EQ y and x EQ c (where x, y are variables and c is a constant) interpreted over an infinite domain [3]. ECL allows RDF$^{\text{i}}$ to represent "marked nulls" as in relational databases [6]. Languages diPCL, dePCL are the first order languages of temporal difference constraints of the form $x - y \leq c$ interpreted over the integers (diPCL) or the rationals (dePCL), and allow RDF$^{\text{i}}$ to represent incomplete temporal information as in [4,5]. PCL, TCL and RCL are spatial constraint languages and are defined below.

Language *PCL* (*P*olygon *C*onstraint *L*anguage) allows us to represent topological properties of non-empty, regular closed subsets of $\mathbb{Q}^2$ that are polygons. PCL is a first-order language with the following 6 binary predicate symbols corresponding to the topological relations of RCC-8 calculus [17]:

DC, EC, PO, EQ, TPP, and NTPP. The constant symbols of PCL represent polygons in $\mathbb{Q}^2$. We write these constants as conjunctions of linear constraints in quotes (half-space representation of the convex polygon). The terms and atomic formulae of PCL are defined as follows. Constants and variables are *terms*. An *atomic formula* of PCL (PCL-constraint) is a formula of the form $t_1\ R\ t_2$ where t_1, t_2 are terms and R is one of the above predicates. The intended structure for PCL, denoted by $\mathbf{M}_{PCL}$, has the set of non-empty, regular closed subsets of $\mathbb{Q}^2$ as its domain. $\mathbf{M}_{PCL}$ interprets each constant symbol by the corresponding polygon in $\mathbb{Q}^2$ and each of the predicate symbols by the corresponding topological relation of RCC-8 [17].

Language TCL (*T*opological *C*onstraint *L*anguage) is defined like PCL, but now terms can only be variables. Language RCL (*R*ectangle *C*onstraint *L*anguage) is a first-order constraint language that represents information about *rectangles* in $\mathbb{Q}^2$ using rational constants and order or difference constraints ($x - y \leq c$) on the vertices of rectangles. RCL has essentially the same expressive power with dePCL, but it's been carefully crafted for rectangles.

3 The RDF$^{\mathrm{i}}$ Framework

As in theoretical treatments of RDF [16], we assume the existence of pairwise-disjoint, countably infinite sets I, B, and L that contain IRIs, blank nodes, and literals respectively. We also assume the existence of a datatype map M and distinguish a set of datatypes A from M for which e-literals are allowed. Finally, we assume the existence of a many-sorted first order constraint language $\mathcal{L}$ with the properties discussed in Section 2. $\mathcal{L}$ is related to the datatype map M in the following way: *a*) The set of sorts of $\mathcal{L}$ is the set of datatypes A of M. *b*) The set of constants of $\mathcal{L}$ is the union of the lexical spaces of the datatypes in A. *c*) $\mathbf{M}_{\mathcal{L}}$ interprets every constant c of $\mathcal{L}$ with sort d by its corresponding value given by the lexical-to-value mapping of the datatype d in A.

The set of constants of $\mathcal{L}$ (equivalently: the set of literals of the datatypes in A) are denoted by C. We also assume the existence of a countably infinite set of e-literals for each datatype in A and use U to denote the union of these sets. By convention, the identifiers of e-literals start with an underscore. C and U are assumed to be disjoint from each other and from I, B, and L. The set of RDF$^{\mathrm{i}}$ *terms*, denoted by T, can now be defined as the union $I \cup B \cup L \cup C \cup U$. In the rest of our examples we will assume that $\mathcal{L}$ is PCL, so C is the set of all polygons in $\mathbb{Q}^2$ written as linear constraints.

We now define the basic concepts of RDF$^{\mathrm{i}}$.

Definition 1. An *e-triple* is an element of the set $(I \cup B) \times I \times T$. If (s, p, o) is an e-triple, s will be called the subject, p the predicate, and o the object of the triple. A *conditional triple* is a pair (t, θ) where t is an e-triple and θ is a conjunction of $\mathcal{L}$-constraints. If (t, θ) is a conditional triple, θ will be called the condition of the triple. A *global constraint* is a Boolean combination of $\mathcal{L}$-constraints. A *conditional graph* is a set of conditional triples. An RDF$^{\mathrm{i}}$ *database* D is a pair $D = (G, \phi)$ where G is a conditional graph and ϕ a global constraint.

Example 1. The following pair is an RDF$^{\text{i}}$ database.

```
({((hotspot1, type, Hotspot), true),        ((fire1, type, Fire), true),
  ((hotspot1, correspondsTo, fire1), true),  ((fire1, occuredIn, _R1), true) },
_R1 NTPP "x ≥ 6 ∧ x ≤ 23 ∧ y ≥ 8 ∧ y ≤ 19")
```

The example comes from a real application [12] and it is about hotspots captured by satellite images. The database mentions a hotspot which is located in a region that is inside but does not intersect with the boundary of rectangle defined by the points $(6, 8)$ and $(23, 19)$. E-literal _R1 is used to represent the region the exact coordinates of which are unknown. The possible values for the e-literal are further constrained by the global constraint.

4 Semantics of RDF$^{\text{i}}$

The semantics of RDF$^{\text{i}}$ are inspired by [6]. An RDF$^{\text{i}}$ database $D = (G, \phi)$ corresponds to a set of possible RDF graphs each one representing a possible state of the real world. This set of possible graphs captures completely the semantics of an RDF$^{\text{i}}$ database. The global constraint ϕ determines the number of possible RDF graphs corresponding to D; there is one RDF graph for each solution of ϕ obtained by considering the e-literals of ϕ as variables and solving constraint ϕ.

Example 2. Let $D = (G, \phi)$ be the RDF$^{\text{i}}$ database given in Example 1. The same knowledge can be represented by an (infinite) set of possible RDF graphs, one for each rectangle inside P. One such graph is the following:

$G_1 =$ `{(hotspot1,type,Hotspot), (fire1,type,Fire), (hotspot1,correspondsTo,fire1),`
`(fire1,occuredIn,"x ≥ 11 ∧ x ≤ 15 ∧ y ≥ 13 ∧ y ≤ 15")}`

In order to be able to go from RDF$^{\text{i}}$ databases to the equivalent set of possible RDF graphs, the notion of *valuation* is needed. Informally, a valuation maps an e-literal to a specific constant from C. We denote by $v(t)$ the application of valuation v to an e-triple t. $v(t)$ is obtained from t by replacing any e-literal $\_l$ appearing in t by $v(\_l)$ and leaving all other terms the same. If θ is a formula of $\mathcal{L}$ (e.g., the condition of a conditional triple or the global constraint of a database) then $v(\theta)$ denotes the application of v to formula θ. The expression $v(\theta)$ is obtained from θ by replacing all e-literals $\_l$ of θ by $v(\_l)$.

Next, we give the definition of applying a valuation to a conditional graph.

Definition 2. Let G be a conditional graph and v a valuation. Then $v(G)$ denotes the RDF graph $\{v(t) \mid (t, \theta) \in G \text{ and } \mathbf{M}_{\mathcal{L}} \models v(\theta)\}$.

The set of valuations that satisfy the global constraint of an RDF$^{\text{i}}$ database determines the set of possible RDF graphs that correspond to it. This set of graphs is denoted using the function *Rep* as it is traditional in incomplete relational databases.

Definition 3. Let $D = (G, \phi)$ be an RDF$^{\text{i}}$ database. The set of RDF graphs corresponding to D is the following:

$Rep(D) = \{H \mid \text{there exists a valuation } v \text{ such that } \mathbf{M}_{\mathcal{L}} \models v(\phi) \text{ and } H \supseteq v(G)\}$

In incomplete relational databases [6], *Rep* is a *semantic* function: it is used to map a table (a syntactic construct) to a set of relational instances (i.e., a set of possible words, a semantic construct). According to the well-known distinction between model theoretic and proof theoretic approaches to relational databases, *Rep* and the approaches based on it [6,3] belong to the model theoretic camp. However, the use of function *Rep* in the above definition is different. *Rep* takes an RDF$^{\text{i}}$ database (a syntactic construct) and maps it to a set of possible RDF graphs (a syntactic construct again). This set of possible graphs can then be mapped to a set of possible worlds using the well-known RDF model theory. This is a deliberate choice in our work since we want to explore which well-known tools from incomplete relational databases carry over to the RDF framework.

Notice that the definition of *Rep* above uses the containment relation instead of equality. The reason for this is to capture the OWA that the RDF model makes. By using the containment relation, $Rep(D)$ includes all graphs H containing at least the triples of $v(G)$. In this respect, we follow the approach of [1, Section 3], where the question of whether SPARQL is a good language for RDF is examined in the light of the fact that RDF adopts the OWA. To account for this, an RDF graph G is seen to correspond to a set of possible RDF graphs H such that $G \subseteq H$ (in the sense of the OWA: all triples in G also hold in H). The above definition takes this concept of [1] to its rightful destination: the full treatment of incomplete information in RDF. As we have already noted in the introduction, the kinds of incomplete information we study here for RDF have not been studied in [1]; only the issue of OWA has been explored there.

Notation 1 Let $\mathcal{G}$ be a set of RDF graphs and q a SPARQL query. The expression $\bigcap \mathcal{G}$ will denote the set $\bigcap_{G \in \mathcal{G}} G$. The expression $[\![q]\!]_{\mathcal{G}}$, which extends the notation of [16] to the case of sets of RDF graphs, will denote the element-wise evaluation of q over $\mathcal{G}$, that is, $[\![q]\!]_{\mathcal{G}} = \{[\![q]\!]_G \mid G \in \mathcal{G}\}$.

Given the semantics of an RDF$^{\text{i}}$ database as a set of possible RDF graphs, the next definition of *certain answer* extends the corresponding definition of Section 3.1 of [1] by applying it to a more general incomplete information setting.

Definition 4. Let q be a query and $\mathcal{G}$ a set of RDF graphs. The *certain answer* to q over $\mathcal{G}$ is the set $\bigcap [\![q]\!]_{\mathcal{G}}$.

Example 3. Let us consider the following query over the database of Example 1: "Find all fires that have occurred in a region which is a non-tangential proper part of the rectangle defined by the points $(2, 4)$ and $(28, 22)$". The certain answer to this query is the set of mappings $\{\{?\text{F} \rightarrow \text{fire1}\}\}$.

5 Evaluating SPARQL on RDF$^{\text{i}}$ Databases

Let us now discuss how to evaluate SPARQL queries on RDF$^{\text{i}}$ databases. Due to the presence of e-literals, query evaluation now becomes more complicated and is

similar to query evaluation for conditional tables [6,3]. We use set semantics for query evaluation by extending the SPARQL query evaluation approach of [16].

We assume the existence of the following disjoint sets of variables: (i) the set of *normal query variables* V_n that range over IRIs, blank nodes, or RDF literals, and (ii) the set of *special query variables* V_s that range over literals from the set C or e-literals from the set U. We use V to denote the set of all variables $V_n \cup V_s$. Set V is assumed to be disjoint from the set of terms T we defined in Section 3.

We now define the concepts of e-mapping and conditional mapping that are extensions of the standard mappings of [16]. An *e-mapping* ν is a partial function $\nu : V \rightarrow T$ such that $\nu(x) \in I \cup B \cup L$ if $x \in V_n$ and $\nu(x) \in C \cup U$ if $x \in V_s$. A *conditional mapping* μ is a pair (ν, θ) where ν is an e-mapping and θ is a conjunction of $\mathcal{L}$-constraints. The notions of domain and restriction of an e-mapping as well as the notion of compatibility of two e-mappings are defined as for mappings in the obvious way [16]. The *domain* of a conditional mapping $\mu = (\nu, \theta)$, denoted by $dom(\mu)$, is the domain of ν, i.e., the subset of V where the partial function ν is defined. Let $\mu = (\nu, \theta)$ be a conditional mapping with domain S and $W \subseteq S$. The *restriction* of the mapping μ to W, denoted by $\mu_{|W}$, is the mapping $(\nu_{|W}, \theta)$ where $\nu_{|W}$ is the restriction of mapping ν to W.

A *triple pattern* is an element of the set $(I \cup V) \times (I \cup V) \times (I \cup L \cup C \cup U \cup V)$. We do not allow blank nodes to appear in a triple pattern as in standard SPARQL since such blank nodes can equivalently be substituted by new query variables. If p is a triple pattern, $var(p)$ denotes the variables appearing in p. A conditional mapping can be applied to a triple pattern. If $\mu = (\nu, \theta)$ is a conditional mapping and p a triple pattern such that $var(p) \subseteq dom(\mu)$, then $\mu(p)$ is the triple obtained from p by replacing each variable $x \in var(p)$ by $\nu(x)$.

We now introduce the notion of compatible conditional mappings by generalizing the relevant notions of [16]. Two conditional mappings $\mu_1 = (\nu_1, \theta_1)$ and $\mu_2 = (\nu_2, \theta_2)$ are *compatible* if the e-mappings ν_1 and ν_2 are compatible, i.e., for all $x \in dom(\mu_1) \cap dom(\mu_2)$, we have $\nu_1(x) = \nu_2(x)$. To take into account e-literals, we also need to define another notion of compatibility of two conditional mappings. Two conditional mappings $\mu_1 = (\nu_1, \theta_1)$ and $\mu_2 = (\nu_2, \theta_2)$ are *possibly compatible* if for all $x \in dom(\mu_1) \cap dom(\mu_2)$, we have $\nu_1(x) = \nu_2(x)$ or at least one of $\nu_1(x), \nu_2(x)$ where $x \in V_s$ is an e-literal from U. If two conditional mappings are possibly compatible, then we can define their join as follows.

Definition 5. Let $\mu_1 = (\nu_1, \theta_1)$ and $\mu_2 = (\nu_2, \theta_2)$ be possibly compatible conditional mappings. The *join* $\mu_1 \bowtie \mu_2$ is a new conditional mapping (ν_3, θ_3) where:

i. $\nu_3(x) = \nu_1(x) = \nu_2(x)$ for each $x \in dom(\mu_1) \cap dom(\mu_2)$ s.t. $\nu_1(x) = \nu_2(x)$.
ii. $\nu_3(x) = \nu_1(x)$ for each $x \in dom(\mu_1) \cap dom(\mu_2)$ s.t. $\nu_1(x)$ is an e-literal and $\nu_2(x)$ is a literal from C.
iii. $\nu_3(x) = \nu_2(x)$ for each $x \in dom(\mu_1) \cap dom(\mu_2)$ s.t. $\nu_2(x)$ is an e-literal and $\nu_1(x)$ is a literal from C.
iv. $\nu_3(x) = \nu_1(x)$ for $x \in dom(\mu_1) \cap dom(\mu_2)$ s.t. both $\nu_1(x)$ and $\nu_2(x)$ are e-literals.
v. $\nu_3(x) = \nu_1(x)$ for $x \in dom(\mu_1) \setminus dom(\mu_2)$.
vi. $\nu_3(x) = \nu_2(x)$ for $x \in dom(\mu_2) \setminus dom(\mu_1)$.

vii. θ_3 is $\theta_1 \wedge \theta_2 \wedge \xi_1 \wedge \xi_2 \wedge \xi_3$ where:
- ξ_1 is $\bigwedge_i \_v_i$ EQ $\_t_i$ where the $\_v_i$'s and $\_t_i$'s are all the pairs of e-literals $\nu_1(x)$, $\nu_2(x)$ from Case (iv). If there are no such pairs, ξ_1 is *true*.
- ξ_2 is $\bigwedge_i \_w_i$ EQ l_i where the $\_w_i$'s and l_i's are all the pairs of e-literals $\nu_1(x)$ and literals $\nu_2(x)$ from the set C from Case (ii). If there are no such pairs, ξ_2 is *true*.
- ξ_3 is $\bigwedge_i \_w_i$ EQ l_i where the $\_w_i$'s and l_i's are all the pairs of e-literals $\nu_2(x)$ and literals $\nu_1(x)$ from the set C from Case (iii). If there are no such pairs, ξ_3 is *true*.

The predicate EQ used in the above definition is the equality predicate of $\mathcal{L}$.

For two sets of conditional mappings Ω_1 and Ω_2, the operations of join, union, and left-outer join can be defined similarly to [16] and can be found in the full version of the paper [11]. Given the previous operations on sets of mappings, graph pattern evaluation in RDF$^{\text{i}}$ can now be defined exactly as in standard SPARQL for RDF graphs [16] except for the case of evaluating a triple pattern and a FILTER graph pattern which are defined next. The other cases (AND, UNION, and OPT graph patterns) may be found in the full version of the paper [11].

Definition 6. Let $D = (G, \phi)$ be an RDF$^{\text{i}}$ database. Evaluating a triple pattern $P = (s, p, o)$ over database D is denoted by $[\![P]\!]_D$ and is defined as follows:
If o is a literal from the set C then

$$[\![P]\!]_D = \{\mu = (\nu, \theta) \mid \text{dom}(\mu) = var(P) \text{ and } (\mu(P), \theta) \in G\} \cup$$
$$\{\mu = (\nu, (\_l \text{ EQ } o) \wedge \theta) \mid \text{dom}(\mu) = var(P), ((\nu(s), \nu(p), \_l), \theta) \in G,\ \_l \in U\}$$

else

$$[\![P]\!]_D = \{\mu = (\nu, \theta) \mid \text{dom}(\mu) = var(P),\ (\mu(P), \theta) \in G\}$$

In the above definition the "else" part is to accommodate the case in which evaluation can be done as in standard SPARQL. The "if" part accommodates the case in which the triple pattern involves a literal o from the set C. Here, there are two alternatives: the graph contains a conditional triple matching with every component of the triple pattern (i.e., a triple which has o in the object position) or it contains a conditional triple with an e-literal $\_l$ from U in the object position. We catch a possible match for the second case by adding in the condition of the mapping the constraint that restricts the value of e-literal $\_l$ to be equal to the literal o of the triple pattern (i.e., the constraint $\_l$ EQ o).

It has been noted in [16] that the OPT operator of SPARQL can be used to express difference in SPARQL. For data models that make the OWA, such an operator is unnatural since negative information cannot be expressed. However, we deliberately include operator OPT (see [11]) because if it is combined with operators AND and FILTER under certain syntactic restrictions (*well-designed graph patterns*), the resulting graph patterns cannot express a difference operator anymore [1].

Definition 7. Given an RDF$^{\text{i}}$ database $D = (G, \phi)$, a graph pattern P and a conjunction of $\mathcal{L}$-constraints R, we have:

$$[\![P \text{ FILTER } R]\!]_D = \{\mu' = (\nu, \theta') \mid \mu = (\nu, \theta) \in [\![P]\!]_D \text{ and } \theta' \text{ is } \theta \wedge \nu(R)\ \}$$

In the above definition, $\nu(R)$ denotes the application of e-mapping ν to condition R, i.e., the conjunction of $\mathcal{L}$-constraints obtained from R when each variable x of R which also belongs to $dom(\nu)$ is substituted by $\nu(x)$. The extension of FILTER to the case that R is a Boolean combination of $\mathcal{L}$-constraints or contains built-in conditions of standard SPARQL [16] is easy to define and is omitted.

We now define the SELECT and CONSTRUCT query forms of SPARQL. A SELECT query q is a pair (W, P) where W is a set of variables from the set V and P is a graph pattern. The *answer* to q over an RDF$^{\text{i}}$ database $D = (G, \phi)$ (in symbols $[\![q]\!]_D$) is the set of conditional mappings $\{\mu_{|W} \mid \mu \in [\![P]\!]_D\}$.

A *template* is a finite subset of set $(T \cup V) \times (I \cup V) \times (T \cup V)$. A CONSTRUCT query is a pair (E, P) where E is a template and P a graph pattern. We denote by $var(E)$ the set of variables appearing in the elements of E and by $\mu(E)$ the application of conditional mapping μ to template E. $\mu(E)$ is obtained from E by replacing in E every variable x of $var(E) \cap dom(\mu)$ by $\mu(x)$. Next we define the concept of answer to a CONSTRUCT query. The definition extends the one of [15] to the case of RDF$^{\text{i}}$.

Definition 8. Let $q = (E, P)$ be a CONSTRUCT query, $D = (G, \phi)$ an RDF$^{\text{i}}$ database and $F = \{f_\mu \mid \mu \in [\![P]\!]_D\}$ a fixed set of renaming functions. The *answer* to q over D (in symbols $[\![q]\!]_D$) is the RDF$^{\text{i}}$ database $D' = (G', \phi)$ where

$$G' = \bigcup_{\mu=(\nu,\theta) \in [\![P]\!]_D} \{(t, \theta) \mid t \in (\mu(f_\mu(E)) \cap ((I \cup B) \times I \times T))\}.$$

Example 4. Let us consider the following query over the database of Example 1: "Find all fires that have occurred in a region which is a non-tangential proper part of rectangle defined by the points $(10, 12)$ and $(21, 17)$". This query can be expressed using the CONSTRUCT query form as follows:

$(\{(?\text{F}, \text{type}, \text{Fire})\}, (?\text{F}, \text{type}, \text{Fire})\ \text{AND}\ (?\text{F}, \text{occuredIn}, ?\text{R})$
$\quad \text{FILTER}\ (?\text{R NTPP "x} \geq 10 \wedge \text{x} \leq 21 \wedge \text{y} \geq 12 \wedge \text{y} \leq 17\text{"}))$

The answer to the above query is is the following RDF$^{\text{i}}$ database:

`( { ((fire1, type, Fire),` _R1 NTPP "x $\geq 10 \wedge$ x $\leq 21 \wedge$ y $\geq 12 \wedge$ y ≤ 17") },
_R1 NTPP "x $\geq 6 \wedge$ x $\leq 23 \wedge$ y $\geq 8 \wedge$ y ≤ 19")

6 Representation Systems for RDF$^{\text{i}}$

Let us now recall the semantics of RDF$^{\text{i}}$ as given by *Rep*. $Rep(D)$ is the set of possible RDF graphs corresponding to an RDF$^{\text{i}}$ database D. Clearly, if we were to evaluate a query q over D, we could use the semantics of RDF$^{\text{i}}$ and evaluate q over any RDF graph of $Rep(D)$ as $[\![q]\!]_{Rep(D)} = \{[\![q]\!]_G \mid G \in Rep(D)\}$. However,

this is not the best answer we wish to have in terms of representation; we queried an RDF$^{\text{i}}$ database and got an answer which is a set of RDF graphs. Any well-defined query language should have the *closure* property, i.e., the output (answer) should be of the same type as the input. Ideally, we would like to have an RDF$^{\text{i}}$ database as the output. Thus, we are interested in finding an RDF$^{\text{i}}$ database $[\![q]\!]_D$ representing the answer $[\![q]\!]_{Rep(D)}$. This requirement is translated to formula:

$$Rep([\![q]\!]_D) = [\![q]\!]_{Rep(D)} \tag{1}$$

Formula (1) allows us to compute the answer to any query over an RDF$^{\text{i}}$ database in a consistent way with respect to the semantics of RDF$^{\text{i}}$ without applying the query on all possible RDF graphs. $[\![q]\!]_D$ can be computed using the algebra of Section 5 above. But can the algebra of Section 5 compute always such a database $[\![q]\!]_D$ representing $[\![q]\!]_{Rep(D)}$? In other words, can we prove (1) for all SPARQL queries considered in Section 5? The answer is *no* in general.

Example 5. Consider the RDF$^{\text{i}}$ database $D = (G, \phi)$, where $G = \{((s, p, o), true)\}$ and $\phi = true$, i.e., D contains the single triple (s, p, o) where $s, p, o \in I$. Consider now a CONSTRUCT query q over D that selects all triples having s as the subject. The algebraic version of query q would be $(\{(s, ?p, ?o)\}, (s, ?p, ?o))$ and evaluated as $[\![q]\!]_D$ using Definition 8. Then, the triple (s, p, o) and nothing else is in the resulting database $[\![q]\!]_D$. However, equation (1) is not satisfied, since for instance (c, d, e) occurs in some $g \in Rep([\![q]\!]_D)$ according to the definition of *Rep*, whereas $(c, d, e) \notin g$ for all $g \in [\![q]\!]_{Rep(D)}$.

The above counterexample to (1) exploits only the fact that RDF makes the OWA. In other words, the counterexample would hold for any approach to incomplete information in RDF which respects the OWA. Thus, unless the CWA is adopted, which we do not want to do since we are in the realm of RDF, condition (1) has to be relaxed[1]. In the rest of this section we follow the literature of incomplete information [6,3] and show how (1) can be weakened. The key concept for achieving this is that of certain answer. Given a fixed fragment of SPARQL $\mathcal{Q}$, two RDF$^{\text{i}}$ databases cannot be distinguished by $\mathcal{Q}$ if they give the same certain answer to every query in $\mathcal{Q}$. The next definition formalizes this fact using the concept of $\mathcal{Q}$-equivalence.

Definition 9. Let $\mathcal{Q}$ be a fragment of SPARQL, and $\mathcal{G}$, $\mathcal{H}$ two sets of RDF graphs. $\mathcal{G}$, $\mathcal{H}$ are called $\mathcal{Q}$-*equivalent* (denoted by $\mathcal{G} \equiv_{\mathcal{Q}} \mathcal{H}$) if they give the same certain answer to every query in the language, i.e., $\bigcap [\![q]\!]_{\mathcal{G}} = \bigcap [\![q]\!]_{\mathcal{H}}$ for all $q \in \mathcal{Q}$.

Next we define the notion of *representation system* which gives a formal characterization of the correctness of computing the answer to a query directly on an RDF$^{\text{i}}$ database instead of using the set of possible graphs given by *Rep*. The definition of representation system corresponds to the notion of *weak query system* defined for incomplete relational databases [3].

[1] If the CWA is adopted, we can prove (1) using similar techniques to the ones that enable us to prove Theorem 1 below.

Definition 10. Let $\mathcal{D}$ be the set of all RDF$^{\text{i}}$ databases, $\mathcal{G}$ the set of all RDF graphs, $Rep : \mathcal{D} \to \mathcal{G}$ a function determining the set of possible RDF graphs corresponding to an RDF$^{\text{i}}$ database, and $\mathcal{Q}$ a fragment of SPARQL. The triple $\langle \mathcal{D}, Rep, \mathcal{Q} \rangle$ is a *representation system* if for all $D \in \mathcal{D}$ and all $q \in \mathcal{Q}$, there exists an RDF$^{\text{i}}$ database $[\![q]\!]_D \in \mathcal{D}$ s.t. condition $Rep([\![q]\!]_D) \equiv_{\mathcal{Q}} [\![q]\!]_{Rep(D)}$ is satisfied.

The next step towards the development of a representation system for RDF$^{\text{i}}$ and SPARQL is to introduce various fragments of SPARQL that we will consider and define the notions of *monotonicity* and *coinitiality* as it is done in [6] for the relational case. As in Section 5, our only addition to standard SPARQL is the extension of FILTERs with another kind of conditions that are constraints of $\mathcal{L}$. We also consider the fragment of SPARQL graph patterns known as *well-designed*. Well-designed graph patterns form a practical fragment of SPARQL graph patterns that include the OPT operator. It has been showed in [16,1] that they have nice properties, such as lower combined complexity of query evaluation than in the general case, a normal form useful for optimization, and they are also weakly monotone. [11] contains formal definitions and relevant background results for well-designed graph patterns.

Notation 2 We denote by $\mathcal{Q}^C_{\mathcal{F}}$ (resp. $\mathcal{Q}^S_{\mathcal{F}}$) the set of all CONSTRUCT (resp. SELECT) queries consisting of triple patterns, and graph pattern expressions from class $\mathcal{F}$. We also denote by $\mathcal{Q}^C_{WD}$ (resp. $\mathcal{Q}^S_{WD}$) the set of all CONSTRUCT (resp. SELECT) queries consisting of well-designed graph patterns. Last, we denote by $\mathcal{Q}^{C'}_{\mathcal{F}}$ all CONSTRUCT queries without blank nodes in their templates.

The following definition introduces the concept of monotone fragments of SPARQL applied to RDF graphs.

Definition 11. A fragment $\mathcal{Q}$ of SPARQL is *monotone* if for every $q \in \mathcal{Q}$ and RDF graphs G and H such that $G \subseteq H$, it is $[\![q]\!]_G \subseteq [\![q]\!]_H$.

Proposition 1. The following results hold with respect to the monotonicity of SPARQL: *a*) Language $\mathcal{Q}^S_{AUF}$ is monotone. *b*) The presence of OPT or CONSTRUCT makes a fragment of SPARQL not monotone. *c*) Language $\mathcal{Q}^{C'}_{AUF}$ is monotone. *d*) Language $\mathcal{Q}^{C'}_{WD}$ is monotone.

Parts $a) - c)$ of the above proposition are trivial extensions of relevant results in [1]. However, part *d*) is an interesting result showing that the weak monotonicity property of well-designed graph patterns suffices to get a monotone fragment of SPARQL containing the OPT operator, i.e., the class of CONSTRUCT queries without blank nodes in their templates. This is a result that cannot be established for the case of SELECT queries and with this respect CONSTRUCT queries deserve closer attention. Monotonicity is a sufficient property for establishing our results about representation systems. Thus, in the following, we focus on the monotone query languages $\mathcal{Q}^{C'}_{AUF}$ and $\mathcal{Q}^{C'}_{WD}$.

Definition 12. Let $\mathcal{G}$ and $\mathcal{H}$ be sets of RDF graphs. We say that $\mathcal{G}$ and $\mathcal{H}$ are *coinitial*, denoted by $\mathcal{G} \approx \mathcal{H}$, if for any $G \in \mathcal{G}$ there exists $H \in \mathcal{H}$ such that $H \subseteq G$, and for any $H \in \mathcal{H}$ there exists $G \in \mathcal{G}$ such that $G \subseteq H$.

A direct consequence of the definition of coinitial sets is that they have the same greatest lower-bound elements with respect to the subset relation.

Proposition 2. Let $\mathcal{Q}$ be a monotone fragment of SPARQL and $\mathcal{G}$ and $\mathcal{H}$ sets of RDF graphs. If $\mathcal{G} \approx \mathcal{H}$ then, for any $q \in \mathcal{Q}$, it holds that $[\![q]\!]_{\mathcal{G}} \approx [\![q]\!]_{\mathcal{H}}$.

Lemma 1. Let $\mathcal{G}$ and $\mathcal{H}$ be sets of RDF graphs. If $\mathcal{G}$ and $\mathcal{H}$ are coinitial then $\mathcal{G} \equiv_{\mathcal{Q}^{C'}_{AUF}} \mathcal{H}$.

We will now present our main theorem which characterizes the evaluation of monotone $\mathcal{Q}^{C'}_{AUF}$ and $\mathcal{Q}^{C'}_{WD}$ queries.

Theorem 1. $\langle \mathcal{D}, Rep, \mathcal{Q}^{C'}_{AUF} \rangle$ and $\langle \mathcal{D}, Rep, \mathcal{Q}^{C'}_{WD} \rangle$ are representation systems.

Since SELECT queries in SPARQL take as input an RDF graph but return a set of mappings (i.e., we do not have closure), it is not clear how to include them in the developed concept of a representation system.

7 Certain Answer Computation

This section studies how the certain answer to a SPARQL query q over an RDF$^{\text{i}}$ database D can be computed, i.e., how to compute $\bigcap [\![q]\!]_{Rep(D)}$. Having Theorem 1, it is easy to compute the certain answer to a query in the fragment of SPARQL $\mathcal{Q}^{C'}_{AUF}$ or $\mathcal{Q}^{C'}_{WD}$. Since $\langle \mathcal{D}, Rep, \mathcal{Q}^{C'}_{AUF} \rangle$ and $\langle \mathcal{D}, Rep, \mathcal{Q}^{C'}_{WD} \rangle$ are representation systems, we can apply Definition 9 for the identity query to get $\bigcap [\![q]\!]_{Rep(D)} = \bigcap Rep([\![q]\!]_D)$ for all q and D. Thus, we can equivalently compute $\bigcap Rep([\![q]\!]_D)$ where $[\![q]\!]_D$ can be computed using the algebraic operations of Section 5. Before presenting the algorithm for certain answer computation, we introduce some auxiliary constructs.

Definition 13. Let $D = (G, \phi)$ be an RDF$^{\text{i}}$ database. The *EQ-completed* form of D is the RDF$^{\text{i}}$ database $D^{\text{EQ}} = (G^{\text{EQ}}, \phi)$ where G^{EQ} is the same as G except that all e-literals $\_l \in U$ appearing in G have been replaced in G^{EQ} by the constant $c \in C$ such that $\phi \models \_l \text{ EQ } c$ (if such a constant exists).

In other words, in the EQ-completed form of an RDF$^{\text{i}}$ database D, all e-literals that are entailed by the global constraint to be equal to a constant from C are substituted by that constant in all the triples in which they appear.

Definition 14. Let $D = (G, \phi)$ be an RDF$^{\text{i}}$ database. The *normalized* form of D is the RDF$^{\text{i}}$ database $D^* = (G^*, \phi)$ where G^* is the set

$$\{(t, \theta) \mid (t, \theta_i) \in G \text{ for all } i = 1 \dots n, \text{ and } \theta \text{ is } \bigvee_i \theta_i\}.$$

The normalized form of an RDF$^{\text{i}}$ database D is one that consists of the same global constraint and a graph in which conditional triples with the same triple part have been joined into a *single conditional triple* with a condition which is the disjunction of the conditions of the original triples. These new conditional triples do not follow Definition 1 which assumes conditions to be conjunctions of $\mathcal{L}$-constraints. We will allow this deviation from Definition 1 in this section.

Lemma 2. Let $D = (G, \phi)$ be an RDF$^{\text{i}}$ database. Then, $\bigcap Rep(D) = \bigcap Rep((D^{\text{EQ}})^*)$.

Based on Lemma 2, the following algorithm computes the certain answer.

Theorem 2. Let $D = (G, \phi)$ be an RDF$^{\text{i}}$ database and q a query from $\mathcal{Q}^{C'}_{AUF}$ or $\mathcal{Q}^{C'}_{WD}$. The certain answer of q over D can be computed as follows: i) compute $[\![q]\!]_D$ according to Section 5 and let $D_q = (G_q, \phi)$ be the resulting RDF$^{\text{i}}$ database, ii) compute the RDF$^{\text{i}}$ database $(H_q, \phi) = ((D_q)^{\text{EQ}})^*$, and iii) return the set of RDF triples $\{(s, p, o) \mid ((s, p, o), \theta) \in H_q$ such that $\phi \models \theta$ and $o \notin U\}$.

Let us now study the data complexity of computing the certain answer to a CONSTRUCT query over an RDF$^{\text{i}}$ database when $\mathcal{L}$ is one of the constraint languages of Section 2. We first define the corresponding decision problem.

Definition 15. Let q be a CONSTRUCT query. The *certainty problem* for query q, RDF graph H, and RDF$^{\text{i}}$ database D, is to decide whether $H \subseteq \bigcap [\![q]\!]_{Rep(D)}$. We denote this problem by $CERT_C(q, H, D)$.

The next theorem shows how one can transform the certainty problem to the problem of deciding whether $\psi \in Th(\mathbf{M}_{\mathcal{L}})$ for an appropriate sentence ψ of $\mathcal{L}$.

Theorem 3. Let $D = (G, \phi)$ be an RDF$^{\text{i}}$ database, q a query from $\mathcal{Q}^{C'}_{AUF}$ or $\mathcal{Q}^{C'}_{WD}$, and H an RDF graph. Then, $CERT_C(q, H, D)$ is equivalent to deciding whether formula $\bigwedge_{t \in H} (\forall \_\mathbf{l})(\phi(\_\mathbf{l}) \supset \Theta(t, q, D, \_\mathbf{l}))$ is *true* in $\mathbf{M}_{\mathcal{L}}$ where:

- $\_\mathbf{l}$ is the vector of all e-literals in the database D.
- $\Theta(t, q, D, \_\mathbf{l})$ is a disjunction $\theta_1 \vee \cdots \vee \theta_k$ that is constructed as follows. Let $[\![q]\!]_D = (G', \phi)$. $\Theta(t, q, D, \_\mathbf{l})$ has a disjunct θ_i for each conditional triple $(t'_i, \theta'_i) \in G'$ such that t and t'_i have the same subject and predicate. θ_i is:
  - θ'_i if t and t'_i have the same object as well.
  - $\theta'_i \wedge (\_l \text{ EQ } o)$ if the object of t is $o \in C$ and the object of t'_i is $\_l \in U$.

  If t does not agree in the subject and predicate position with some t'_i, then $\Theta(t, q, D, \_\mathbf{l})$ is taken to be $false$.

7.1 Data Complexity Results

In the full version of the paper, we show that the data complexity of the certainty problem, $CERT_C(q, H, D)$, for q in the $\mathcal{Q}^{C'}_{AUF}$ fragment of SPARQL and D in the set of RDF$^{\text{i}}$ databases with constraints from ECL, diPCL, dePCL, and RCL is coNP-complete. This follows from known results of [3] for ECL and [7] for diPCL, dePCL, RCL. Thus, we have the expected increase in data complexity given that the complexity of evaluating AND, UNION, and FILTER graph patterns over RDF graphs is LOGSPACE [16].

Theorem 3 gives us immediately some easy upper bounds on the data complexity of the certainty problem in the case of RDF$^{\text{i}}$ with $\mathcal{L}$ equal to TCL or PCL. The satisfiability problem for conjunctions of TCL-constraints is known to be in PTIME [18]. Thus, the entailment problems arising in Theorem 3 can be trivially solved in EXPTIME. Therefore, the certainty problem is also in

EXPTIME. To the best of our knowledge, no better bounds are known in the literature of TCL that we could have used to achieve a tighter bound for the certainty problem as we have done with the languages of the previous paragraph. [9] shows that conjunctions of atomic RCC-5 constraints involving constants that are polygons in V-representation can be decided in PTIME. Therefore, by restricting PCL so that only RCC-5 constraints are allowed and constants are given in V-representation, the certainty problem in this case is also in EXPTIME.

8 Related Work

Related work for incomplete information in relational databases, XML, and RDF has been discussed in the introduction, so we do not repeat the details here. The study of incomplete information in RDF undertaken in this paper goes beyond [1] where only the issue of OWA for RDF is investigated. Other cases of incomplete information in RDF (e.g., blank nodes according to the W3C RDF semantics which is different than the SPARQL semantics as we pointed out in Section 5) can also be investigated using an approach similar to ours. Comparing our work with [4,5], we point out that these papers study complementary issues in the sense that they concentrate on temporal information of a specific kind only (validity time for a tuple). From a technical point of view, the approach of [5] is similar to ours since it is based on constraints, but, whereas we concentrate on query answering for RDF$^{\text{i}}$, [5] concentrates more on semantic issues such as temporal graph entailment. RDF$^{\text{i}}$ can be used to represent incomplete temporal information that can be modeled as the object of a triple using any of the temporal constraint languages mentioned in Section 2. In this way RDF$^{\text{i}}$ can represent *user-defined* time (e.g., the time an event occured) which has not been studied in [4,5].

It is interesting to compare the expressive power that RDF$^{\text{i}}$ gives us to other recent works that use Semantic Web data models and languages for geospatial applications. When equipped with a constraint language like TCL, PCL, or RCL, RDF$^{\text{i}}$ goes beyond the proposals of the geospatial extensions of SPARQL, stSPARQL [8] and GeoSPARQL [13] that cannot query incomplete geospatial information. While GeoSPARQL provides a vocabulary for asserting topological relations the complexity of query evaluation over RDF graphs in this case remains an open problem. Incomplete geospatial information as it is studied in this paper can also be expressed in spatial description logics [14,10]. For efficiency reasons, spatial DL reasoners such as RacerPro[2] and PelletSpatial [20] have opted for separating spatial relations from standard DL axioms as we have done by separating graphs and constraints. Since RDF graphs can be seen as DL ABoxes with atomic concepts only, all the results of this paper can be transferred to the relevant subsets of spatial DLs and their reasoners so they are applicable to this important Semantic Web area as well. It is an open problem how to extend our results to DLs with non-trivial TBoxes.

[2] `http://www.racer-systems.com/`

9 Future Work

In the future we will: 1) explore other fragments of SPARQL that can be used to define a representation system for RDF$^{\text{i}}$, 2) study in more depth the complexity of certain answer computation for the various constraint languages $\mathcal{L}$ we considered or the one used in [5] and identify tractable classes, 3) study the complexity of evaluating various fragments of SPARQL over RDF$^{\text{i}}$ databases as in [16,19].

References

1. Arenas, M., Pérez, J.: Querying semantic web data with SPARQL. In: PODS. pp. 305–316 (2011)
2. Barceló, P., Libkin, L., Poggi, A., Sirangelo, C.: XML with incomplete information. JACM 58(1), 4 (2010)
3. Grahne, G.: The Problem of Incomplete Information in Relational Databases. LNCS, Springer Verlag (1991)
4. Gutierrez, C., Hurtado, C.A., Vaisman, A.A.: Introducing Time into RDF. IEEE TKDE 19(2) (2007)
5. Hurtado, C.A., Vaisman, A.A.: Reasoning with Temporal Constraints in RDF. In: PPSWR. Springer (2006)
6. Imielinski, T., Lipski, W.: Incomplete Information in Relational Databases. JACM 31(4), 761–791 (1984)
7. Koubarakis, M.: Complexity results for first-order theories of temporal constraints. In: KR. pp. 379–390 (1994)
8. Kyzirakos, K., Karpathiotakis, M., Koubarakis, M.: Strabon: A Semantic Geospatial DBMS. In: ISWC'12. pp. 295–311 (2012)
9. Liu, W., Wang, S., Li, S., Liu, D.: Solving qualitative constraints involving landmarks. In: CP (2011)
10. Lutz, C., Miličić, M.: A tableau algorithm for description logics with concrete domains and general tboxes. J. Autom. Reason. 38, 227–259 (April 2007)
11. Nikolaou, C., Koubarakis, M.: Incomplete information in RDF. CoRR abs/1209.3756 (2012)
12. Nikolaou, C., Koubarakis, M.: Querying Linked Geospatial Data with Incomplete Information. In: 5th International Terra Cognita Workshop. Boston, USA (2012)
13. Open Geospatial Consortium: GeoSPARQL - A geographic query language for RDF data. OGC (2010)
14. Özcep, Ö., Möller, R.: Computationally feasible query answering over spatiothematic ontologies. In: GEOProcessing (2012)
15. Pérez, J., Arenas, M., Gutierrez, C.: Semantics of SPARQL. Tech. rep., Univ. de Chile (2006), `http://ing.utalca.cl/~jperez/papers/sparql_semantics.pdf`
16. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. ACM TODS 34(3), 1–45 (2009)
17. Randell, D.A., Cui, Z., Cohn, A.G.: A spatial logic based on regions and connection. In: KR (1992)
18. Renz, J., Nebel, B.: On the complexity of qualitative spatial reasoning: A maximal tractable fragment of the region connection calculus. AIJ 108(1-2), 69–123 (1999)
19. Schmidt, M., Meier, M., Lausen, G.: Foundations of SPARQL query optimization. In: ICDT. pp. 4–33 (2010)
20. Stocker, M., Sirin, E.: PelletSpatial: A Hybrid RCC-8 and RDF/OWL Reasoning and Query Engine. In: OWLED (2009)