Querying incomplete information in RDF with SPARQL[☆]

Charalampos Nikolaou[1], Manolis Koubarakis

Dept. of Informatics and Telecommunications
National and Kapodistrian University of Athens
Panepistimiopolis, Ilisia
Athens 15784 Greece

Abstract

Incomplete information has been studied in-depth in relational databases and knowledge representation. In the context of the Web, incomplete information issues have been studied in detail for XML, but very few papers exist that do the same for RDF. In this paper we make the first general proposal for extending RDF with the ability to represent property values that exist but are unknown or partially known using constraints. Following ideas from incomplete information literature, we develop a semantics for this extension of RDF, called RDF$^{\mathrm{i}}$, and study query evaluation for SPARQL. We transfer the concept of representation systems from incomplete information in relational databases to the case of RDF$^{\mathrm{i}}$ and identify two very important fragments of SPARQL that can be used to define a representation system for RDF$^{\mathrm{i}}$. The first corresponds to the monotone fragment of graph patterns that uses only the operators AND, UNION, and FILTER. The second corresponds to the well-designed graph patterns, that is, a fragment that uses only operators AND, FILTER, and OPT, and enjoys interesting properties that make query evaluation efficient. We prove that each of the

[☆]This is an extended version of a paper that appeared in Proc. of the 8th International Conference on Web Reasoning and Rule Systems, RR '13 [70].

Email addresses: charnik@di.uoa.gr (Charalampos Nikolaou), koubarak@di.uoa.gr (Manolis Koubarakis)

URL: http://cgi.di.uoa.gr/~charnik (Charalampos Nikolaou), http://cgi.di.uoa.gr/~koubarak (Manolis Koubarakis)

[1]Corresponding author.

two fragments can be used to define a representation system for CONSTRUCT queries without blank nodes in their templates. We also define the fundamental concept of certain answers to SPARQL queries over RDF$^{\mathrm{i}}$ databases and present an algorithm for its computation. Then, we present complexity results for computing certain answers by considering equality, temporal, and spatial constraint languages and the class of CONSTRUCT queries of our representation systems. Finally, we demonstrate the usefulness of RDF$^{\mathrm{i}}$ in geospatial Semantic Web applications by giving a number of examples and comparing the modeling capabilities of RDF$^{\mathrm{i}}$ with related formalisms found in the literature.
Keywords: Incomplete information, semantic web, RDF, SPARQL
2010 MSC: 68T27, 68T30

1. Introduction

Incomplete information has been studied in-depth in relational databases [41, 30, 29] and knowledge representation [79, 15]. 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, then we cannot assume its negation as in the Closed World Assumption (CWA). Application knowledge captured by databases and knowledge bases is often incomplete, thus the OWA is a useful assumption to make. In general, the richer an application domain is, the more possible it is that a framework based on incomplete information will be required. For example, medical and biological information systems, geographical information systems, planning and scheduling systems, as well as design systems are just a few examples of systems operating in application domains with very rich semantics that require modeling of incomplete information.

Incomplete information can also arise even if we start from complete databases, e.g., in relational view updates [1], data integration [51], or data exchange [26, 53, 7]. Incomplete information is also very important since it can serve as a

means for representing all repairs of a possibly inconsistent database [8, 18, 23]. Thus, the detailed study of incomplete information has been a recurring theme in the literature throughout the years.

In the context of the Web, incomplete information has recently been studied in detail for XML [4, 12, 20]. As Semantic Web technologies achieve maturity and gain acceptance in a wide variety of application domains through the creation of ontologies and linked data pools, we expect the study of issues related to incomplete information to gain more attention in the Semantic Web community as well. There have been some recent papers that confirm our expectations.

Gutierrez et al. [35] introduce the concept of *anonymous timestamps* in general temporal RDF graphs, i.e., graphs G 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. Hurtado and Vaisman [39] subsequently extend the concept of general temporal RDF graphs so that one is allowed to express temporal constraints involving anonymous timestamps. Such temporal constraints are given using conjunctions of order constraints of the form $x_1 \; OP \; x_2$, where OP is an arithmetic comparison operator, such as $<$, $\leq$, etc. Hurtado and Vaisman [39] call the resulting pairs (G, ϕ) *c-temporal graphs*. In the same paper, the authors define a semantics for c-temporal graphs and study the relevant problem of entailment.

More recently, the work of Arenas and Pérez [10] examines the question of whether SPARQL is an appropriate language for RDF given the OWA typically associated with the framework. It defines *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. Arenas and Pérez [10] show 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 [76], then we get the same result we would get if we used the certain answers semantics. The converse also holds,

thus monotone SPARQL graph patterns have this nice property. However, the OPTIONAL operator (OPT) is not monotone and the two semantics do not coincide for it. Arenas and Pérez [10] define the notion of weak monotonicity that appears to capture the intuition behind OPT and show that a SPARQL query q is weakly monotone if and only if evaluating q under the W3C semantics gives the same result as evaluating q under a new semantics appropriate for weakly monotone queries. Finally, in the same work they show that the fragment of SPARQL consisting of the well-designed graph patterns defined originally by Pérez et al. [75] is weakly monotone.

1.1. Contributions

In this paper we continue the line of research started by Gutierrez et al. [35], Hurtado and Vaisman [39], and Arenas and Pérez [10], 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.

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.* Such information is abundant in recent applications where RDF is being used (e.g., sensor networks [33], the modeling of geospatial information [49, 71]). In the proposed extension of RDF, called RDF$^{\mathrm{i}}$ (where "i" stands for "incomplete"), e-literals are allowed to appear only in the object position of triples.

Previous research on incomplete information in databases and knowledge representation has shown that in many applications, having the ability to state *constraints* about values that are only partially known is a very desirable feature and leads to the development of very expressive formalisms [30, 47]. In the spirit of this tradition, RDF$^{\mathrm{i}}$ 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$^{\mathrm{i}}$ extends the concept of an RDF graph to the concept of an RDF$^{\mathrm{i}}$ *database* that is a pair (G, ϕ) where G is an RDF

Table 1: Data complexity of the certainty problem for CONSTRUCT queries, RDF$^{\mathrm{i}}$ databases, and various constraint languages $\mathcal{L}$ in comparison with the data complexity of the evaluation problem of SPARQL graph patterns over RDF graphs

Problem	**Constraint Language $\mathcal{L}$**	**Data Complexity**
Certainty	ECL	coNP-complete (Proposition 9.11)
	diPCL/dePCL/RCL	coNP-complete (Proposition 9.12)
	TCL	coNP-complete (Proposition 9.13)
	PCL	coNP-complete (Proposition 9.14)
Evaluation of SPARQL graph patterns	—	in LOGSPACE [75]

graph possibly containing triples with e-literals in their object positions, and ϕ is a quantifier-free formula of $\mathcal{L}$. The choice of the constraint language $\mathcal{L}$ parameterizes RDF$^{\mathrm{i}}$ making it a framework rather than a data model. Our workshop paper [69] motivates the need for introducing RDF$^{\mathrm{i}}$ by concentrating on the representation of incomplete spatial knowledge.

Following ideas from the incomplete information literature [41, 30], we develop a semantics for RDF$^{\mathrm{i}}$ databases and SPARQL query evaluation. The semantics defines the set of possible RDF graphs corresponding to an RDF$^{\mathrm{i}}$ database and the fundamental concept of certain answers for SPARQL query evaluation over an RDF$^{\mathrm{i}}$ database. We transfer the well-known concept of *representation systems* of Imieliński and Lipski [41] to the case of RDF$^{\mathrm{i}}$, 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 Pérez et al. [75] can be used to define a representation system for RDF$^{\mathrm{i}}$. Our results on the monotonicity of CONSTRUCT queries (even in the case of well-designed patterns that contain operator OPT) indicate their importance and sets an interesting subject to explore in theoretical treatments of RDF.

We define the fundamental concept of *certain answers* to SPARQL queries over RDF$^{\mathrm{i}}$ databases and present an algorithm for its computation. Finally, we

present complexity results for the associated decision problem, named *certainty*, by considering equality (language ECL), temporal (languages diPCL/dePCL), and spatial (languages RCL, TCL, and PCL) constraint languages and the class of CONSTRUCT queries that forms a representation system for RDF$^{\text{i}}$. Our results are summarized in Table 1 and show that the data complexity of the certainty problem for RDF$^{\text{i}}$ and this class of queries increases from LOGSPACE[2] (the upper bound for queries from this class over RDF graphs [75]) to coNP-complete for all of the considered constraint languages. This result is in line with similar complexity results for querying incomplete information in relational databases [30, 46]. The complexity of the closely related problem of SPARQL query evaluation over RDF graphs as manifested in the geospatial extensions stSPARQL [50] and GeoSPARQL [71] has not been investigated so far in any detail, and it remains an open problem.

The organization of the paper is as follows. Section 2 introduces RDF$^{\text{i}}$ by giving examples and comparing it with well-known concepts of the relational database literature on incomplete information. Section 3 presents the properties that we expect constraint languages to have so that they can be used in RDF$^{\text{i}}$. In addition, it defines some useful constraint languages that are used in the paper. Section 4 formally introduces RDF$^{\text{i}}$ and then Section 5 defines its semantics. Section 6 defines the evaluation of SPARQL queries over RDF$^{\text{i}}$ databases and Section 7 presents two monotone fragments of SPARQL. Then, Section 8 defines the concept of certain answers and investigates which fragments of SPARQL can be used to define a representation system for RDF$^{\text{i}}$. Section 9 gives an algorithm for computing the certain answers for SPARQL queries over RDF$^{\text{i}}$ databases and presents relevant complexity results for the associated decision problem. Then, Section 10 compares RDF$^{\text{i}}$ with related work, and last, Section 11 summarizes our work and discusses future work.

[2]One can prove that the problem is in AC^0 for data complexity as it is the case for the problem of evaluating first-order queries in relational databases [2].

2. Motivation

Incomplete information is often present in geospatial applications where data is imprecise, indefinite, or qualitative. For example, in the FP7 European project TELEIOS[3] satellite images are used for environmental disaster monitoring (e.g., fires, floods). The following is a list of triples (namespaces are omitted) that gives an example of the kind of representation employed in TELEIOS for representing pixels of a satellite image (called hotspots) corresponding to geographic regions that are probably on fire.

hotspot1 a Hotspot .	fire1 a Fire .
hotspot1 correspondsTo fire1 .	fire1 occurredIn point1 .
point1 hasGeometry "x = 24.82566 ∧ y = 35.31064"^^SemiLinearPointSet .	

The above set of triples is a graph in the model stRDF of Koubarakis and Kyzirakos [49]. The stRDF model extends RDF with the ability to represent geometries over $\mathbb{Q}^k$ that change over time following the paradigm of constraint databases [44]. Geometries and valid times of triples correspond to *semi-linear point sets* that are the subsets of $\mathbb{Q}^k$ defined by Boolean combinations of linear constraints. In stRDF, such combinations of constraints are given as literals of type SemiLinearPointSet. The above graph represents *definite* information; it states that there is a hotspot corresponding to a fire taking place at the point $(24.82566, 35.31064) \in \mathbb{Q}^2$.

Due to the medium resolution of the satellite images, each image pixel representing a hotspot corresponds to a 3km by 3km rectangle in geographic space. Thus, a more appropriate representation of the real world situation that corresponds to a hotspot would be to state that there is a geographic region with unknown exact coordinates where a fire is taking place, and that region is included in a known 3km by 3km rectangle. This real world situation can be represented by an RDF$^{\text{i}}$ database as shown in Example 2.1 below.

[3] http://earthobservatory.eu/

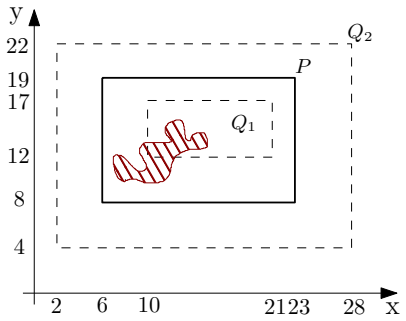


Figure 1: Rectangles mentioned in the examples

Example 2.1. The following is an RDF$^{\text{i}}$ database.

hotspot1 a Hotspot . fire1 a Fire .

hotspot1 correspondsTo fire1 . fire1 occurredIn _R1 .

_R1 NTPP "$x \geq 6 \wedge x \leq 23 \wedge y \geq 8 \wedge y \leq 19$"

Fire fire1 (shaded area of Figure 1) is asserted to have taken place inside region _R1. Term _R1 is an e-literal of datatype SemiLinearPointSet and is asserted to be inside the rectangle formed by the points $(6, 8)$ and $(23, 19)$ (rectangle P in Figure 1). This is stated with a constraint expressed in the language PCL to be defined below, in Section 3.2. NTPP is the "non-tangential proper part" relation of RCC-8 [78]. Constraints in PCL can express qualitative and quantitative spatial information about regions in $\mathbb{Q}^2$.

E-literals in RDF$^{\text{i}}$ are like existentially quantified variables in first-order logic or Skolem constants. That is, RDF$^{\text{i}}$ treats e-literals the same way the RDF model theory [37] treats blank nodes. However, due to the mismatch in the treatment of blank nodes in the literature [9, 64], we choose to dissociate e-literals from blank nodes in RDF$^{\text{i}}$. This is captured formally by our definitions in Section 4 which employ two disjoint sets for the two concepts. A similar assumption is made by Gutierrez et al. [35] and Hurtado and Vaisman [39] where anonymous nodes are taken to be different from blank nodes. RDF$^{\text{i}}$ databases like the one of Example 2.1 consist of two parts: a graph (i.e., a set of triples) and a *global constraint*. Global constraints can in general be quantifier-free

formulae of some first-order constraint language. RDF$^{\text{i}}$ databases are syntactic devices for the representation of incomplete information.

Example 2.2. Let us consider the query "Find all fires that have occurred in a region which is a non-tangential proper part of rectangle Q_1 of Figure 1" over the database of Example 2.1. Using SELECT in SPARQL, this query can be formulated as follows:

```
SELECT ?F WHERE {
    ?F a Fire . ?F occurredIn ?R .
    FILTER (?R NTPP "x ≥ 10 ∧ x ≤ 21 ∧ y ≥ 12 ∧ y ≤ 17") }.
```

In the above query, the FILTER operator contains as a condition a constraint expressed in the PCL language. SPARQL queries in RDF$^{\text{i}}$ differ from standard SPARQL only with respect to the FILTER operator which is extended with another kind of conditions that are Boolean combinations of atomic constraints of the constraint language $\mathcal{L}$ with which RDF$^{\text{i}}$ is parameterized.

If we examine the database of Example 2.1, we can see that the answer should be *conditional* [41]. We cannot say for sure whether fire1 satisfies the requirements of the query because the information in the database is indefinite (the exact geometry of _R1 is not known). Fire fire1 qualifies only in the possible graphs where _R1 is a non-tangential proper part of the rectangle mentioned in the query. For every object that qualifies as an answer, the query answering procedure should also provide a *condition* characterizing this set of possible graphs. Following the ideas of conditional tables of Imieliński and Lipski [41], this answer can be represented by the following set of *conditional mappings*:

$$\big\{(\{?\text{F} \to \text{fire1}\},\ \text{\_R1 NTPP "x} \geq 10 \wedge \text{x} \leq 21 \wedge \text{y} \geq 12 \wedge \text{y} \leq 17\text{"})\big\}.$$

The e-literals in the above answer are implicitly constrained by the global constraint of the original database. This will be captured formally in Section 6. Conditional mappings are different from standard SPARQL mappings [75] in the sense that they map variables to constants only if a condition holds. Thus,

they are reminiscent of conditional tuples in the conditional table model of Grahne [30].

Example 2.3. If we wanted to have an RDF$^{\text{i}}$ database as the answer to a query like the one of Example 2.2, then we would have queried the RDF$^{\text{i}}$ database of Example 2.1 using the CONSTRUCT query form of SPARQL. The query would be expressed as follows:

CONSTRUCT { ?F a Fire . }
WHERE { ?F a Fire . ?F occurredIn ?R .
FILTER (?R NTPP "$x \geq 10 \wedge x \leq 21 \wedge y \geq 12 \wedge y \leq 17$") }.

The answer to this query would be an RDF$^{\text{i}}$ database containing *conditional triples* adhering to the query template $\{(?\text{F}, \text{a}, \text{Fire})\}$. The template is instantiated for each conditional mapping from the evaluation of the graph pattern of the query, and the resulting triple together with the condition of the mapping form a conditional triple in the resulting database. Therefore, the answer to query of Example 2.3 consists of the following conditional triple:

$\big($(fire1 a Fire), _R1 NTPP "$x \geq 10 \wedge x \leq 21 \wedge y \geq 12 \wedge y \leq 17$"$\big)$.

In some cases the user might know that the information in the database is incomplete. Thus, she might wish to find all values that *certainly* satisfy some qualification (this is the well-known notion of certain answers in incomplete databases [30]). Let us consider the query of Example 2.3 again. If we rephrase it to "Find fires that have *certainly* occurred in a region which is a non-tangential proper part of rectangle Q_2 in Figure 1", then fire1 satisfies the query unconditionally and the certain answers is the set of RDF triples containing the following triple: fire1 a Fire.

3. Constraint Languages

We will consider many-sorted first-order languages, structures, and theories [25]. Every language $\mathcal{L}$ will be interpreted over a *fixed* structure, called the

intended structure, which will be denoted by $\mathbf{M}_{\mathcal{L}}$. If $\mathbf{M}_{\mathcal{L}}$ is a structure then $Th(\mathbf{M}_{\mathcal{L}})$ will denote the theory of $\mathbf{M}_{\mathcal{L}}$, i.e., the set of sentences of $\mathcal{L}$ that are true in $\mathbf{M}_{\mathcal{L}}$. For every language $\mathcal{L}$, the class of its atomic formulae will include *true* and *false* with obvious semantics. We will also distinguish a class of quantifier-free formulae called $\mathcal{L}$-*constraints* which will include the atomic formulae of $\mathcal{L}$ and any Boolean combination of them. Every first-order language $\mathcal{L}$ we consider has also a distinguished equality predicate, EQ, with the standard semantics.

In the following we define formally various constraint languages that allow us to explore the scope of modeling possibilities RDF$^{\text{i}}$ offers.

3.1. The Language ECL

Language *ECL* (*E*quality *C*onstraint *L*anguage) is the first-order language of equality constraints. The atomic formulae of ECL are of the form x_1 EQ x_2 where x_1, x_2 are variables or constants interpreted over an infinite domain. The intended structure for this language, $\mathbf{M}_{ECL}$, interprets symbol EQ as equality and constants as "themselves".

ECL has been used by Kanellakis et al. [44] for the development of an extended relational model based on ECL-constraints and by Imieliński and Lipski [41], Abiteboul et al. [3], and Grahne [30] for querying and updating incomplete information in relational databases. Therefore, when used in RDF$^{\text{i}}$, this language allows us to extend RDF with the ability to represent "marked nulls" as in classical relational databases [41].

3.2. The Languages PCL and TCL

Language *PCL* (*P*olygon *C*onstraint *L*anguage) allows us to represent topological properties of non-empty regular closed subsets of $\mathbb{Q}^2$. In topology, a subset of a topological space is said to be *regular closed* if it is equal to the closure of its interior (we will call these subsets *regions* for brevity). PCL is a first-order language with the following 8 binary predicate symbols corresponding to the topological relations of the RCC-8 calculus [78]: DC, EC, PO, TPP, NTPP, TPPi, NTPPi, and EQ. The constant symbols of PCL represent

polygons in $\mathbb{Q}^2$. We will write these constants as Boolean combinations of linear constraints in double quotes[4]. Such quantifier-free formulae define sets of points in $\mathbb{Q}^2$, called semi-linear sets, and can be used to represent a variety of spatial geometries, such as points, lines, line segments, bounded or unbounded polygons, convex or non-convex, as well as unions of polygons possibly with holes. In PCL, however, constants are restricted to those quantifier-free formulae defining point sets that represent bounded polygons, i.e., their area is finite. The terms and atomic formulae of PCL are defined as follows. Constants and variables are *terms*. An *atomic formula* of PCL is a formula of the form $t_1\ R\ t_2$ where t_1, t_2 are terms and R is one of the RCC-8 predicates. For example, the following are PCL-constraints:

$$\mathrm{r}_1 \text{ NTPP } \mathrm{r}_2,\ \mathrm{r}_2 \text{ EC "}\mathrm{x} - \mathrm{y} \geq 0 \wedge \mathrm{x} \leq 1 \wedge \mathrm{y} \geq 0\text{"}.$$

The intended structure for PCL, denoted by $\mathbf{M}_{PCL}$, has the set of non-empty regions 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.

PCL is the language that we have used in the paper [69] and we will also use it in the examples of this article. PCL can be used to capture the topology of regions of interest some of which may be known to have a certain polygonal shape.

Language TCL (*T*opological *C*onstraint *L*anguage) is defined like PCL, but now terms can only be variables (no topological reasoning with constants is allowed). Language TCL allows us to capture the topology of regions of interest to an application but makes no commitment regarding other non-topological properties of these regions, e.g., shape.

The following two languages, diPCL and dePCL, were defined by Koubarakis [46].

[4] In computational geometry, this is the so-called half-space representation or H-representation.

3.3. The Languages diPCL and dePCL

Language *diPCL* (*di*screte *P*oint *C*onstraint *L*anguage) allows us to make statements about points in discrete time. It is a first-order language with constants from the set of integers $\mathbb{Z}$, a binary function symbol $-$, and a binary predicate symbol $<$. The terms and atomic formulae of diPCL are defined as follows. Constants and variables are *terms*. If t_1 and t_2 are constants or variables, then $t_1 - t_2$ is a term. An *atomic formula* of diPCL is a formula of the form $t < c$ or t EQ c where t is a term and c is a constant. For example, the following are diPCL-constraints:

$$x_1 - x_2 < 2,\ x_1 \text{ EQ } 5,\ x_1 < 6.$$

The intended structure for diPCL, denoted by $\mathbf{M}_{diPCL}$, has the set of integers as its domain. $\mathbf{M}_{diPCL}$ interprets each constant symbol by the corresponding number in $\mathbb{Z}$, function symbol $-$ by the subtraction operation over the integers, and predicate symbol $<$ by the relation "less than". Then, theory $Th(\mathbf{M}_{diPCL})$ is a sub-theory of $Th(\mathbb{Z}, +, <)$, the theory of integers with addition and order (or Presburger arithmetic) [77].

The language *dePCL* (*de*nse *P*oint *C*onstraint *L*anguage) allows us to make statements about points in dense time. Language dePCL is defined like diPCL above with the following exception: the constants of dePCL are from the set of rational numbers $\mathbb{Q}$ that serves also as its domain. Then, theory $Th(\mathbf{M}_{dePCL})$ is a sub-theory of $Th(\mathbb{R}, +, <)$, the theory of real numbers with addition and order [77].

Languages diPCL and dePCL are constraint languages that allow RDF$^{\text{i}}$ to represent incomplete temporal information as in the works of Gutierrez et al. [35], Hurtado and Vaisman [39], and Koubarakis [48].

3.4. The Language RCL

Language *RCL* (*R*ectangle *C*onstraint *L*anguage) allows us to capture spatial and metric constraints (e.g., topological or directional, and horizontal or vertical distance constraints among the edges of rectangles) involving rectangles with

sides parallel to the axes in $\mathbb{Q}^2$ (we will call them *boxes*). RCL is useful not only for modeling regions of space with such rectangular shapes but also for modeling *minimum bounding rectangles* that are typically used as approximations of spatial objects, e.g., in spatial data structures and elsewhere.

RCL is a first-order language with equality and two sorts: the sort $\mathcal{Q}$ for rational constants, and the sort $\mathcal{R}$ for boxes. The set of non-logical symbols of RCL includes: all rational constants of sort $\mathcal{Q}$, a binary function symbol $-$ of sort $(\mathcal{Q}, \mathcal{Q}, \mathcal{Q})$, function symbols $LL_x(\cdot), LL_y(\cdot), UR_x(\cdot), UR_y(\cdot)$ of sort $(\mathcal{R}, \mathcal{Q})$, and predicate symbol $<$ of sort $(\mathcal{Q}, \mathcal{Q})$.

The terms and atomic formulae of RCL are defined as follows. Constants of sort $\mathcal{Q}$ and variables of sort $\mathcal{R}$ are terms. If r is a variable of sort $\mathcal{R}$ then $LL_x(r), LL_y(r), UR_x(r)$ and $UR_y(r)$ are terms of sort $\mathcal{Q}$. If t_1, t_2 are terms of sort $\mathcal{Q}$, then $t_1 - t_2$ is a term of sort $\mathcal{Q}$. An *atomic formula* of RCL is a formula of the form $t < c$ or t EQ c where t is a term of sort $\mathcal{Q}$ and c a rational constant. For example, the following are RCL-constraints:

$$LL_x(r_2) - LL_x(r_1) < 0,\ UR_y(r_1) - LL_y(r_2)\ \text{EQ}\ \frac{5}{2}.$$

The intended structure for RCL, denoted by $\mathbf{M}_{RCL}$, interprets each non-logical symbol as follows. Each rational constant is interpreted by its corresponding rational number. The function symbol $-$ is interpreted by the subtraction operation over the rationals, while the function symbols $LL_x(\cdot)$, $LL_y(\cdot)$, $UR_x(\cdot)$, and $UR_y(\cdot)$ are interpreted by the easily-defined functions that given a box in $\mathbb{Q}^2$, return the x- and y-coordinate of its lower-left and upper-right vertex respectively. Predicate $<$ is interpreted by the relation "less than" over $\mathbb{Q}$.

The above constraint languages will be considered again in Section 9 where we investigate how they affect the complexity of computing certain answers for queries over RDF$^{\text{i}}$ databases. Although the definitions of these languages are general, the technical developments in that section make use of sentences with universal quantifiers only.

4. The RDF$^{\text{i}}$ Framework

As in theoretical treatments of RDF [34], 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 set of typed literals U, disjoint from I, B, and L above, that we call e-literals. Further, we assume the existence of a datatype map M [37] and distinguish a set of datatypes A from M for which e-literals are allowed. For each datatype d in A, we assume the existence of a countably infinite set U_d of e-literals. The sets U_d are assumed to be pairwise disjoint. Then, set U above is simply the union of the sets U_d. By convention, the identifiers of e-literals will start with an underscore, e.g., _R5. Finally, we assume the existence of a many-sorted first-order constraint language $\mathcal{L}$ with the properties discussed in Section 3. Language $\mathcal{L}$ is related to the datatype map M in the following way:

- The set of sorts of $\mathcal{L}$ is the set of datatypes A of M.
- The set of constants of $\mathcal{L}$ is the union of the lexical spaces of the datatypes in A.
- $\mathbf{M}_{\mathcal{L}}$ 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) will be denoted by C. The set C is assumed to be disjoint from I, B, L, and U. The set of RDF$^{\text{i}}$ *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 in the linear constraint syntax of Section 3.

We now define the basic concepts of RDF$^{\text{i}}$: e-triples, conditional triples, conditional graphs, global constraints, and databases.

Definition 4.1. 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 an $\mathcal{L}$-constraint. If (t, θ) is a conditional triple, then θ will be called the condition of the triple. A *conditional graph* is a set of conditional triples. A *global constraint* is a satisfiable $\mathcal{L}$-constraint. An RDF$^{\mathrm{i}}$ *database* D is a pair $D = (G, \phi)$ where G is a conditional graph and ϕ a global constraint.

In the rest of the paper, when we want to refer to standard RDF constructs we will write "RDF triple" and "RDF graph" so that no confusion with RDF$^{\mathrm{i}}$ is possible.

Example 4.2. Using the formal notation of this section, the pair

$$\big(\{((\text{hotspot1}, \text{a}, \text{Hotspot}), \text{true}),\ ((\text{fire1}, \text{a}, \text{Fire}), \text{true}),$$
$$((\text{hotspot1}, \text{correspondsTo}, \text{fire1}), \text{true}),$$
$$((\text{fire1}, \text{occurredIn}, \_\text{R1}), \text{true})\}, \_\text{R1 NTPP "g1"}\big)$$

is the RDF$^{\mathrm{i}}$ database of Example 2.1 where g1 corresponds to the half-space representation of rectangle P in Figure 1, i.e., the linear constraint $\text{x} \geq 6 \wedge \text{x} \leq 23 \wedge \text{y} \geq 8 \wedge \text{y} \leq 19$. In the rest of the article, we will re-use "g1" and introduce further aliases for polygonal constants when appropriate. By convention, different aliases will denote different (and non-equivalent) linear constraints, so that their comparison is immediate.

5. Semantics of RDF$^{\mathrm{i}}$

The semantics of RDF$^{\mathrm{i}}$ is inspired by Imieliński and Lipski [41]. An RDF$^{\mathrm{i}}$ 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$^{\mathrm{i}}$ database. The global constraint ϕ determines the set 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 the constraint ϕ.

Example 5.1. Let $D = (G, \phi)$ be the RDF$^{\mathrm{i}}$ database given in Example 4.2. It mentions a hotspot, which is located in a region that is inside but does not

intersect with the boundary of the rectangle P defined by the points $(6, 8)$ and $(23, 19)$ (see also Figure 1). The same knowledge can be represented by a (possibly infinite) set of possible RDF graphs, one for each rectangle inside P. Two of these graphs are:

$$
\begin{aligned}
G_1 = \{\ & (\text{hotspot1}, \text{a}, \text{Hotspot}),\ (\text{fire1}, \text{a}, \text{Fire}),\\
& (\text{hotspot1}, \text{correspondsTo}, \text{fire1}),\\
& (\text{fire1}, \text{occurredIn}, \text{"x} \geq 11 \wedge \text{x} \leq 15 \wedge \text{y} \geq 13 \wedge \text{y} \leq 15\text{"})\ \},\\
G_2 = \{\ & (\text{hotspot1}, \text{a}, \text{Hotspot}),\ (\text{fire1}, \text{a}, \text{Fire}),\\
& (\text{hotspot1}, \text{correspondsTo}, \text{fire1}),\\
& (\text{fire1}, \text{occurredIn}, \text{"x} \geq 10 \wedge \text{x} \leq 21 \wedge \text{y} \geq 12 \wedge \text{y} \leq 17\text{"})\ \}.
\end{aligned}
$$

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.

Definition 5.2. A *valuation* v is a function from U to C assigning to each e-literal from U a constant from C.

We denote by $v(t)$ the application of a valuation v to an e-triple t. The expression $v(t)$ replaces all e-literals $\_l$ appearing in t by $v(\_l)$ and leaves all other terms the same. If θ is an $\mathcal{L}$-constraint (e.g., the condition of a conditional triple or the global constraint of a database), then the expression $v(\theta)$ denotes the application of v to θ and 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 5.3. Let G be a conditional graph and v a valuation. Then $v(G)$ denotes the RDF graph

$$\big\{v(t) \mid (t, \theta) \in G \text{ and } \mathbf{M}_{\mathcal{L}} \models v(\theta)\big\}.$$

The concept of valuations and their application to conditional graphs is demonstrated in the following example.

Example 5.4. Let $D = (G, \phi)$ be the RDF$^{\text{i}}$ database in Example 4.2 and v_1, v_2 two valuations such that $v_1(\_\text{R1}) = \text{"x} \geq 11 \wedge \text{x} \leq 15 \wedge \text{y} \geq 13 \wedge \text{y} \leq 15\text{"}$ and

$v_2(\_\text{R1}) = \text{"x} \geq 10 \wedge \text{x} \leq 21 \wedge \text{y} \geq 12 \wedge \text{y} \leq 17\text{"}$. Then, expressions $v_1(G)$ and $v_2(G)$ correspond respectively to the RDF graphs G_1 and G_2 of Example 5.1. Evidently, the application of a valuation to a conditional graph replaces all e-literals of the graph with constants and keeps only those triples with conditions equivalent to *true* making the result an RDF graph.

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

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

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 RDF graphs H containing at least the triples of $v(G)$. In this respect, we follow the approach of Arenas and Pérez [10, 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 Arenas and Pérez [10] 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 [10]; only the issue of OWA has been explored there.

In incomplete relational databases [41], *Rep* is a *semantic* function: it maps a table (a syntactic construct) to a set of relational instances (i.e., a set of possible worlds, 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 [41, 30] belong to the model-theoretic camp. However, the use of *Rep* in the above definition is different. It 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 [37]. This is a deliberate choice in our work since we want to explore which well-known techniques from incomplete relational databases carry over to the RDF framework.

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

Let us now discuss how to evaluate SPARQL queries on RDF$^{\text{i}}$ databases. We will use the algebraic syntax of SPARQL presented by Pérez et al. [75]. We will deal with both SELECT and CONSTRUCT query forms. Due to the presence of e-literals, query evaluation now becomes more complicated and is similar to query evaluation for conditional tables [41, 30]. This section provides the exact details.

6.1. Semantic Extensions for Incomplete Information

We use set semantics for query evaluation by extending the SPARQL query evaluation approach of Pérez et al. [75]. Blank nodes are interpreted as in SPARQL, i.e., as constants different from each other. Notice that this is not the same as the semantics of blank nodes in RDF model theory [37] where they are treated as existentially quantified variables.

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 disjoint from the set of terms T we defined in Section 4.

We first define the concept of *e-mappings* ("e" from the word "existential") which extends the concept of mappings of Pérez et al. [75] with the ability to have literals from the set C and e-literals as values for a special query variable.

Definition 6.1. An *e-mapping* ν is a partial function $\nu : V \to T$ such that $\nu(x) \in I \cup B \cup L$ if $x \in V_n$ and $\nu(x) \in C \cup U$ if $x \in V_s$.

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 [75] (we also use the same notation for them).

Example 6.2. The following are e-mappings:

$$\mu_1 = \{\ ?\mathrm{F} \to \mathrm{fire1}, ?\mathrm{S} \to\ \texttt{"g2"}\ \},\ \ \mu_2 = \{\ ?\mathrm{F} \to \mathrm{fire1}, ?\mathrm{S} \to \_\mathrm{R1}\ \}$$

where g2 is $\mathrm{x} \geq 1 \wedge \mathrm{x} \leq 2 \wedge \mathrm{y} \geq 1 \wedge \mathrm{y} \leq 2$. The domain of the above e-mappings consists of two query variables, namely, ?F and ?S. Variable ?F belongs to the set of normal query variables V_n, while variable ?S belongs to the set of special query variables V_s.

We now extend the concept of e-mappings and define *conditional mappings*.

Definition 6.3. A *conditional mapping* μ is a pair (ν, θ) where ν is an e-mapping and θ is an $\mathcal{L}$-constraint.

Example 6.4. The following are conditional mappings:

$\mu_1 = \big(\{?\mathrm{F} \to \mathrm{fire1}, ?\mathrm{S} \to\ \texttt{"g2"}\},\ \ \mathrm{true}\big)$,
$\mu_2 = \big(\{?\mathrm{F} \to \mathrm{fire1}, ?\mathrm{S} \to \_\mathrm{R1}\},\ \ \_\mathrm{R1}\ \ \mathrm{NTPP}\ \ \texttt{"g3"}\big)$,
$\mu_3 = \big(\{?\mathrm{F} \to \mathrm{fire1}, ?\mathrm{S} \to \_\mathrm{R1}\},\ \ (\_\mathrm{R1}\ \ \mathrm{NTPP}\ \ \_\mathrm{R2})\ \ \wedge\ \ (\_\mathrm{R2}\ \ \mathrm{DC}\ \ \texttt{"g4"})\big)$,
$\mu_4 = \big(\{?\mathrm{F} \to \mathrm{fire1}, ?\mathrm{S} \to \_\mathrm{R1}\},\ \ \mathrm{true}\big)$

where g2 is as in Example 6.2, g3 is $\mathrm{x} \geq 0 \wedge \mathrm{x} \leq 10 \wedge \mathrm{y} \geq 0 \wedge \mathrm{y} \leq 10$, and g4 is $\mathrm{x} \geq 0 \wedge \mathrm{x} \leq 1 \wedge \mathrm{y} \geq 0 \wedge \mathrm{y} \leq 1$.

Definition 6.5. 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. If $W \subseteq dom(\mu)$, then 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.

We now introduce the notion of *compatible* conditional mappings as it is done in [75] for standard mappings. To take into account e-literals, we also define the notion of *possibly compatible* conditional mappings, which is more relaxed than compatibility.

Definition 6.6. 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 $\nu_1(x) = \nu_2(x)$. The conditional mappings μ_1 and μ_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.

Example 6.7. Mappings μ_1 and μ_2 from Example 6.4 are not compatible, while mappings μ_2 and μ_3 are. Further, all conditional mappings of Example 6.4 are pairwise possibly compatible.

If two conditional mappings are possibly compatible, then we can define their *join* as follows.

Definition 6.8. 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 conditional mapping (ν_3, θ_3) where:

i. $\nu_3(x) = \nu_1(x)$ for $x \in dom(\mu_1) \setminus dom(\mu_2)$ and $\nu_3(x) = \nu_2(x)$ for $x \in dom(\mu_2) \setminus dom(\mu_1)$.

ii. For each $x \in dom(\mu_1) \cap dom(\mu_2)$, $\nu_3(x) = \nu_1(x)$ if $\nu_1(x)$ is an e-literal, otherwise $\nu_3(x) = \nu_2(x)$.

iii. θ_3 is $\theta_1 \wedge \theta_2 \wedge \bigwedge_{x \in dom(\mu_1) \cap dom(\mu_2) \cap V_s} \xi_x$ where ξ_x is $\nu_1(x)$ EQ $\nu_2(x)$ if either $\nu_1(x)$ or $\nu_2(x)$ is an e-literal, otherwise ξ_x is $true$.

Example 6.9. If μ_1 and μ_2 are the conditional mappings of Example 6.4, then

$$\mu_1 \bowtie \mu_2 = \big(\{?\text{F} \to \text{fire1}, ?\text{S} \to \text{\_R1}\},\ \underbrace{true}_{\theta_1} \wedge \underbrace{\text{\_R1 NTPP "g3"}}_{\theta_2} \wedge \underbrace{\text{\_R1 EQ "g2"}}_{\xi_{?\text{S}}}\big).$$

For two sets of conditional mappings Ω_1 and Ω_2, the operation of join is now defined as follows:

$$\Omega_1 \bowtie \Omega_2 = \{\mu_1 \bowtie \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ are possibly compatible conditional mappings}\}.$$

The reader is invited to compare this definition with the definition of join of mappings for SPARQL [75]. The new aspect with conditional mappings is that due to the presence of e-literals, we have to anticipate the possibility that two mappings from Ω_1 and Ω_2 become compatible when e-literals are replaced by constants from C. We anticipate this case by adding relevant constraints to the condition of a mapping.

The operation of union is defined as in the standard case:

$$\Omega_1 \cup \Omega_2 = \{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\}.$$

We now define the operator of difference:

$$\begin{aligned}
&\Omega_1 \setminus \Omega_2 = \\
&\big\{\mu_1 \in \Omega_1 \mid \text{for all } \mu_2 \in \Omega_2,\ \mu_1 \text{ and } \mu_2 \text{ are not possibly compatible}\big\} \cup \\
&\big\{(\nu, \theta') \mid \mu = (\nu, \theta) \in \Omega_1 \text{ and } \mu_1 = (\nu_1, \theta_1), \ldots, \mu_n = (\nu_n, \theta_n) \in \Omega_2 \\
&\text{are such that } \mu, \mu_i \text{ are possibly compatible, for all } i, \text{ and every} \\
&\mu' \in \Omega_2 \setminus \{\mu_1, \ldots, \mu_n\} \text{ is not compatible with } \mu. \text{ In this case} \\
&\qquad \theta' \ = \ \theta \wedge \bigwedge_{1 \leq i \leq n} \left(\theta_i \supset \bigvee_{x \in dom(\mu) \cap dom(\mu_i) \cap V_s} \neg\big(\mu(x) \text{ EQ } \mu_i(x)\big) \right). \\
&\big\}.
\end{aligned}$$

In the above set, symbol $\supset$ denotes logical implication.

The reader is invited to compare this definition with the definition of difference by Pérez et al. [75]. In [75] the qualifying condition for a mapping μ from Ω_1 to be in the result of the difference $\Omega_1 \setminus \Omega_2$ is to be incompatible with all the mappings of Ω_2. That qualifying condition is maintained in the above

definition for the set of mappings from Ω_1 that do not map any special variable to an e-literal (the first set of the union operator that makes up the set $\Omega_1 \setminus \Omega_2$)[5]. However, in the context of RDF$^{\text{i}}$ such qualifying condition should be broadened to accommodate the case of mapping μ being possibly compatible with some mappings from Ω_2 and incompatible with the rest of the mappings from Ω_2 (the second set of the union operator that makes up the set $\Omega_1 \setminus \Omega_2$), i.e., accommodate the presence of e-literals. In this latter case, we need to capture the conditions under which each possibly compatible pair of mappings is made incompatible and impose them as further constraints to the condition of a mapping μ of the set $\Omega_1 \setminus \Omega_2$.

Let us now point out a subtlety in the formula of the previous definition that is used to capture the conditions for making a possibly compatible pair of mappings $\mu \in \Omega_1$, $\mu_i \in \Omega_2$ incompatible. This is the use of logical implication between the condition θ_i of mapping μ_i and the constraint under which this pair of mappings becomes incompatible, i.e., $\neg\big(\mu(x)\ \text{EQ}\ \mu_i(x)\big)$. The logical implication serves only to mark that this constraining should be taken into account only when the condition of mapping θ_i is satisfiable. Otherwise, mapping μ_i should not be affecting the result of the difference. Then, logical implication guarantees exactly that.

Example 6.10. Let $\Omega_1 = \{\mu_{11}, \mu_{12}\}$, $\Omega_2 = \{\mu_{21}, \mu_{22}\}$ be sets of conditional mappings such that

$\mu_{11} = \big(\{?\text{F} \to \text{fire1}, ?\text{S} \to \_\text{R1}\}, \_\text{R1}\ \text{NTPP}\ \texttt{"g3"}\big)$,
$\mu_{12} = \big(\{?\text{F} \to \text{fire1}, ?\text{S} \to\ \texttt{"g2"}\}, \_\text{R1}\ \text{PO}\ \texttt{"g3"}\big)$,
$\mu_{21} = \big(\{?\text{F} \to \text{fire2}\},\ \text{true}\big)$,
$\mu_{22} = \big(\{?\text{F} \to \text{fire1}, ?\text{S} \to\ \texttt{"g2"}\}, \text{true}\big)$.

Then, $\Omega_1 \setminus \Omega_2 = \{\mu\}$ where μ is constructed from $\mu_{11} \in \Omega_1$:

$\mu = \big(\{?\text{F} \to \text{fire1}, ?\text{S} \to \_\text{R1}\}, (\_\text{R1}\ \text{NTPP}\ \texttt{"g3"}) \wedge \big(\text{true} \supset \neg(\_\text{R1}\ \text{EQ}\ \texttt{"g2"})\big)\big)$.

[5]Observe that if two conditional mappings are not possibly compatible, then they are not compatible either.

The operation of left-outer join is defined as in the standard case [75]:

$$\Omega_1 \mathbin{⟕} \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2).$$

6.2. Graph Pattern Evaluation

Graph patterns in RDF$^{\text{i}}$ are like in standard SPARQL [75] and are built recursively from *triple patterns* and the operators AND, UNION, OPT, and FILTER. However, the definition of triple patterns in RDF$^{\text{i}}$ differs from the one in standard SPARQL and is as follows.

Definition 6.11. A *triple pattern* is an element of the set $(I \cup V_n) \times (I \cup V_n) \times (I \cup L \cup C \cup V)$.

Note that 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 fresh 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. Let $\mu = (\nu, \theta)$ be a conditional mapping and p a triple pattern such that $var(p) \subseteq dom(\mu)$. We denote by $\mu(p)$ the conditional triple (t, θ) such that t is obtained from p by replacing each variable $x \in var(p)$ by $\nu(x)$.

An important fragment of SPARQL we consider in this work is the so-called *well-designed* graph patterns originally identified by Pérez et al. [75].

Definition 6.12. Let P be a graph pattern in the AND-FILTER-OPT fragment of SPARQL. Then P is *well-designed* if (1) P is safe, i.e., for every sub-pattern (P_1 FILTER R) of P, it holds that $var(R) \subseteq var(P_1)$, and (2) for every sub-pattern $P' = (P_1$ OPT $P_2)$ of P and variable $?X$, if $?X$ occurs both inside P_2 and outside P', then it also occurs in P_1.

Pérez et al. [75] and Arenas and Pérez [10] identified in well-designed graph patterns unique and interesting properties that make query evaluation more efficient in contrast to what would get without the syntactic restrictions imposed on the graph patterns by the above definition. In the context of this work,

well-designed graph patterns play an important role because it is one of the two fragments of SPARQL graph patterns for which the SPARQL query evaluation as it is developed in this section is correct given the semantics of RDF$^{\text{i}}$ introduced in Section 5. The correctness of SPARQL query evaluation is studied in Section 8 where representation systems for RDF$^{\text{i}}$ are investigated.

Based on the operations on sets of conditional mappings from the previous subsection, graph pattern evaluation in RDF$^{\text{i}}$ can now be defined exactly as in standard SPARQL for RDF graphs [75] except for the case of evaluating a triple pattern. For the sake of readability, the evaluation of FILTER graph patterns is considered later in this section.

Definition 6.13. Let $D = (G, \phi)$ be an RDF$^{\text{i}}$ database. Evaluating a graph pattern P over database D is denoted by $[\![P]\!]_D$ and is defined recursively as follows:

1. If P is the triple pattern (s, p, o) then

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

2. If P is P_1 AND P_2 then $[\![P]\!]_D = [\![P_1]\!]_D \bowtie [\![P_2]\!]_D$.

3. If P is P_1 UNION P_2 then $[\![P]\!]_D = [\![P_1]\!]_D \cup [\![P_2]\!]_D$.

4. If P is P_1 OPT P_2 then $[\![P]\!]_D = [\![P_1]\!]_D \mathbin{⟕} [\![P_2]\!]_D$.

In Definition 6.13 (1), the first set making up $[\![P]\!]_D$ accommodates the case in which evaluation can be done as in standard SPARQL using triple pattern matching. The second set accommodates the case in which the triple pattern involves a literal o from the set C and the graph contains a conditional triple with an e-literal $\_l$ from U in the object position. We catch a possible match 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). In either case of Definition 6.13 (1), since the triples in the database are conditional, their conditions become parts of the conditions of the mappings in the answer.

Example 6.14. Consider the following RDF$^{\text{i}}$ database D

$$\Big(\big\{((\text{hotspot1}, \text{a}, \text{Hotspot}), \text{true}),\ \ ((\text{fire1}, \text{a}, \text{Fire}), \text{true}),$$
$$((\text{hotspot1}, \text{correspondsTo}, \text{fire1}), \text{true}),$$
$$((\text{fire1}, \text{occurredIn}, \_\text{R1}), \text{true}),$$
$$((\text{fire2}, \text{occurredIn}, \texttt{"g1"}), \text{true})\big\},$$
$$(\_\text{R1 NTPP "g1"} \wedge \_\text{R1 NTPP "g5"}) \vee \_\text{R1 PO "g6"}\Big)$$

where g1 is as in Example 4.2, g5 is $\text{x} \geq 10 \wedge \text{x} \leq 21 \wedge \text{y} \geq 12 \wedge \text{y} \leq 17$, and g6 is $\text{x} \geq 2 \wedge \text{x} \leq 6 \wedge \text{y} \geq 4 \wedge \text{y} \leq 8$. Next we show how the evaluation of two triple patterns over the database D can be done.

a) Evaluation of triple pattern (?F, occurredIn, "g1").

$$[\![(\text{?F, occurredIn, "g1"})]\!]_{\text{D}} =$$
$$\Big\{\big(\{\text{?F} \to \text{fire2}\},\ \text{true}\big)\Big\} \cup \Big\{\big(\{\text{?F} \to \text{fire1}\}, \_\text{R1 EQ "g1"}\big)\Big\}$$

b) Evaluation of triple pattern (?F, occurredIn, "g2") where g2 is as in Example 6.2.

$$[\![(\text{?F, occurredIn, "g2"})]\!]_{\text{D}} = \emptyset \ \cup\ \Big\{\big(\{\text{?F} \to \text{fire1}\}, \_\text{R1 EQ "g2"}\big)\Big\}$$

These examples showcase the two possibilities of triple pattern matching that arise in Definition 6.13 (1) when the triple pattern involves a literal from the set C. Case (a) involves both kinds of matchings (both sets of Definition 6.13 (1) that make up $[\![P]\!]_D$), while case (b) involves a matching derived from the second set of Definition 6.13 (1) that makes up $[\![P]\!]_D$.

Example 6.15. Let us now give an example of an evaluation of graph pattern P_1 AND P_2 over the database D of Example 4.2, where P_1, P_2 are the triple patterns $(\text{?F}, \text{a}, \text{Fire})$ and $(\text{?F}, \text{occurredIn}, \text{?R})$ respectively. According to the above definition, we have:

$[\![(\mathrm{P}_1 \text{ AND } \mathrm{P}_2)]\!]_\mathrm{D} = [\![\mathrm{P}_1]\!]_\mathrm{D} \bowtie [\![\mathrm{P}_2]\!]_\mathrm{D} =$
$[\![(?\mathrm{F},\ \mathrm{a},\ \mathrm{Fire})]\!]_\mathrm{D} \bowtie [\![(?\mathrm{F},\ \mathrm{occurredIn},\ ?\mathrm{R})]\!]_\mathrm{D} =$
$\Big\{\big(\{?\mathrm{F} \to \mathrm{fire1}\}, \mathrm{true}\big)\Big\} \bowtie \Big\{\big(\{?\mathrm{F} \to \mathrm{fire1}, ?\mathrm{R} \to \_\mathrm{R1}\}, \mathrm{true}\big)\Big\} =$
$\Big\{\big(\{?\mathrm{F} \to \mathrm{fire1}, ?\mathrm{R} \to \_\mathrm{R1}\}, \mathrm{true}\big)\Big\}.$

Evaluation of both triple patterns P_1, P_2 is done as in standard SPARQL, but here conditions of matched triples have to be transferred to the respective answer, i.e., we have conditional mappings.

Let us now consider the operator FILTER. It is natural to allow FILTER graph patterns to contain an $\mathcal{L}$-constraint as expression that constrains query variables. When $\mathcal{L}$ is PCL, examples of such expressions are the atomic PCL-constraints ?X NTPP ?Y, ?X EQ "$\mathrm{x} \geq 1 \wedge \mathrm{x} \leq 2 \wedge \mathrm{y} \geq 1 \wedge \mathrm{y} \leq 2$", or any Boolean combination of them[6]. The extension of FILTER to the case of having expressions containing also other built-in conditions of standard SPARQL [75] is easy to define and is omitted. However, care must be taken so that $\mathcal{L}$-constraints and built-in conditions are not intermixed.

The evaluation of FILTER graph patterns involving $\mathcal{L}$-constraints can now be defined as follows. Notice that the evaluation does not check for satisfaction of the constraints as in standard SPARQL [75], but simply imposes these constraints on the mappings that are in the answer of the graph pattern involved.

Definition 6.16. Given an RDF$^\mathrm{i}$ database $D = (G, \phi)$, a graph pattern P and an $\mathcal{L}$-constraint R, we have:

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

where $\nu(R)$ denotes the application of e-mapping ν to condition R, i.e., the formula of $\mathcal{L}$-constraints obtained from R when each special variable x of R which also belongs to $dom(\nu)$ is replaced by $\nu(x)$.

[6]Note that we have overloaded the concept of $\mathcal{L}$-constraints so that special variables are substituted for variables of the constraint language $\mathcal{L}$.

The following example illustrates the definition and shows that the purpose of constraint $\nu(R)$ is to deal in a uniform way with the case that the object of a triple is a constant from C or an e-literal from U. Notice that $\nu(R)$ is required because mappings in our case can contain variables with e-literals as values, thus we might not be able to deduce their satisfaction yet. In an implementation, one can also simplify constraints at this stage; such issues are beyond the scope of this paper.

Example 6.17. Based on the evaluation of the graph pattern of Example 6.15, the evaluation of the graph pattern $\big((P_1 \text{ AND } P_2) \text{ FILTER } R\big)$, where R is the PCL-constraint (?R NTPP "g5"), is the following:

$$[\![(\text{P}_1 \text{ AND } \text{P}_2) \text{ FILTER } \text{R}]\!]_{\text{D}} = \Big\{\big(\{?\text{F} \to \text{fire1}, ?\text{R} \to \_\text{R1}\}, \_\text{R1 NTPP "g5"}\big)\Big\}.$$

6.3. Evaluation of SPARQL Queries

Having defined the evaluation of graph patterns, in this section we introduce the SELECT and CONSTRUCT query forms of SPARQL and study their evaluation in RDF$^{\text{i}}$.

Definition 6.18 (Pérez et al. [74])**.** A SELECT query is a pair (W, P) where W is a set of variables from the set V and P is a graph pattern.

Example 6.19. Let us consider the following query over the database of Example 4.2: "Find all fires that have occurred in a region which is a non-tangential proper part of the rectangle defined by the points $(10, 12)$ and $(21, 17)$". This query can be expressed as

$$\big(\{?\text{F}\}, (?\text{F}, \text{a}, \text{Fire}) \text{ AND } (?\text{F}, \text{occurredIn}, ?\text{R}) \text{ FILTER } (?\text{R NTPP "g5"})\big)$$

where g5 is as in Example 6.14.

Next we define the notion of an *answer* to a SELECT query. In contrast to SELECT queries over RDF graphs, SELECT queries over RDF$^{\text{i}}$ databases have answers that consist of conditional mappings so they might be harder to understand (see Example 6.21 below).

Definition 6.20. Let $q = (W, P)$ be a SELECT query. The *answer* to q over an RDF$^{\text{i}}$ database $D = (G, \phi)$ (in symbols $[\![q]\!]_D$) is the pair (Ω, ϕ) where Ω is the set of conditional mappings $\{\mu_{|W} \mid \mu \in [\![P]\!]_D\}$ and ϕ the global constraint of D.

In contrast to standard SPARQL, answers to SELECT queries in RDF$^{\text{i}}$ are pairs of a set of mappings and a global constraint rather than sets of mappings. The reason is that conditional mappings in the answer to a query might contain e-literals. These e-literals are constrained by the global constraint ϕ, which is also included in the answer.

Example 6.21. The answer to the query from Example 6.19 can be obtained from the evaluation of the respective graph pattern from Example 6.17. The answer is the pair (Ω, ϕ) where the set of conditional mappings Ω and the global constraint ϕ are as follows:

$$\Omega = \Big\{\big(\{?\text{F} \to \text{fire1}\}, \text{\_R1 NTPP "g5"}\big)\Big\}, \quad \phi = \text{\_R1 NTPP "g1"}.$$

This answer is conditional. Because the information in the database of Example 4.2 is indefinite (the exact geometry of _R1 is not known), we cannot say for sure whether fire1 satisfies the requirements of the query. These requirements are satisfied under the condition given in the above mapping and the global constraint ϕ.

Let us now introduce the notion of a *template* and define the CONSTRUCT query form.

Definition 6.22. A *template* E is a finite subset of the set $(T \cup V) \times (I \cup V) \times (T \cup V)$.

The elements of a template are like triple patterns but blank nodes and e-literals are also allowed in the subject and object positions. We denote by $var(E)$ and $blank(E)$ the set of variables and set of blank nodes appearing in the elements of E respectively. Templates are used to specify the graph that results from the evaluation of a CONSTRUCT query.

Definition 6.23. A CONSTRUCT query is a pair (E, P), where E is a template and P a graph pattern.

Example 6.24. Let us consider the query of Example 6.19. A new version of this query using the CONSTRUCT query form is:

$$\Big(\big\{(?\mathrm{F}, \mathrm{a}, \mathrm{Fire})\big\}, (?\mathrm{F}, \mathrm{a}, \mathrm{Fire})\ \text{AND}\ (?\mathrm{F}, \mathrm{occurredIn}, ?\mathrm{R})\ \text{FILTER}\ (?\mathrm{R}\ \text{NTPP}\ \texttt{"g5"})\Big).$$

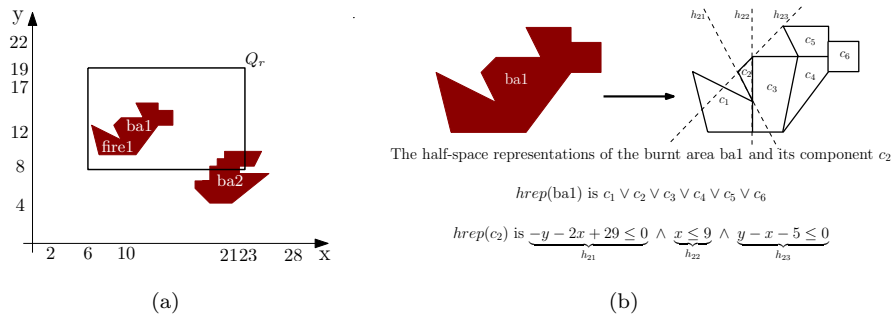
If $\mu = (\nu, \theta)$ is a conditional mapping and E a template, then we denote by $\mu(E)$ the application of the conditional mapping μ to template E. The effect of expression $\mu(E)$ is the replacement of every variable x of $var(E) \cap dom(\mu)$ by $\nu(x)$ and the attachment of constraint θ to every element of E.

Example 6.25. Let us consider the template $E = \big\{(?\mathrm{F}, \mathrm{a}, ?\mathrm{Z}), (?\mathrm{F}, \mathrm{occurredIn}, ?\mathrm{S})\big\}$ and mapping μ_4 from Example 6.4. The result of applying μ_4 to E is the set $\big\{\big((\mathrm{fire1}, \mathrm{a}, ?\mathrm{Z}), \mathrm{true}\big), \big((\mathrm{fire1}, \mathrm{occurredIn}, \_\mathrm{R1}), \mathrm{true}\big)\big\}$.

Notice that the definition above does not require a conditional mapping to share any variables with the template to which it is applied. As a consequence, the first element of $\mu_4(E)$ is not a valid conditional triple, i.e., the triple part of it is not an element of the set $(I \cup B) \times I \times T$ as Definition 4.1 for e-triples requires. Such conditional triples are dropped from the answers to CONSTRUCT queries (see Definition 6.27 below).

Next we define the notion of the answer to a CONSTRUCT query. The definition extends the specification of standard SPARQL [76] to account for the RDF$^{\mathrm{i}}$ framework and follows the formal approach of Pérez et al. [74]. Before we give the definition, we need to introduce the notion of a *renaming function*.

Definition 6.26. Let E be a template, P a graph pattern, and $D = (G, \phi)$ an RDF$^{\mathrm{i}}$ database. The set $\{f_\mu \mid \mu \in [\![P]\!]_D\}$ is a set of *renaming functions* for E and $[\![P]\!]_D$ if the following properties are satisfied: 1) the domain of every function f_μ is $blank(E)$ and its range is a subset of $(B \setminus blank(G))$, 2) every function f_μ is one-to-one, and 3) for every pair of distinct mappings $\mu_1, \mu_2 \in [\![P]\!]_D$, f_{μ_1}, f_{μ_2} have disjoint ranges.



(a) (b)

Figure 2: (a) Spatial configuration of the burnt areas of Example 6.29 and (b) half-space representation of the polygon corresponding to the burnt area identified as ba1

The application of a renaming function f_μ to a template E is denoted by $f_\mu(E)$ and results in renaming the blank nodes of E according to f_μ.

Definition 6.27. 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

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

In the above definition, renaming functions are used to ensure that new blank nodes are created for each conditional mapping μ. The intersection with the set $(I \cup B) \times I \times T$ makes sure that no illegal conditional triples are returned as answers (see Example 6.25 above).

Example 6.28. The answer to the CONSTRUCT query from Example 6.24 can be obtained from the evaluation of the respective graph pattern from Example 6.17. The answer is the following RDF$^{\text{i}}$ database:

$\big(\{((\text{fire1}, \text{a}, \text{Fire}),\ \_\text{R1 NTPP "g5"})\},\ \ \_\text{R1 NTPP "g1"}\big)$.

Before closing this section, we give a complete and last example that demonstrates most of the definitions of this section which are required for evaluating a CONSTRUCT query over an RDF$^{\text{i}}$ database.

Example 6.29. Let us consider the following RDF$^{\mathrm{i}}$ database $D = (G, \phi)$ that augments the one of Example 4.2 with a new set of conditional triples. These new triples assert two burnt areas with identifiers ba1 and ba2 and exact geometries "g7" and "g8" where g7 and g8 denote the half-space representation of the polygons corresponding to ba1 and ba2 respectively. For instance, the half-space representation for ba1 is given in Figure 2(b). We also re-use the polygonal constant "g1" from Example 4.2. Constants "g7" and "g8" participate in the global constraint in two equality PCL-constraints with the e-literals _R2 and _R3 respectively. Notice also that the e-literal _R1 corresponding to fire fire1 has now been further constrained to be equal to _R2, i.e., the region corresponding to the burnt area ba1. The spatial configuration is shown in Figure 2(a).

$\big(\{\ ((\text{hotspot1}, \text{a}, \text{Hotspot}), \text{true}),\ \ ((\text{fire1}, \text{a}, \text{Fire}), \text{true}),$
$((\text{hotspot1}, \text{correspondsTo}, \text{fire1}), \text{true}),$
$((\text{fire1}, \text{occurredIn}, \text{\_R1}), \text{true}),$
$((\text{ba1}, \text{a}, \text{BurntArea}), \text{true}),\ \ ((\text{ba1}, \text{hasGeometry}, \text{\_R2}), \text{true}),$
$((\text{ba2}, \text{a}, \text{BurntArea}), \text{true}),\ \ ((\text{ba2}, \text{hasGeometry}, \text{\_R3}), \text{true})\ \},$
$\text{\_R1 NTPP "g1"} \wedge \text{\_R2 EQ "g7"} \wedge \text{\_R3 EQ "g8"} \wedge \text{\_R1 EQ \_R2}\big)$

Updates such as the above are customarily made by national agencies in charge of crisis management as a means of assessing the damage caused by a fire event after it has been completed. In the jargon of fire monitoring and management this process is known as "burnt scar mapping".

Let us consider the following CONSTRUCT query over the above database: "Find all fires and burnt areas that have been identified in a region which is a non-tangential proper part of the rectangle defined by the points (6, 8) and (23, 19)" (rectangle Q_r in Figure 2(a)). This query is expressed as follows:

$\Big(\big\{(\text{?F}, \text{hasGeometry}, \text{?R1}), (\text{?BA}, \text{hasGeometry}, \text{?R2})\big\},$
$(\text{?F}, \text{a}, \text{Fire})\ \ \text{AND}\ \ (\text{?F}, \text{occurredIn}, \text{?R1})\ \ \text{AND}$
$(\text{?BA}, \text{a}, \text{BurntArea})\ \ \text{AND}\ \ (\text{?BA}, \text{hasGeometry}, \text{?R2})$
$\text{FILTER (?R1 NTPP "g1"} \wedge \text{?R2 NTPP "g1")}\Big).$

For brevity, we will refer to the template of the above query, the triple patterns and the condition of the FILTER operator as E, $P_1, \ldots, P_4$, and R respectively. We assume that the evaluation of graph pattern expressions is done from left to right, i.e., $\Big(\big(((P_1 \text{ AND } P2) \text{ AND } P_3) \text{ AND } P_4\big) \text{ FILTER } R\Big)$. Then, we will use P to refer to the AND graph pattern expression composed of P_1, P_2, P_3, and P_4.

The evaluation of triple patterns results in the following sets of conditional mappings (see Definition 6.13 (1)).

$\Omega_1 = [\![P_1]\!]_D = \big\{\big(\{?\text{F} \to \text{fire1}\}, \text{true}\big)\big\}$

$\Omega_2 = [\![P_2]\!]_D = \big\{\big(\{?\text{F} \to \text{fire1}, ?\text{R1} \to \_\text{R1}\}, \text{true}\big)\big\}$

$\Omega_3 = [\![P_3]\!]_D = \big\{\big(\{?\text{BA} \to \text{ba1}\}, \text{true}\big), \big(\{?\text{BA} \to \text{ba2}\}, \text{true}\big)\big\}$

$\Omega_4 = [\![P_4]\!]_D = \big\{\big(\{?\text{BA} \to \text{ba1}, ?\text{R2} \to \_\text{R2}\}, \text{true}\big), \big(\{?\text{BA} \to \text{ba2}, ?\text{R2} \to \_\text{R3}\}, \text{true}\big)\ \big\}$

According to Definition 6.13 (2) the evaluation of AND graph pattern expressions is done by joining the above sets of mappings as follows $\big((\Omega_1 \bowtie \Omega_2) \bowtie \Omega_3\big) \bowtie \Omega_4\big)$. The result, Ω_5, is shown below.

$$\begin{aligned}\Omega_5 = \big\{&\big(\{?\text{F} \to \text{fire1}, ?\text{R1} \to \_\text{R1}, ?\text{BA} \to \text{ba1}, ?\text{R2} \to \_\text{R2}\}, \text{true}\big),\\ &\big(\{?\text{F} \to \text{fire1}, ?\text{R1} \to \_\text{R1}, ?\text{BA} \to \text{ba2}, ?\text{R2} \to \_\text{R3}\}, \text{true}\big)\big\}\end{aligned}$$

Having evaluated the graph pattern P, the evaluation of the graph pattern P FILTER R just attaches the constraint $\nu(R)$ to every conditional mapping (ν, true) of the set Ω_5 above (see Definition 6.16). The resulting set of conditional mappings is the following:

$$\begin{aligned}\Omega = \big\{&\big(\{?\text{F} \to \text{fire1}, ?\text{R1} \to \_\text{R1}, ?\text{BA} \to \text{ba1}, ?\text{R2} \to \_\text{R2}\},\ \theta_1 \wedge \theta_2\big),\\ &\big(\{?\text{F} \to \text{fire1}, ?\text{R1} \to \_\text{R1}, ?\text{BA} \to \text{ba2}, ?\text{R2} \to \_\text{R3}\},\ \theta_1 \wedge \theta_3\big)\big\}\end{aligned}$$

where $\theta_\text{i} = \_\text{Ri NTPP "g1"}$ for $\text{i} = 1, 2, 3$.

As a last step for computing the answer to the CONSTRUCT query of this example is the application of the conditional mappings of the set Ω to the template E (see Definition 6.27). This leads to the RDF$^\text{i}$ database $D_q = (G_q, \phi)$ shown below.

$\big(\{((\text{fire1}, \text{hasGeometry}, \text{\_R1}),\ \theta_1 \wedge \theta_2),\ ((\text{ba1}, \text{hasGeometry}, \text{\_R2}),\ \theta_1 \wedge \theta_2),$
$((\text{fire1}, \text{hasGeometry}, \text{\_R1}),\ \theta_1 \wedge \theta_3),\ ((\text{ba2}, \text{hasGeometry}, \text{\_R3}),\ \theta_1 \wedge \theta_3)\ \},$
$\text{\_R1 NTPP "g1"} \ \wedge\ \text{\_R2 EQ "g7"} \ \wedge\ \text{\_R3 EQ "g8"} \ \wedge\ \text{\_R1 EQ \_R2}\big)$

7. Monotonicity of SPARQL

In this section, we define the notion of monotonicity for SPARQL queries evaluated over RDF graphs and prove that two important fragments of SPARQL are monotone. The first fragment employs graph patterns built using operators AND, UNION, and FILTER, while the second employs the well-designed graph patterns introduced in Section 6. As in Section 6, our only addition to standard SPARQL is the extension of FILTER operator with another kind of conditions that are constraints of $\mathcal{L}$. Due to this extension, the constants of $\mathcal{L}$ should also be allowed to appear in RDF graphs. This set of constants corresponds to the typed literals of RDF$^{\text{i}}$ from the set C defined in Section 4. In the following, we denote by T^C the terms of RDF that will be the union of the sets I, B, L, and C. Therefore, T^C differs from T, i.e., the terms of RDF$^{\text{i}}$, in that it does not include e-literals from the set U.

Before proceeding to the details, we introduce some notation that will be useful in the remainder of this work.

Notation 1. 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 a class $\mathcal{F}$. Class AUF will denote the graph pattern expressions built using operators AND, UNION, and FILTER, while class WD will denote the well-designed graph patterns. We will denote by $\mathcal{Q}^{C'}_{\mathcal{F}}$ all CONSTRUCT queries without blank nodes in their templates.

The following definition introduces the concept of *monotonicity* for SPARQL queries which is equivalent to the notion of monotonicity for graph patterns as defined by Arenas and Pérez [10].

Definition 7.1. A fragment $\mathcal{Q}$ of SPARQL is *monotone* if for every $q \in \mathcal{Q}$ and RDF graphs G and H such that $G \subseteq H$, we have $[\![q]\!]_G \subseteq [\![q]\!]_H$.

Before proving which fragments of SPARQL queries are monotone, we recall the relevant notions of *subsumption* for standard mappings [75] and *weak monotonicity* for graph patterns [10].

Definition 7.2 (Subsumption of Mappings [75])**.** Let μ_1, μ_2 be mappings. We say that μ_1 *is subsumed by* μ_2, denoted by $\mu_1 \preceq \mu_2$, if $dom(\mu_1) \subseteq dom(\mu_2)$ and $\mu_1(x) = \mu_2(x)$ for every $x \in dom(\mu_1)$. Let Ω_1, Ω_2 be sets of mappings. We say that Ω_1 *is subsumed by* Ω_2, denoted by $\Omega_1 \sqsubseteq \Omega_2$, if for every $\mu_1 \in \Omega_1$ there exists a mapping $\mu_2 \in \Omega_2$ such that $\mu_1 \preceq \mu_2$.

Informally, if a mapping μ subsumes a mapping μ', then μ may contain additional information to μ', i.e., it may map additional variables to RDF terms.

Example 7.3. Let us consider the following mappings:

$$\mu_1 = \{\ ?\text{F} \rightarrow \text{fire1}\ \},\ \ \mu_2 = \{\ ?\text{F} \rightarrow \text{fire1}, ?\text{S} \rightarrow\ \text{"g2"}\ \}.$$

Mapping μ_1 is subsumed by mapping μ_2, i.e., $\mu_1 \preceq \mu_2$.

Definition 7.4 (Weak Monotonicity [10])**.** Let P be a graph pattern of SPARQL. P is said to be *weakly monotone* if for every pair G, H of RDF graphs such that $G \subseteq H$, it is $[\![P]\!]_G \sqsubseteq [\![P]\!]_H$.

The following theorem summarizes the properties of graph patterns with respect to monotonicity using the notation introduced above.

Theorem 7.5 (Arenas and Pérez [10])**.** The class of AUF graph patterns is monotone. The class of WD graph patterns is weakly monotone.

We stress here that our extension of the FILTER conditions to include $\mathcal{L}$-constraints does not affect in any way the proof of the above theorem. Testing whether such conditions are satisfied by a mapping is similar to checking whether a built-in condition is satisfied with the only exception that one might need a specialized algorithm for the new kind of conditions. For example, when the

constraint language is PCL, satisfaction could be checked by an algorithm of computational geometry, since in that case we need to check whether a certain topological relation holds between two geometrical objects in $\mathbb{Q}^2$. This is what is done in the geospatial RDF store Strabon [50], which implements the geospatial and temporal extensions of RDF and SPARQL, named stRDF and stSPARQL respectively, proposed by Koubarakis and Kyzirakos [49].

Having Theorem 7.5, it is now easy to prove Proposition 7.6 below, which gives us some fragments of SPARQL that are monotone.

Proposition 7.6. *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.

Proof. *a*) and *b*) The results are trivial extensions of relevant results by Arenas and Pérez [10].
c) Consider a query $q = (E, P) \in \mathcal{Q}^{C}_{AUF}$ and let G, H be two RDF graphs such that $G \subseteq H$. According to Definition 4.6 of CONSTRUCT for RDF graphs as given by Pérez et al. [74] we have

$$[\![q]\!]_G = \bigcup_{\mu \in [\![P]\!]_G} \{\mu(f_\mu(E)) \cap ((I \cup B) \times I \times T^C)\}, \tag{1}$$

$$[\![q]\!]_H = \bigcup_{\mu' \in [\![P]\!]_H} \{\mu'(f_{\mu'}(E)) \cap ((I \cup B) \times I \times T^C)\}. \tag{2}$$

From the monotonicity property of AUF graph patterns (Theorem 7.5), we have that $[\![P]\!]_G \subseteq [\![P]\!]_H$. Therefore, all mappings μ appearing in the union expression of formula (1) appear also in the union expression of formula (2). Hence, if the sets making up the union in formulae (1) and (2) are the same, then we shall get the required relation for monotonicity, that is, $[\![q]\!]_G \subseteq [\![q]\!]_H$.

Let us first consider CONSTRUCT queries without blank nodes in their template. Then, the renaming functions do not have any effect on the templates, therefore, the sets in formulae (1) and (2) are the same for same mappings. Thus, $[\![q]\!]_G \subseteq [\![q]\!]_H$.

The same does not hold if we allow blank nodes to appear in templates.

The reason is that renaming functions rename blank nodes to fresh ones with respect to the underlying RDF graph over which a graph pattern is evaluated. Therefore, a renaming function used in formula (1) could have possibly renamed a blank node to a fresh one regarding G, but not a fresh one regarding H, i.e., that blank node could have been already in H.

d) Consider a query $q = (E, P) \in \mathcal{Q}_{WD}^{C'}$ and let G, H be two RDF graphs such that $G \subseteq H$. Let also $[\![q]\!]_G$ and $[\![q]\!]_H$ be as in formulae (1) and (2) respectively. Since the template E does not contain blank nodes, we can omit the renaming functions from these expressions and get

$$[\![q]\!]_G = \bigcup_{\mu \in [\![P]\!]_G} \{\mu(E) \cap ((I \cup B) \times I \times T^C)\},$$
$$[\![q]\!]_H = \bigcup_{\mu' \in [\![P]\!]_H} \{\mu'(E) \cap ((I \cup B) \times I \times T^C)\}.$$

Since $P \in WD$, by Theorem 7.5, P is weakly monotone. Therefore, $[\![P]\!]_G \sqsubseteq [\![P]\!]_H$. Hence, for every mapping $\mu \in [\![P]\!]_G$ there exists mapping $\mu' \in [\![P]\!]_H$ such that $\mu \preceq \mu'$. This means that μ and μ' map the common variables of their domains to the same RDF terms. Hence, if a mapping $\mu \in [\![P]\!]_G$ produces triple t in $[\![q]\!]_G$, that triple is also produced in $[\![q]\!]_H$ by a mapping $\mu' \in [\![P]\!]_H$ such that $\mu \preceq \mu'$. Thus, $[\![q]\!]_G \subseteq [\![q]\!]_H$. □

It has been noted by Angles and Gutierrez [6] 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, Proposition 7.6 *d*) shows 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, which can be proved to be weakly monotone for the class of well-designed patterns, but not monotone.

8. Representation Systems for RDF$^{\text{i}}$

Let us now recall the semantics of RDF$^{\text{i}}$: $Rep(D)$ 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 all RDF graphs of $Rep(D)$ as follows:

$$[\![q]\!]_{Rep(D)} = \big\{ [\![q]\!]_G \mid G \in Rep(D) \big\}.$$

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 the following formula:

$$Rep([\![q]\!]_D) = [\![q]\!]_{Rep(D)}. \tag{3}$$

Formula (3) 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 having the need to apply the query on all possible RDF graphs. Expression $[\![q]\!]_D$ can be computed using the algebra of Section 6 above. But can the algebra of Section 6 always compute such a database $[\![q]\!]_D$ representing $[\![q]\!]_{Rep(D)}$? In other words, can we prove (3) for all SPARQL queries considered in Section 6? The answer is *no* in general. The following example modeled after [31] illustrates this negative fact.

Example 8.1. Consider the RDF$^{\text{i}}$ database D with the triple (s, p, o) and a global constraint equal to *true*, and a CONSTRUCT query q over D that constructs an RDF graph out of triples having s as the subject. Then, triple (s, p, o) and nothing else is in the resulting database $[\![q]\!]_D$. However, equation (3) 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 $g \in [\![q]\!]_{Rep(D)}$.

Note that the above counterexample to (3) 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 (3) has to be relaxed[7].

In the rest of this section we follow the approach of incomplete relational databases [41, 30] and show how (3) can be weakened. The key concept for achieving this is the concept of *certain answers* that we define below. Before proceeding to the definition, we need to introduce some useful notation.

Notation 2. 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 Pérez et al. [75] to the case of sets of RDF graphs, will denote the element-wise evaluation of q over $\mathcal{G}$, that is, $[\![q]\!]_{\mathcal{G}} = \{[\![q]\!]_G \mid G \in \mathcal{G}\}$.

The following definition of *certain answers* extends the corresponding definition of Section 3.1 of Arenas and Pérez [10] by applying it to a more general incomplete information setting.

Definition 8.2. Let q be a query and $\mathcal{G}$ a set of RDF graphs. The *certain answers* to q over $\mathcal{G}$ is the set $\bigcap [\![q]\!]_{\mathcal{G}}$.

Although the concept of certain answers is defined above for sets of RDF graphs, it is natural to refer to the certain answers of queries over RDF$^{\text{i}}$ databases without introducing any further formal definition. This should not cause any confusion since the semantics of an RDF$^{\text{i}}$ database makes the concept of an RDF$^{\text{i}}$ database equivalent to a set of RDF graphs.

Example 8.3. Let us consider the following query over the database of Example 4.2: "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 $(28, 22)$" (see Fig-

[7] If the CWA is adopted, we can prove (3) using similar techniques to the ones that enable us to prove Theorem 8.15 below.

ure 1 in Section 2). The certain answers to this query is the set of mappings $\{\{?\text{F} \to \text{fire1}\}\}$.

Given a fixed fragment $\mathcal{Q}$ of SPARQL, two sets of RDF graphs cannot be distinguished by $\mathcal{Q}$ if they give the same certain answers to every query in $\mathcal{Q}$. The next definition formalizes this fact using the concept of $\mathcal{Q}$-*equivalence*.

Definition 8.4. Let $\mathcal{Q}$ be a fragment of SPARQL, and $\mathcal{G}$, $\mathcal{H}$ two sets of RDF graphs. $\mathcal{G}$ and $\mathcal{H}$ are called $\mathcal{Q}$-*equivalent* (denoted by $\mathcal{G} \equiv_{\mathcal{Q}} \mathcal{H}$) if they give the same certain answers to every query in the language, that is, $\bigcap [\![q]\!]_{\mathcal{G}} = \bigcap [\![q]\!]_{\mathcal{H}}$ for all $q \in \mathcal{Q}$.

We can now define the notion of a *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*.

Definition 8.5. Let $\mathcal{D}$ be the set of all RDF$^{\text{i}}$ databases, $\mathcal{G}$ the set of all RDF graphs, $Rep : \mathcal{D} \to 2^{\mathcal{G}}$ 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 $[\![q]\!]_D \in \mathcal{D}$ such that $Rep([\![q]\!]_D) \equiv_{\mathcal{Q}} [\![q]\!]_{Rep(D)}$.

The next step towards the development of a representation system for RDF$^{\text{i}}$ and SPARQL is to define the notion of *coinitiality*.

Definition 8.6. 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 every $G \in \mathcal{G}$ there exists some $H \in \mathcal{H}$ such that $H \subseteq G$, and for every $H \in \mathcal{H}$ there exists some $G \in \mathcal{G}$ such that $G \subseteq H$.

Example 8.7. The following sets are coinitial:

$$\mathcal{G} = \Big\{ \underbrace{\big\{(a,b,c),(a,e,d),(a,f,g)\big\}}_{G_1}, \underbrace{\big\{(a,b,c),(a,e,d)\big\}}_{G_2}, \underbrace{\big\{(a,b,c)\big\}}_{G_3} \Big\},$$

$$\mathcal{H} = \Big\{ \underbrace{\big\{(a,b,c),(a,e,d)\big\}}_{H_1}, \underbrace{\big\{(a,b,c)\big\}}_{H_2} \Big\}.$$

To see why the sets $\mathcal{G}$ and $\mathcal{H}$ are coinitial, observe that for every $G \in \mathcal{G}$, there exists an element $H \in \mathcal{H}$ such that $H \subseteq G$. Take for example the pairs (G_1, H_1), (G_2, H_1), and (G_3, H_2). Similarly, such elements in $\mathcal{G}$ do exist for every element of $\mathcal{H}$, i.e., (H_1, G_2) and (H_2, G_3).

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. In the above example, the greatest lower bound is $\bigcap \mathcal{G} = \bigcap \mathcal{H} = \{(a, b, c)\}$.

An important property of monotone SPARQL queries is that they preserve coinitiality. This is stated formally in the following proposition.

Proposition 8.8. 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 every $q \in \mathcal{Q}$ it holds that $[\![q]\!]_{\mathcal{G}} \approx [\![q]\!]_{\mathcal{H}}$.

Proof. The proof is straightforward from the monotonicity property. Since $\mathcal{G} \approx \mathcal{H}$ we have the following:

- for every $G \in \mathcal{G}$ there exists some $H_G \in \mathcal{H}$ such that $H_G \subseteq G$ and
- for every $H \in \mathcal{H}$ there exists some $G_H \in \mathcal{G}$ such that $G_H \subseteq H$.

Let $q \in \mathcal{Q}$. Because $\mathcal{Q}$ is monotone, from the first item above we have that $[\![q]\!]_{H_G} \subseteq [\![q]\!]_G$ for every $G \in \mathcal{G}$ and some $H_G \in \mathcal{H}$. Notice also that this property holds for every set making up $[\![q]\!]_{\mathcal{G}}$ and some $[\![q]\!]_{H_G} \in [\![q]\!]_{\mathcal{H}}$. Working similarly for the second item, we prove that $[\![q]\!]_{G_H} \subseteq [\![q]\!]_H$ for every $H \in \mathcal{H}$ and some $G_H \in \mathcal{G}$. Hence, $[\![q]\!]_{\mathcal{G}}$ and $[\![q]\!]_{\mathcal{H}}$ are coinitial, that is, $[\![q]\!]_{\mathcal{G}} \approx [\![q]\!]_{\mathcal{H}}$. □

It turns out that monotonicity is sufficient for establishing our results about representation systems. Thus, in the following, we focus on the monotone fragments of SPARQL as identified in Section 7, namely, $\mathcal{Q}^{C'}_{AUF}$ and $\mathcal{Q}^{C'}_{WD}$.

Lemma 8.9. Let $\mathcal{G}$ and $\mathcal{H}$ be sets of RDF graphs. If $\mathcal{G}$ and $\mathcal{H}$ are coinitial then $\mathcal{G}$ and $\mathcal{H}$ are $\mathcal{Q}^{C'}_{AUF}$-equivalent (i.e., $\mathcal{G} \equiv_{\mathcal{Q}^{C'}_{AUF}} \mathcal{H}$) and $\mathcal{Q}^{C'}_{WD}$-equivalent (i.e., $\mathcal{G} \equiv_{\mathcal{Q}^{C'}_{WD}} \mathcal{H}$).

Proof. The proof is similar to the one given in [41, Lemma 4.2]. We give the proof for $\mathcal{Q}_{AUF}^{C'}$. The proof for $\mathcal{Q}_{WD}^{C'}$ is similar.

We have to prove that $\bigcap[\![q]\!]_{\mathcal{G}} = \bigcap[\![q]\!]_{\mathcal{H}}$ for every $q \in \mathcal{Q}_{AUF}^{C'}$. Let $\mathcal{G} \approx \mathcal{H}$. Then, from Proposition 8.8 and because of the monotonicity property of $\mathcal{Q}_{AUF}^{C'}$, we have that $[\![q]\!]_{\mathcal{G}} \approx [\![q]\!]_{\mathcal{H}}$ for every $q \in \mathcal{Q}_{AUF}^{C'}$. Thus, for every $[\![q]\!]_G \in [\![q]\!]_{\mathcal{G}}$ there exists a $[\![q]\!]_{H_G} \in [\![q]\!]_{\mathcal{H}}$ such that $[\![q]\!]_{H_G} \subseteq [\![q]\!]_G$. So, we have

$$\bigcap[\![q]\!]_{\mathcal{G}} = \bigcap_{G\in\mathcal{G}} [\![q]\!]_G \supseteq \bigcap_{G\in\mathcal{G}} [\![q]\!]_{H_G} \supseteq \bigcap_{H\in\mathcal{H}} [\![q]\!]_H = \bigcap[\![q]\!]_{\mathcal{H}}.$$

To see why $\bigcap_{G\in\mathcal{G}} [\![q]\!]_G \supseteq \bigcap_{G\in\mathcal{G}} [\![q]\!]_{H_G}$, notice that the expressions $\bigcap_{G\in\mathcal{G}} [\![q]\!]_G$ and $\bigcap_{G\in\mathcal{G}} [\![q]\!]_{H_G}$ can be written respectively as

$$[\![q]\!]_{G_1} \cap [\![q]\!]_{G_2} \cap \cdots \quad \text{and} \quad [\![q]\!]_{H_{G_1}} \cap [\![q]\!]_{H_{G_2}} \cap \cdots$$

and that the relation $[\![q]\!]_{H_{G_i}} \subseteq [\![q]\!]_{G_i}$ holds. Therefore, if an element x is in $\bigcap_{G\in\mathcal{G}} [\![q]\!]_{H_G}$, it will be in every $[\![q]\!]_{H_{G_i}}$, and thus it will be in every $[\![q]\!]_{G_i}$, which proves the relation.

Now, to see why $\bigcap_{G\in\mathcal{G}} [\![q]\!]_{H_G} \supseteq \bigcap_{H\in\mathcal{H}} [\![q]\!]_H$, notice that the relation can be written as

$$\bigcap[\![q]\!]_{\mathcal{H}_{\mathcal{G}}} \supseteq \bigcap[\![q]\!]_{\mathcal{H}}$$

where

$$\mathcal{H}_{\mathcal{G}} \equiv \{H \in \mathcal{H} \mid H \subseteq G \text{ for some } G \in \mathcal{G}\}.$$

Thus, $\mathcal{H}_{\mathcal{G}} \subseteq \mathcal{H}$, and therefore, $\bigcap \mathcal{H}_{\mathcal{G}} \supseteq \bigcap \mathcal{H}$. Similarly if q is a monotone query, then $[\![q]\!]_{\mathcal{H}_{\mathcal{G}}} \subseteq [\![q]\!]_{\mathcal{H}}$ and $\bigcap[\![q]\!]_{\mathcal{H}_{\mathcal{G}}} \supseteq \bigcap[\![q]\!]_{\mathcal{H}}$.

Therefore, we showed that $\bigcap[\![q]\!]_{\mathcal{G}} \supseteq \bigcap[\![q]\!]_{\mathcal{H}}$. We work similarly to prove $\bigcap[\![q]\!]_{\mathcal{H}} \supseteq \bigcap[\![q]\!]_{\mathcal{G}}$ and finally get $\bigcap[\![q]\!]_{\mathcal{G}} = \bigcap[\![q]\!]_{\mathcal{H}}$. □

We will now present our main theorem which characterizes the evaluation of monotone $\mathcal{Q}_{AUF}^{C'}$ and $\mathcal{Q}_{WD}^{C'}$ queries (Theorem 8.15). Before we do this, we need a few definitions and preliminary results. The first definition allows us to apply a valuation to a conditional mapping.

Definition 8.10. Let $v : U \to C$ be a valuation and $\mu = (\nu, \theta)$ a conditional mapping such that $\mathbf{M}_{\mathcal{L}} \models v(\theta)$. Then $v(\mu)$ denotes the mapping that is obtained from ν by replacing each e-literal $\_l$ appearing in ν by the constant $v(\_l)$.

By applying a valuation to a conditional mapping, we get an *ordinary mapping* like in the case of RDF. This occurs because the application of the valuation to the constraint of the mapping makes the resulting constraint equivalent to *true*, and hence, we can simply disregard it.

In a similar way, we can extend a valuation v to a set of mappings Ω as follows.

Definition 8.11. Let Ω be a set of conditional mappings and $v : U \to C$ a valuation. Then, $v(\Omega) = \big\{v(\mu) \mid \mu = (\nu, \theta) \in \Omega \text{ and } \mathbf{M}_{\mathcal{L}} \models v(\theta)\big\}$.

The next definition allows us to apply a valuation to an RDF$^{\text{i}}$ database.

Definition 8.12. Let $v : U \to C$ be a valuation and $D = (G, \phi)$ an RDF$^{\text{i}}$ database such that $\mathbf{M}_{\mathcal{L}} \models v(\phi)$. Then $v(D)$ denotes the RDF graph $v(G)$.

Proposition 8.13. Let $D = (G, \phi)$ be an RDF$^{\text{i}}$ database and q a query from a monotone fragment $\mathcal{Q}$ of SPARQL. If for all valuations v such that $\mathbf{M}_{\mathcal{L}} \models v(\phi)$ equality $v(\llbracket q \rrbracket_D) = \llbracket q \rrbracket_{v(D)}$ holds, then $\mathit{Rep}(\llbracket q \rrbracket_D) \approx \llbracket q \rrbracket_{\mathit{Rep}(D)}$.

Proof. Let $\llbracket q \rrbracket_D$ be the pair $D_1 = (G_1, \phi)$ and G' an RDF graph such that $G' \in \mathit{Rep}(D_1)$. By the definition of *Rep*, there exists a valuation v' such that $\mathbf{M}_{\mathcal{L}} \models v'(\phi)$ and $G' \supseteq v'(G_1)$. Since $v(\llbracket q \rrbracket_D) = \llbracket q \rrbracket_{v(D)}$ and $\mathbf{M}_{\mathcal{L}} \models v'(\phi)$ we get

$$G' \supseteq v'(G_1) = v'(D_1) = v'(\llbracket q \rrbracket_D) = \llbracket q \rrbracket_{v'(D)} = H$$

where H is a new symbol introduced for convenience. Now, observe that $v'(D)$ is the RDF graph $v'(G)$ which is an element of $\mathit{Rep}(D)$ since $\mathbf{M}_{\mathcal{L}} \models v'(\phi)$. Since also

$$\llbracket q \rrbracket_{\mathit{Rep}(D)} = \big\{\llbracket q \rrbracket_G \mid G \in \mathit{Rep}(D)\big\}$$

it turns out that $H \in \llbracket q \rrbracket_{\mathit{Rep}(D)}$. To see this, notice that $H = \llbracket q \rrbracket_{v'(D)}$ and that $v'(D) \in \mathit{Rep}(D)$.

This proves that, for each $G' \in Rep(\llbracket q \rrbracket_D)$, there exists an $H \in \llbracket q \rrbracket_{Rep(D)}$ such that $H \subseteq G'$. To prove that $Rep(\llbracket q \rrbracket_D) \approx \llbracket q \rrbracket_{Rep(D)}$ we need to show the same for the other direction.

Let H' be an RDF graph such that $H' \in \llbracket q \rrbracket_{Rep(D)}$. Then $H' = \llbracket q \rrbracket_H$ for some $H \in Rep(D)$. By the definition of Rep, there exists a valuation v' such that $\mathbf{M}_{\mathcal{L}} \models v'(\phi)$ and $H \supseteq v'(G)$ or equivalently $H \supseteq v'(D)$. Since $v(\llbracket q \rrbracket_D) = \llbracket q \rrbracket_{v(D)}$ and $\mathbf{M}_{\mathcal{L}} \models v'(\phi)$, we have $\llbracket q \rrbracket_{v'(D)} = v'(\llbracket q \rrbracket_D)$. Since q belongs to a monotone fragment of SPARQL and $H \supseteq v'(D)$, we have $\llbracket q \rrbracket_H \supseteq \llbracket q \rrbracket_{v'(D)}$ which is equivalent to $H' \supseteq v'(\llbracket q \rrbracket_D)$. Now observe that since $\mathbf{M}_{\mathcal{L}} \models v'(\phi)$, $v'(\llbracket q \rrbracket_D)$ is an RDF graph G' and that $G' \in Rep(\llbracket q \rrbracket_D)$. Therefore, we showed that for every $H' \in \llbracket q \rrbracket_{Rep(D)}$ there exists a $G' \in Rep(\llbracket q \rrbracket_D)$ such that $G' \subseteq H'$. Hence, $Rep(\llbracket q \rrbracket_D) \approx \llbracket q \rrbracket_{Rep(D)}$. □

Proposition 8.14. Let $D = (G, \phi)$ be an RDF$^{\mathrm{i}}$ database, $q = (E, P)$ a CONSTRUCT query without blank nodes in E, and v a valuation such that $\mathbf{M}_{\mathcal{L}} \models v(\phi)$. Then, $v(\llbracket P \rrbracket_D) = \llbracket P \rrbracket_{v(D)}$ implies $v(\llbracket q \rrbracket_D) = \llbracket q \rrbracket_{v(D)}$.

The proof of Proposition 8.14 may be found in Appendix B. We are now ready to prove our main result.

Theorem 8.15 (Representation Theorem)**.** The triples $\langle \mathcal{D}, Rep, \mathcal{Q}^{C'}_{AUF} \rangle$ and $\langle \mathcal{D}, Rep, \mathcal{Q}^{C'}_{WD} \rangle$ are representation systems.

Proof. We give a sketch of the proof. The full proof may be found in Appendix A.

To prove that triple $\langle \mathcal{D}, Rep, \mathcal{Q}^{C'}_{AUF} \rangle$ is a representation system, it is sufficient to show that for any $D = (G, \phi) \in \mathcal{D}$ and any query $q = (E, P) \in \mathcal{Q}^{C'}_{AUF}$ the evaluation of q over D, that is $\llbracket q \rrbracket_D$, is such that $Rep(\llbracket q \rrbracket_D) \equiv_{\mathcal{Q}^{C'}_{AUF}} \llbracket q \rrbracket_{Rep(D)}$. By Lemma 8.9 it is sufficient to prove that

$$Rep(\llbracket q \rrbracket_D) \approx \llbracket q \rrbracket_{Rep(D)}. \tag{4}$$

By Propositions 8.13 and 8.14, relation (4) holds if $v(\llbracket P \rrbracket_D) = \llbracket P \rrbracket_{v(D)}$ holds for all valuations v such that $\mathbf{M}_{\mathcal{L}} \models v(\phi)$. We prove this in Appendix A by induction on the structure of graph patterns P of $\mathcal{Q}^{C'}_{AUF}$.

To show that the triple $\langle \mathcal{D}, Rep, \mathcal{Q}^{C'}_{WD} \rangle$ is a representation system, the proof is similar to the above and differs only in the inductive step for the OPT operator. □

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 (also see the discussion on SELECT in Section 9).

Remark. The concepts of $\mathcal{Q}$-equivalence (Definition 8.4), representation systems (Definition 8.5), and coinitiality (Definition 8.6) were originally defined by Imieliński and Lipski [41] for incomplete relational databases. In the same context, Grahne [30] refers to representation systems as *weak query systems* and to coinitiality as $+$-*equivalence*. If Definition 8.5 is modified so that formula (3) is used in place of its more relaxed version $Rep(\llbracket q \rrbracket_D) \equiv_{\mathcal{Q}} \llbracket q \rrbracket_{Rep(D)}$, then the concept of representation systems we get corresponds to the *strong query systems* of Grahne [30].

9. Certain Answers Computation

This section studies how the certain answers to a SPARQL query q over an RDF$^{\text{i}}$ database D can be computed, i.e., how to compute $\bigcap \llbracket q \rrbracket_{Rep(D)}$. It also defines the associated decision problem, namely the certainty problem, and it studies its computational complexity.

9.1. An Algorithm for Computing Certain Answers

Having Theorem 8.15, it is easy to compute the certain answers to a query in the fragments $\mathcal{Q}^{C'}_{AUF}$ and $\mathcal{Q}^{C'}_{WD}$ of SPARQL. Since $\langle \mathcal{D}, Rep, \mathcal{Q}^{C'}_{AUF} \rangle$ and $\langle \mathcal{D}, Rep, \mathcal{Q}^{C'}_{WD} \rangle$ are representation systems, by Definition 8.5 $Rep(\llbracket q \rrbracket_D) \equiv_{\mathcal{Q}^{C'}_{AUF}} \llbracket q \rrbracket_{Rep(D)}$ and $Rep(\llbracket q \rrbracket_D) \equiv_{\mathcal{Q}^{C'}_{WD}} \llbracket q \rrbracket_{Rep(D)}$ for all q in the above fragments and RDF$^{\text{i}}$ databases $D \in \mathcal{D}$. Then, by Definition 8.4 and the identity query on the sets $Rep(\llbracket q \rrbracket_D)$ and $\llbracket q \rrbracket_{Rep(D)}$, we have $\bigcap Rep(\llbracket q \rrbracket_D) = \bigcap \llbracket q \rrbracket_{Rep(D)}$. Thus, we

can equivalently compute the certain answers to q over D by computing the set $\bigcap Rep(\llbracket q \rrbracket_D)$, where $\llbracket q \rrbracket_D$ is computed using the algebra of Section 6.

Before presenting the algorithm for certain answers computation, we need to introduce some auxiliary constructs similar to the ones defined by Imieliński and Lipski [41] and Grahne [30] in the case of incomplete relational databases.

Definition 9.1. 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. This is demonstrated in the following example.

Example 9.2. Let us consider the RDF$^{\text{i}}$ database $D_q = (G_q, \phi)$ from Example 6.29. The EQ-completed form of D_q is the RDF$^{\text{i}}$ database $D_q^{\text{EQ}} = (G_q^{\text{EQ}}, \phi)$, and is as follows:

$$\big(\big\{((\text{fire1}, \text{hasGeometry}, \texttt{"g7"}), \theta_1 \wedge \theta_2),\ ((\text{ba1}, \text{hasGeometry}, \texttt{"g7"}), \theta_1 \wedge \theta_2),$$
$$((\text{fire1}, \text{hasGeometry}, \texttt{"g7"}), \theta_1 \wedge \theta_3),\ ((\text{ba2}, \text{hasGeometry}, \texttt{"g8"}), \theta_1 \wedge \theta_3)\big\},$$
$$\_\text{R1 NTPP } \texttt{"g1"} \ \wedge\ \_\text{R2 EQ } \texttt{"g7"} \ \wedge\ \_\text{R3 EQ } \texttt{"g8"} \ \wedge\ \_\text{R1 EQ } \_\text{R2}\big).$$

In the conditional graph G_q^{EQ} above, the e-literals $_$R1, $_$R2, and $_$R3 from the conditional graph G_q have been replaced by the constants "g7", "g7", and "g8" respectively, since the PCL-constraints $_$R1 EQ "g7", $_$R2 EQ "g7", and $_$R3 EQ "g8" are all entailed by the global constraint ϕ.

Definition 9.3. 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

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

Given the above definition, 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. This is demonstrated in the following example.

Example 9.4. Let us consider the RDF$^{\mathrm{i}}$ database $D_q^{\mathrm{EQ}} = (G_q^{\mathrm{EQ}}, \phi)$ from Example 9.2. The normalized form of D_q^{EQ} is the RDF$^{\mathrm{i}}$ database $D_q^{\mathrm{EQ}^*} = (G_q^{\mathrm{EQ}^*}, \phi)$, and is as follows:

$\big(\big\{((\mathrm{fire1}, \mathrm{hasGeometry}, \texttt{"g7"}),\ (\theta_1 \wedge \theta_2) \vee (\theta_1 \wedge \theta_3)),$
$((\mathrm{ba1}, \mathrm{hasGeometry}, \texttt{"g7"}),\ \theta_1 \wedge \theta_2),\ ((\mathrm{ba2}, \mathrm{hasGeometry}, \texttt{"g8"}),\ \theta_1 \wedge \theta_3)\big\},$
$\_\mathrm{R1}\ \mathrm{NTPP}\ \texttt{"g1"}\ \ \wedge\ \ \_\mathrm{R2}\ \mathrm{EQ}\ \texttt{"g7"}\ \ \wedge\ \ \_\mathrm{R3}\ \mathrm{EQ}\ \texttt{"g8"}\ \wedge\ \_\mathrm{R1}\ \mathrm{EQ}\ \_\mathrm{R2}\big).$

In the conditional graph G_q^{EQ} the e-triple $(\mathrm{fire1}, \mathrm{hasGeometry}, \texttt{"g7"})$ appears in the triple part of two conditional triples. In the conditional graph $G_q^{\mathrm{EQ}^*}$ above, these conditional triples have been combined into a new one, the triple part of which is the aforementioned e-triple and the conditional part the disjunction of the constraints of the combined conditional triples.

Lemma 9.5. Let $D = (G, \phi)$ be an RDF$^{\mathrm{i}}$ database. Then

$$\bigcap \mathit{Rep}(D) = \bigcap \mathit{Rep}((D^{\mathrm{EQ}})^*).$$

Proof. We will first prove that $\bigcap \mathit{Rep}(D) \subseteq \bigcap \mathit{Rep}((D^{\mathrm{EQ}})^*)$. Let t be an RDF triple such that $t \notin \bigcap \mathit{Rep}((D^{\mathrm{EQ}})^*)$. Then, by the definition of Rep we get

$$t \notin \bigcap \big\{H \mid \text{there exists valuation } v \text{ such that } \mathbf{M}_{\mathcal{L}} \models v(\phi) \text{ and } H \supseteq v((G^{\mathrm{EQ}})^*)\big\}.$$

Therefore, there exists valuation v such that $\mathbf{M}_{\mathcal{L}} \models v(\phi)$ and $t \notin v((G^{\mathrm{EQ}})^*)$, and thus

a) either there is no conditional triple $(t', \theta') \in (G^{\mathrm{EQ}})^*$ such that $\mathbf{M}_{\mathcal{L}} \models v(\theta')$, that is, $\mathbf{M}_{\mathcal{L}} \not\models v(\theta')$,

b) or all conditional triples $(t', \theta') \in (G^{\mathrm{EQ}})^*$ such that $\mathbf{M}_{\mathcal{L}} \models v(\theta')$ are such that $v(t') \neq t$.

Observe now that for conditional triples in $(G^{\mathrm{EQ}})^*$, θ' can be written as $\bigvee_i \theta'_i$. So, if $(t', \theta') \in (G^{\mathrm{EQ}})^*$, then $(t', \theta'_i) \in G^{\mathrm{EQ}}$. Therefore, there is a conditional

triple $(t'', \theta'_i) \in G$, such that t' and t'' possibly differ in their object position. In the following we construct G and we show that $t \notin v(G)$ for this particular v.

For item a), since $\mathbf{M}_{\mathcal{L}} \not\models v(\theta')$ we have that $\mathbf{M}_{\mathcal{L}} \not\models v(\theta'_i)$ for every θ'_i, and thus such triples are dropped during application of valuation v to G. Hence, if it was the case that $t \in \bigcap Rep(D)$, it would be so only from item b). Consider now item b) and a triple $(t', \theta') \in (G^{\text{EQ}})^*$. Since $\mathbf{M}_{\mathcal{L}} \models v(\theta')$ and $(t', \theta'_i) \in G^{\text{EQ}}$, then some (or even all) θ'_i would be such that $\mathbf{M}_{\mathcal{L}} \models v(\theta'_i)$.

Let us now construct the conditional graph G from G^{EQ}. Since $(t', \theta'_i) \in G^{\text{EQ}}$, there exists a conditional triple $(t'', \theta'_i) \in G$ such that t' and t'' possibly differ in their object position. Let t' be the e-triple (s, p, o). Then:

1. If $o \in C$, then either t'' is the same with t', or it has in its object position an e-literal $\_l$ such that $\phi \models \_l \text{ EQ } o$.
2. If $o \notin C$, then t' and t'' are the same.

Let us now apply valuation v to G. Notice that $v(G)$ contains only RDF triples coming from conditional triples with a condition θ such that $v(\theta)$ is *true*. Thus, we could focus only on the conditional triples of G with such conditions (it is clear from above that such conditional triples do exist). To construct the RDF graph $v(G)$ it suffices to consider the two items above when applying v to a conditional triple (t'', θ'_i) of G.

According to item 2. and since $v(t') \neq t$ (we are considering item b) above), we have that $v(t'') \neq t$ as well. As for item 1., if $t' = t''$, then clearly we have $v(t'') \neq t$, since $v(t') \neq t$. Otherwise, t'' would be the triple $(s, p, \_l)$ such that $\phi \models \_l \text{ EQ } o$. In such a case, the application of v to t' would leave t' unchanged, thus the RDF triple t would contain in the object position a literal from C and one that would be different from o. Since also $\phi \models \_l \text{ EQ } o$, then every valuation v' that makes $v'(\phi)$ *true* it should make $v'(\_l \text{ EQ } o)$ *true* as well. Thus, such valuations would map the e-literal $\_l$ to the constant o. Since the valuation v we consider is such a valuation, it maps $\_l$ to the constant o. Thus, again $v(t'') \neq t$.

Therefore, we showed that $t \notin v(G)$. Hence, from the definition of *Rep* we have $t \notin \bigcap Rep(D)$ which proves that $\bigcap Rep(D) \subseteq \bigcap Rep((D^{\text{EQ}})^*)$.

The other direction of the proof for showing $\bigcap Rep((D^{\mathrm{EQ}})^*) \subseteq \bigcap Rep(D)$ is similar. □

Having Lemma 9.5, it is easy to give an algorithm that computes the certain answers to a query.

Theorem 9.6. Let $D = (G, \phi)$ be an RDF$^{\mathrm{i}}$ database and q a query from $\mathcal{Q}^{C'}_{AUF}$ or $\mathcal{Q}^{C'}_{WD}$. The certain answers of q over D can be computed as follows: i) compute $[\![q]\!]_D$ according to Section 6 and let $D_q = (G_q, \phi)$ be the resulting RDF$^{\mathrm{i}}$ database, ii) compute the RDF$^{\mathrm{i}}$ database $(H_q, \phi) = ((D_q)^{\mathrm{EQ}})^*$, and iii) return the following set of RDF triples:

$$\big\{(s, p, o) \mid ((s, p, o), \theta) \in H_q \text{ such that } \phi \models \theta \text{ and } o \notin U\big\}.$$

Proof. Notice that the certain answers for q over D is the set $\bigcap [\![q]\!]_{Rep(D)}$. By the Representation Theorem (Theorem 8.15) and since $q \in \mathcal{Q}^{C'}_{AUF}$ or $q \in \mathcal{Q}^{C'}_{WD}$, it suffices to show that the algorithm computes the set $\bigcap Rep([\![q]\!]_D)$. Notice that equation $\bigcap [\![q]\!]_{Rep(D)} = \bigcap Rep([\![q]\!]_D)$ is a logical consequence of Definition 8.5 for the identity query (see how the same equation was proved in the beginning of this section).

By Lemma 9.5, it now suffices to prove that the given algorithm computes the set $\bigcap Rep(([\![q]\!]_D)^{\mathrm{EQ}})^*)$, or using the notation of Theorem 9.6, the set $\bigcap Rep((D_q)^{\mathrm{EQ}})^*)$.

Step i) of the algorithm evaluates q over D, that is, it computes $D_q = (G_q, \phi)$, while step ii) computes the EQ-completed form of D_q, that is, $(D_q)^{\mathrm{EQ}}$, and then its normalized form, $((D_q)^{\mathrm{EQ}})^*$. It remains to show that step iii) computes exactly the intersection over the RDF graphs in $Rep(((D_q)^{\mathrm{EQ}})^*)$. Consider the set $\bigcap Rep((D_q)^{\mathrm{EQ}})^*)$ or equivalently the set

$$\bigcap \big\{H \mid \text{there exists valuation } v \text{ such that } \mathbf{M}_{\mathcal{L}} \models v(\phi) \text{ and } H \supseteq v(((D_q)^{\mathrm{EQ}})^*)\big\}.$$

An RDF triple t belongs to the above set iff for all valuations v such that $\mathbf{M}_{\mathcal{L}} \models v(\phi)$, it holds $t \in v(((D_q)^{\mathrm{EQ}})^*)$. This is equivalent to requiring that a conditional triple (t', θ') exists in H_q such that $\mathbf{M}_{\mathcal{L}} \models v(\theta')$ and $t = v(t')$ for

all valuations v such that $\mathbf{M}_{\mathcal{L}} \models v(\phi)$. Since step iii) requires that $\phi \models \theta'$ is true, then $\mathbf{M}_{\mathcal{L}} \models v(\theta')$ holds for all valuations v such that $\mathbf{M}_{\mathcal{L}} \models v(\phi)$. Also, equation $t = v(t')$ holds for any valuation v such that $\mathbf{M}_{\mathcal{L}} \models v(\phi)$ iff t' respects the following two cases:

- it does not contain any e-literal in the object position,
- it does contain an e-literal $\_l$ and all valuations v above map $\_l$ to the same constant $c \in C$, which t has it in its object position.

Since step iii) selects all conditional triples (t', θ) of H_q such that $\phi \models \theta$ and $o \notin U$, the first case above is satisfied. The second case above is out of question: H_q does not contain such a triple since all such e-literals have already been substituted by the respective constant $c \in C$ such that $\phi \models \_l \text{ EQ } c$.

Thus, step iii) computes exactly the set $\bigcap Rep(((D_q)^{\text{EQ}})^*)$. □

The following example illustrates the algorithm of Theorem 9.6 for computing certain answers.

Example 9.7. Let us consider the RDF$^{\text{i}}$ database $D = (G, \phi)$ and the CONSTRUCT query q from Example 6.29. The certain answers to q over D can be computed according to the steps of Theorem 9.6 as follows:

i) The RDF$^{\text{i}}$ database $D_q = (G_q, \phi)$ corresponding to the evaluation of q over D is the one from Example 6.29.

ii) The EQ-completed form of D_q, D_q^{EQ}, is given in Example 9.2. The normalized form of D_q^{EQ}, $D_q^{\text{EQ}^*}$, is given in Example 9.4.

iii) The certain answers are the following set of RDF triples:

$$\{\ (\text{fire1}, \text{hasGeometry}, \texttt{"g7"}),\ \ (\text{ba1}, \text{hasGeometry}, \texttt{"g7"})\ \}.$$

These RDF triples come from the triple part of the first two conditional triples of the RDF$^{\text{i}}$ database $D_q^{\text{EQ}^*}$. The triple parts of these conditional triples do not have any e-literal in the object position and also their constraints are entailed by the global constraint ϕ. To see this, observe that

the first condition requires that either the pair _R1 and _R2 or the pair _R1 and _R3 are completely inside rectangle Q_r of Figure 2(a) (see also Example 6.29 on page 33 where these conditions θ_i are defined). Observing the global constraint ϕ and since _R1 and _R2 are equal to "g7", which is clearly inside rectangle Q_r, this condition is entailed by ϕ. Consequently, we get the RDF triple (fire1, hasGeometry, "g7") in the certain answers. The same applies to the condition of the second conditional triple of database $D_q^{\mathrm{EQ}^*}$ from which we get the RDF triple (ba1, hasGeometry, "g7"). As for the condition of the third conditional triple of database $D_q^{\mathrm{EQ}^*}$, it can be easily seen that it is not entailed by ϕ since _R3 is required to be inside rectangle Q_r, a fact that does not hold because the global constraint requires that _R3 is equal to "g8" which is not completely inside Q_r.

9.2. The Certainty Problem

In order to analyze the data complexity of computing the certain answers to a CONSTRUCT query over an RDF$^{\mathrm{i}}$ database when $\mathcal{L}$ is a constraint language, we need to consider the associated decision problem. Following Grahne [30], we define the *certainty problem*.

Definition 9.8. Let q be a CONSTRUCT query. The *certainty problem* for query q, RDF graph H, and RDF$^{\mathrm{i}}$ 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 9.9. Let $D = (G, \phi)$ be an RDF$^{\mathrm{i}}$ database, $\_\mathbf{l}$ a vector of all e-literals in D, 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

$$\bigwedge_{t \in H} \big(\forall \_\mathbf{l}\big)\big(\phi(\_\mathbf{l}) \supset \Theta(t, q, D, \_\mathbf{l})\big) \tag{5}$$

is *true* in $\mathbf{M}_{\mathcal{L}}$ where $\Theta(t, q, D, \_\mathbf{l})$ is a disjunction $\theta_1 \vee \cdots \vee \theta_k$ that is constructed as follows. Let $[\![q]\!]_D = (G', \phi)$. Expression $\Theta(t, q, D, \_\mathbf{l})$ 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. Condition θ_i is then:

- 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 all t'_i, then $\Theta(t, q, D, \_\mathbf{l})$ is taken to be $false$.

Proof. Working similar to the proof of Theorem 9.6, it suffices to show that an RDF triple t is in the certain answers of $q \in \mathcal{Q}^{C'}_{AUF} \cup \mathcal{Q}^{C'}_{WD}$ over D, that is, $t \in \bigcap Rep([\![q]\!]_D)$, if and only if the following formula is valid:

$$(\forall \_\mathbf{l})(\phi(\_\mathbf{l}) \supset \theta_1 \vee \ldots \vee \theta_k). \tag{6}$$

Let $[\![q]\!]_D = (G', \phi)$ and consider an RDF triple $t \notin \bigcap Rep([\![q]\!]_D)$. Then there exists valuation v such that $\mathbf{M}_{\mathcal{L}} \models v(\phi)$ and $t \notin v(G')$. Therefore, G' contains conditional triples (t', θ') such that either

- $\mathbf{M}_{\mathcal{L}} \not\models v(\theta')$ or
- $\mathbf{M}_{\mathcal{L}} \models v(\theta')$ and $t \neq v(t')$.

In the former case and since $\mathbf{M}_{\mathcal{L}} \not\models v(\theta')$, formula (6) would be unsatisfiable. To see this, notice that $\mathbf{M}_{\mathcal{L}} \not\models v(\theta')$ implies $\mathbf{M}_{\mathcal{L}} \not\models v(\theta'_i)$ and $\mathbf{M}_{\mathcal{L}} \not\models v(\theta'_i \wedge (\_l \text{ EQ } o))$ and thus the disjunction $\theta_1 \vee \ldots \vee \theta_k$ in (6) is always $false$, and hence the whole formula is unsatisfiable.

In the latter case, notice that $t \neq v(t')$ implies one of the following cases:

- t and t' do not agree in the subject or predicate position, or
- if they do, either they do not agree in the object position, or their objects are not of the proper kind (i.e., the object of t is a constant from C and the object of t' is an e-literal from U), or if they are, then valuation v does not map that e-literal $\_l$ to that constant o, i.e., $v(\_l) \neq o$.

From the first case, no constraint θ_i is included in formula (6). As for the second case, either no constraint θ_i is generated (the case in which they also differ in the object position) or θ_i is $\theta_i' \wedge (\_l \text{ EQ } o)$. As we pointed above, since valuation v does not map the e-literal $\_l$ to the constant o, then θ_i is $false$. Hence, formula (6) is unsatisfiable as well.

We sketch the other direction of the proof. We just need to consider an RDF triple t such that $t \in \bigcap Rep(\llbracket q \rrbracket_D)$ and prove that $\big(\forall \_\mathbf{l}\big)\big(\phi(\_\mathbf{l}) \supset \Theta(t, q, D, \_\mathbf{l})\big)$ is $true$ in $\mathbf{M}_{\mathcal{L}}$. From Theorem 9.6, we have that

$$t \in \big\{(s,p,o) \mid ((s,p,o),\theta) \in H_q \text{ such that } \phi \models \theta \text{ and } o \notin U\big\}$$

where H_q is the conditional graph of the database $((\llbracket q \rrbracket_D)^{\text{EQ}})^*$. It is easy to see now that condition θ in the above set complies with the structure of expression $\Theta(t, q, D, \_\mathbf{l})$. Since also in the above set we have that $\phi \models \theta$, then $\big(\forall \_\mathbf{l}\big)\big(\phi(\_\mathbf{l}) \supset \Theta(t, q, D, \_\mathbf{l})\big)$ is $true$. □

We can also prove a theorem like the above for SELECT queries by defining the relevant decision problem and developing appropriate versions of the relevant results of Section 8 that are needed. This involves first modifying Definition 8.4 so that $\mathcal{H}$ and $\mathcal{G}$ are sets of sets of mappings and q is a SELECT query form (we call this SELECT-equivalence). Then, the condition of Definition 8.5, modified so that $\mathcal{Q}$-equivalence is substituted by SELECT-equivalence, can be proved using essentially the same techniques as the ones used to prove Theorem 8.15.

The following example illustrates how one can use Theorem 9.9 to decide the certainty problem.

Example 9.10. Let us consider the RDF$^{\text{i}}$ database $D = (G, \phi)$, the CONSTRUCT query q from Example 6.29, and the certain answers H to q over D as computed in Example 9.7. In the following, we verify whether the elements of H belong to the certain answers of q over D. Having Theorem 9.9, we need only check the validity of formula (5), which in our example becomes

$$\big(\forall \_\mathbf{l}\big)\big(\phi(\_\mathbf{l}) \supset \Theta(t_1, q, D, \_\mathbf{l})\big) \wedge \big(\forall \_\mathbf{l}\big)\big(\phi(\_\mathbf{l}) \supset \Theta(t_2, q, D, \_\mathbf{l})\big) \tag{7}$$

where t_1 is the RDF triple $(\text{fire1}, \text{hasGeometry}, \texttt{"g7"})$ and t_2 the RDF triple $(\text{ba1}, \text{hasGeometry}, \texttt{"g7"})$.

Let us now expand the expressions $\Theta(t_1, q, D, \_\mathbf{l})$ and $\Theta(t_2, q, D, \_\mathbf{l})$. First, we need to compute $[\![q]\!]_D$ according to Section 6. This has been done already in Example 6.29 where the evaluation has resulted in the RDF$^{\text{i}}$ database $D_q = (G_q, \phi)$. Second, we need to expand the sub-formulae Θ considering only the conditional triples of G_q the triple part of which matches with t_1 or t_2 on the subject and predicate positions. Triple t_1 matches with the first and third conditional triples of G_q, while triple t_2 matches with the second one. Therefore, the sub-formulae Θ will be expanded based on the conditions of the matched triples. Up to this point, the sub-formulae Θ have the following structure:

$$\Theta(t_1, q, D, \_\mathbf{l}) \text{ is } (\theta_1 \wedge \theta_2) \vee (\theta_1 \wedge \theta_3),$$
$$\Theta(t_2, q, D, \_\mathbf{l}) \text{ is } \theta_1 \wedge \theta_2.$$

Observe now that the object positions of t_1, t_2 contain a constant, whereas the object positions of the matched conditional triples contain e-literals. According to Theorem 9.9, the above sub-formulae should be augmented with an equality constraint between these e-literals and the constants. The sub-formulae Θ are now the following:

$$\Theta(t_1, q, D, \_\mathbf{l}) \text{ is } \big(\theta_1 \wedge \theta_2 \wedge \_\text{R1 EQ } \texttt{"g7"}\big) \vee \big(\theta_1 \wedge \theta_3 \wedge \_\text{R1 EQ } \texttt{"g7"}\big),$$
$$\Theta(t_2, q, D, \_\mathbf{l}) \text{ is } \theta_1 \wedge \theta_2 \wedge \_\text{R2 EQ } \texttt{"g7"}.$$

The two conjuncts of formula (7) are now the following:

$$\big(\forall \_\mathbf{l}\big)\big(\phi(\_\mathbf{l}) \supset \big(\theta_1 \wedge \theta_2 \wedge \_\text{R1 EQ } \texttt{"g7"}\big) \vee \big(\theta_1 \wedge \theta_3 \wedge \_\text{R1 EQ } \texttt{"g7"}\big)\big),$$
$$\big(\forall \_\mathbf{l}\big)\big(\phi(\_\mathbf{l}) \supset \theta_1 \wedge \theta_2 \wedge \_\text{R2 EQ } \texttt{"g7"}\big).$$

Since the universal quantifier ranges over the e-literals of the database, the

above becomes as follows:

$$\big(\forall \_\mathrm{R1}\big)\big(\forall \_\mathrm{R2}\big)\big(\forall \_\mathrm{R3}\big)\big(\phi(\_\mathrm{R1}, \_\mathrm{R2}, \_\mathrm{R3}) \supset \big(\theta_1 \wedge \theta_2 \wedge \_\mathrm{R1}\ \mathrm{EQ}\ \texttt{"g7"}\big) \vee \big(\theta_1 \wedge \theta_3 \wedge \_\mathrm{R1}\ \mathrm{EQ}\ \texttt{"g7"}\big)\big), \quad (8)$$

$$\big(\forall \_\mathrm{R1}\big)\big(\forall \_\mathrm{R2}\big)\big(\forall \_\mathrm{R3}\big)\big(\phi(\_\mathrm{R1}, \_\mathrm{R2}, \_\mathrm{R3}) \supset \theta_1 \wedge \theta_2 \wedge \_\mathrm{R2}\ \mathrm{EQ}\ \texttt{"g7"}\big). \quad (9)$$

Observe now that since ϕ entails _R1 EQ _R2 and θ_1, it also entails θ_2 and, therefore, it entails $\theta_1 \wedge \theta_2$. Since also ϕ entails _R1 EQ "g7", formulae (8) and (9) are valid.

9.3. The Data Complexity of the Certainty Problem

The data complexity of the certainty problem, $CERT_C(q, H, D)$, for q in the $\mathcal{Q}^{C'}_{AUF}$ or $\mathcal{Q}^{C'}_{WD}$ fragments of SPARQL and D in the set of RDF$^{\mathrm{i}}$ databases with constraints from ECL, dePCL, diPCL, and RCL is coNP-complete. This follows easily from known results by Grahne [30] for ECL and by Koubarakis [46, 48] and Van der Meyden [85] for dePCL, diPCL, and RCL. Thus, we have the expected increase in data complexity given that the data complexity of evaluating any SPARQL graph pattern over RDF graphs can be done in LOGSPACE [75].

The following proposition states this result for the language ECL formally.

Proposition 9.11. Let $D = (G, \phi)$ be an RDF$^{\mathrm{i}}$ database, q a query from the fragments of SPARQL $\mathcal{Q}^{C'}_{AUF}$ or $\mathcal{Q}^{C'}_{WD}$, and H an RDF graph. The certainty problem, $CERT_C(q, H, D)$, is coNP-complete for data complexity when the language of $\mathcal{L}$-constraints is ECL.

Proof. To decide $CERT_C(q, H, D)$, we have to check that $H \subseteq \bigcap [\![q]\!]_{Rep(D)}$ which, by Definition 5.5, is equivalent to checking that $H \subseteq [\![q]\!]_{v(D)}$ for all valuations v such that $\mathbf{M}_{ECL} \models v(\phi)$. Notice that the complement of this problem is to check whether there exists a valuation v such that $\mathbf{M}_{ECL} \models v(\phi)$ and $H \not\subseteq [\![q]\!]_{v(D)}$. In other words, it suffices to find a valuation v and a triple $t \in H$ such that $\mathbf{M}_{ECL} \models v(\phi)$ and $t \notin [\![q]\!]_{v(D)}$. This last problem is in NP, thus the certainty problem is in coNP.

Let us see why the complement problem defined above is in NP. We need only guess a valuation v with length equal to the number of e-literals in D, check that $\mathbf{M}_{ECL} \models v(\phi)$, a computation that is in the AC complexity class, and then check that there exists $t \in H$ such that $t \notin [\![q]\!]_{v(D)}$. The steps for accomplishing the latter check, using Definition 4.6 for evaluating CONSTRUCT query forms of standard SPARQL [74], are the following: 1) Choose the next triple $t \in H$. 2) Loop over all candidate mappings μ for set $[\![P]\!]_{v(D)}$ generating a mapping per iteration, where P is the graph pattern of query q. 3) Check that $\mu \in [\![P]\!]_{v(D)}$. 4) Construct the renaming function f_μ based on the mapping μ. 5) Generate set $S_\mu = \{\mu(f_\mu(E)) \cap (I \cup B) \times I \times T^C\}$. 6) Check whether $t \in S_\mu$. If yes, move to step 1, otherwise move to step 2. If there is no other mapping μ to check, return "yes". If there is no other triple to choose, return "no".

Step 2 above requires logarithmic space since the space required to store a candidate mapping μ from the set $[\![P]\!]_{v(D)}$ is $O(|P|(log|P| + log|D|))$ bits. This is because the mapping will contain $|P|$ variables and for each variable, it has to contain its value from D. The required space for each variable and value is $log|P|$ and $log|D|$, respectively. Since q is fixed, the graph pattern P is also fixed, therefore the space becomes logarithmic in the size of the database D.

Step 3 above can also be computed in LOGSPACE using the evaluation procedure EVAL presented by Pérez et al. [75]. Further, since q is fixed, then also the template E and graph pattern P are fixed. Thus, set S_μ of step 5 is of fixed size.

The coNP-hardness of $CERT_C(q, H, D)$ comes from a reduction from 3DNF tautology, which is known to be coNP-complete, and it is similar to the one employed by Grahne [30, Theorem 5.11, p. 118]. □

A similar proposition to the above for the languages dePCL, diPCL, and RCL follows.

Proposition 9.12. Let $D = (G, \phi)$ be an RDF$^{\text{i}}$ database, q a query from the fragments of SPARQL $\mathcal{Q}^{C'}_{AUF}$ or $\mathcal{Q}^{C'}_{WD}$, and H an RDF graph. The certainty problem, $CERT_C(q, H, D)$, is coNP-complete for data complexity when the lan-

guage of $\mathcal{L}$-constraints is one of dePCL, diPCL, or RCL.

Proof. We sketch the proof for dePCL. The proof is similar for diPCL and RCL. Similar to the proof for ECL, we show that the complement of $CERT_C(q, H, D)$ is NP-complete. To show membership in NP we use a non-deterministic Turing machine to guess in polynomial time a valuation v that satisfies ϕ and then iterate over every triple t of RDF graph H checking whether $t \notin [\![q]\!]_{v(D)}$. This last check is done using the procedure described in the proof for ECL.

Let us see now how we can guess a valuation v satisfying the global constraint ϕ of D in polynomial time. To do this, we have to guess a rational number for every e-literal of the database D, substitute these values for e-literals in the global constraint ϕ and check that ϕ is *true* in polynomial time. Using Lemma 7.3 and Theorem 8.5 of Koubarakis [48], and Theorem 9.9 of our work, we can restrict the values over which the e-literals range only to a finite number of integers. The exact ranges are given in [48] and depend on the maximum absolute value of the constants appearing in formula ϕ. Each value in these ranges takes up only polynomial amount of space with respect to the database size and the maximum absolute value of the constants of ϕ, thus the guessing step can be done in polynomial time. Then, it is trivial to verify that $v(\phi)$ is *true*.

This proves that the complement of $CERT_C(q, H, D)$ is in NP and consequently that $CERT_C(q, H, D)$ is in coNP. coNP-hardness of $CERT_C(q, H, D)$ follows from Proposition 3.1 of Van der Meyden [85] where a sub-language of dePCL/diPCL, similar to RCL, is considered that contains only the "less-than" predicate over rational or integer constants. Therefore, this lower bound holds for the languages dePCL, diPCL, and RCL as well. □

The following proposition analyzes the data complexity of the certainty problem when the constraint language is TCL.

Proposition 9.13. Let $D = (G, \phi)$ be an RDF$^{\text{i}}$ database, q a query from the fragments of SPARQL $\mathcal{Q}^{C'}_{AUF}$ or $\mathcal{Q}^{C'}_{WD}$, and H an RDF graph. The certainty

problem, $CERT_C(q, H, D)$, is coNP-complete for data complexity when the language of $\mathcal{L}$-constraints is TCL.

Proof. By Theorem 9.9, deciding $CERT_C(q, H, D)$ is equivalent to deciding whether formula (5) is valid. The complement of this problem is to decide whether its negation is unsatisfiable, that is, whether

$$\bigvee_{t \in H} \big(\exists \_\mathbf{l}\big)\big(\phi(\_\mathbf{l}) \wedge \neg\Theta(t, q, D, \_\mathbf{l})\big) \text{ is unsatisfiable.} \tag{10}$$

Formula (10) is unsatisfiable if every disjunct made up from each $t \in H$ is unsatisfiable. Each disjunct constitutes a satisfiability problem for TCL and overall we have $|H|$ such problems to decide. The satisfiability problem for TCL is NP-complete as shown by Renz and Nebel [81], and hence, the problem of deciding $|H|$ such problems remains NP-complete. Consequently, our initial problem $CERT_C(q, H, D)$, which was the complement of satisfiability, is coNP-complete.

A first remark on the above proof is that the satisfiability problem for TCL as defined by Renz and Nebel [81] requires as input a formula given in conjunctive normal formal (CNF) where each conjunct is a disjunction of TCL-constraints between the same pair of variables. The complexity of that problem (resp. for its complement) is not affected by the structure of the input formula. For an arbitrary formula of TCL-constraints Φ, the satisfiability (resp. validity) problem remains NP-complete (resp. coNP-complete). This holds because there exists a satisfiability-preserving encoding of Φ in the propositional modal logic S5 [14, 81, 87], which is a notational variant of the one-variable fragment of first-order logic, the satisfiability problem of which is NP-complete.

A second remark is that satisfiability of Φ does not fix a specific topological space, whereas in our definition of TCL in Section 3, we restrict variables to range over regular closed subsets of $\mathbb{Q}^2$. Therefore, it might be the case that Φ is satisfiable in some topological space other than $\mathbb{Q}^2$. Fortunately, the above result still holds for $\mathbb{Q}^2$ since it has been shown by Renz [80] that if a formula is satisfiable, then it is satisfiable for any dimension $d \geq 1$. □

A similar result to the above can be established for the data complexity of the certainty problem when the constraint language is PCL. This is formally stated in the proposition that follows.

Proposition 9.14. Let $D = (G, \phi)$ be an RDF$^{\text{i}}$ database, q a query from the fragments of SPARQL $\mathcal{Q}^{C'}_{AUF}$ or $\mathcal{Q}^{C'}_{WD}$, and H an RDF graph. The certainty problem, $CERT_C(q, H, D)$, is coNP-complete for data complexity when the language of $\mathcal{L}$-constraints is PCL.

Proof. Similar to the proof for TCL, we show that the complement of $CERT_C(q, H, D)$, that is, satisfiability of a PCL formula, is NP-complete. First, we observe that satisfiability for TCL is a sub-problem of PCL, i.e., when no polygonal constants are involved. Therefore, satisfiability for PCL is NP-hard. Next, we show that satisfiability for PCL is in NP.

Let $\mathcal{M}$ be the non-deterministic Turing machine that decides the problem $\text{CSPSAT}_{\text{S}}(\text{RCC8})$ defined by Li et al. [52] in polynomial time. $\mathcal{M}$ accepts a PCL formula Ψ in CNF iff Ψ is satisfiable. Let Φ be a Boolean combination of atomic PCL-constraints. Clearly, if Φ is satisfiable, then there exists a truth assignment σ for the PCL-constraints $R(u, v)$ appearing in Φ such that $\sigma(\Phi)$ evaluates to *true* propositionally[8]. We devise a non-deterministic Turing machine $\mathcal{M}'$ that runs in polynomial time and accepts Φ iff it is satisfiable. $\mathcal{M}'$ guesses an assignment σ, checks whether $\sigma(\Phi)$ evaluates to *true* in polynomial time, and constructs a PCL formula Ψ in CNF that is given as input to $\mathcal{M}$ to decide.

Formula Ψ is constructed as follows. Let $terms(\Phi)$ denote the variables or constant symbols appearing in Φ and T_σ the set of pairs (u, v) appearing in Φ such that $\sigma(R(u, v))$ is *true* for some RCC-8 relation R. Then, Ψ contains: *a*) the PCL-constraint $R(u, v)$ as a conjunct whenever $(u, v) \in T_\sigma$ and *b*) the conjunct $\bigvee_{R \in \mathcal{B}} R(u, v)$[9] for every pair of terms (u, v) not appearing in Φ or in

[8] Note that if $\sigma(R(u, v))$ is *true*, then $\sigma(S(u, v))$ is $false$ whenever S is different from R and $S(u, v)$ appears in Φ.

[9] $\mathcal{B}$ denotes the set of the 8 base RCC-8 relations $\{\text{DC}, \text{EC}, \text{PO}, \text{NTPP}, \text{NTPPi}, \text{TPP}, \text{TPPi}, \text{EQ}\}$.

T_σ. Therefore,

$$\Psi \equiv \bigwedge_{(u,v)\in T_\sigma} R(u,v) \wedge \bigwedge_{u,v\in terms(\Phi) \text{ and } (u,v)\notin T_\sigma} \bigvee_{R\in\mathcal{B}} R(u,v).$$

Note that Ψ contains at most 8 PCL-constraints for every pair of terms u, v appearing in Φ. Therefore, its size is bounded by $8|terms(\Phi)|^2$. We claim that the following equivalence holds: $\mathcal{M}'$ accepts Φ iff $\mathcal{M}$ accepts Ψ. Therefore, satisfiability for PCL is in NP. In the following we prove the above claim.

Let DNF(Φ) denote the DNF form of Φ. Each disjunct of DNF(Φ) is a conjunction of PCL-constraints appearing in Φ and constitutes a constraint network. Suppose that $\mathcal{M}'$ accepts Φ. Therefore, Φ is satisfiable, and hence, there exists truth assignment σ such that $\sigma(\Phi)$ evaluates to *true* propositionally. Furthermore, there exists a disjunct $\text{DNF}_i(\Phi)$ for Φ that is satisfiable as well and $\sigma(\text{DNF}_i(\Phi))$ is *true*. Consequently, the constraint network corresponding to $\text{DNF}_i(\Phi)$ is consistent. By construction, Ψ contains $\text{DNF}_i(\Phi)$ as a sub-formula, and hence, $\mathcal{M}$ should accept it as well, since the rest of the conjuncts in Ψ not appearing in $\text{DNF}_i(\Phi)$ do not restrict Ψ in any way.

Suppose now that $\mathcal{M}$ accepts Ψ for a guessed assignment σ such that $\sigma(\Phi)$ evaluates to *true*. Therefore, there exist PCL-constraints $R(u,v)$ for all possible pairs of the terms u, v appearing in Ψ such that $\Psi' \equiv \bigwedge_{u,v\in terms(\Phi)} R(u,v)$ is satisfiable and in agreement with σ. By construction, there exists $\text{DNF}_i(\Phi)$ such that $\sigma(\text{DNF}_i(\Phi))$ is *true*, and therefore, $\text{DNF}_i(\Phi)$ should be consistent as a subformula of Ψ'. Consequently, $\mathcal{M}'$ should accept Φ.

We stress here that Li et al. [52] represent polygonal constants in V-representation, i.e., they use points instead of half-spaces like we have done in this work. Since we are in the topological space defined over $\mathbb{Q}^2$, i.e., dimension is fixed, the complexity of CSPSAT_S(RCC8) is not affected when constants are given in H-representation. □

10. Comparison with Related Work

Related work for incomplete information in RDF is discussed in the papers [35, 39, 10] and has been reviewed in the introduction. Here, we give the corresponding comparison with our work.

Comparing our work with Gutierrez et al. [35] and Hurtado and Vaisman [39], 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 Hurtado and Vaisman [39] is similar to ours since it is based on constraints, but, whereas we concentrate on query processing for RDF$^{\text{i}}$, they concentrate more on semantic issues such as temporal graph entailment. It is easy to see that 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 of Koubarakis [47], namely diPCL and dePCL, which we defined in Section 3. An example of this situation is when we want to represent incomplete information about the time an event occurred. This is called *user-defined* time in the temporal database literature and it has not been studied by Gutierrez et al. [35] and Hurtado and Vaisman [39].

The study of incomplete information in RDF undertaken in this paper goes beyond Arenas and Pérez [10] where only the issue of OWA for RDF is investigated. Other cases of incomplete information in RDF can also be investigated using an approach similar to ours. For example, one such case could be the investigation of how blank nodes could be treated as a source of incomplete information employing the various semantics that have been adopted by both practitioners and theoreticians [9, 64, 38] and which deviate from the W3C RDF semantics [37] and SPARQL semantics [76].

Throughout the article, we gave examples of the use of RDF$^{\text{i}}$ in geospatial applications and presented complexity results for the spatial constraint languages defined in Section 3, namely, TCL, PCL, and RCL. It is interesting to compare the expressive power that RDF$^{\text{i}}$ gives us to other recent works that use seman-

tic web data models and languages for geospatial applications. When equipped with a constraint language like TCL, PCL, or RCL, the RDF$^{\text{i}}$ framework goes beyond the proposals of the geospatial extensions of SPARQL, stSPARQL [50] and GeoSPARQL [71], which cannot query incomplete geospatial information. While GeoSPARQL provides a vocabulary for asserting topological relations (the topology vocabulary extension), the semantics and complexity of query evaluation over RDF graphs in this case has not been investigated so far in any detail and remains an open problem.

Incomplete information as it is studied in this article can be also expressed in DLs equipped with concrete domains [11]. Informally, a concrete domain is a formalism for describing concrete qualities of real-world objects, such as temperature, time, and spatial extension, and hence, it allows for reasoning about domains, such as real numbers, time intervals, and spatial regions. A concrete domain is formalized as a first-order theory; it is defined over a set of objects, called *domain*, and a set of predicates interpreted over that domain. An example of a concrete domain is the real numbers with order where the domain is the set of real numbers, $\mathbb{R}$, and the set of predicates consists of the predicate $<$ interpreted by the "less-than" relation over the real numbers. Other examples include the Allen's interval and RCC-8 calculi [36, 63, 72], as well as the cardinal direction calculus [19].

Usually, a DL is glued with a concrete domain both through the TBox and the ABox. The TBox is extended with appropriate concept constructors that provide the means to associate individuals (i.e., objects from the logical domain of the DL) to objects from the concrete domain, and relate such objects through predicates provided by the concrete domain. For example, the concept of hotspot introduced in Section 2, could have been captured in the TBox of a DL extended with a concrete domain for the RCC-8 calculus using the following

concept inclusion axioms[10]:

$$\mathsf{Hotspot} \sqsubseteq \exists \mathsf{correspondsTo}.\mathsf{Fire},$$

$$\mathsf{Hotspot} \sqsubseteq \exists(\mathsf{correspondsTo}\ \mathsf{loc}),\ (\mathsf{loc}).\mathsf{ntpp}.$$

In the second expression above, correspondsTo is a role relating individuals of the logical domain, loc is a partial function, called *concrete feature*, associating individuals to objects of the concrete domain (i.e., spatial regions), and ntpp the predicate of the concrete domain corresponding to the relation "non-tangential proper part" of the RCC-8 calculus. The first inclusion above expresses the fact that a hotspot corresponds to a fire, while the second expresses the fact that a hotspot should spatially contain a fire.

Similarly, the ABox is extended by allowing assertions for associating individuals to objects of the concrete domain (e.g., using the concrete feature loc introduced above) and stating relations that hold between the objects of the concrete domain (e.g., using the predicate ntpp introduced above). Therefore, the integration of a DL with a concrete domain results in very expressive formalisms for knowledge representation and reasoning and is very beneficial, since all the results of the underlying theory of the concrete domain can be reused in the context of the DL.

DLs with concrete domains are far more expressive than RDF$^{\text{i}}$, because the TBox can define concepts based on the relations of the concrete objects associated with the individuals of the DL. Therefore, it goes beyond the terminological knowledge that can be captured in RDFS. As such, the works considering equipping a DL with a concrete domain for Allen's interval calculus or RCC-8 calculus [36, 63, 73, 72] go beyond RDF$^{\text{i}}$ when equipped with diPCL/dePCL or TCL. However, the high expressivity of DLs with concrete domains does not come without problems. The complexity of concept satisfiability could be as high as PSPACE-complete or even NEXPTIME for RCC-8 [62], while decidability of concept satisfiability can be easily lost when considering general

[10] Please note that in our examples we have not considered hotspots to have a spatial extent.

TBoxes, so one should restrict the expressiveness of the TBox or that of the concrete domain [62]. On the other hand, when considering DLs with an empty TBox, it is easy to see that their expressiveness matches that of RDF$^{\text{i}}$. In this respect, there are two DL reasoners that offer such reasoning capabilities, which we review in the following.

PelletSpatial [83] is a hybrid spatial reasoner that provides RCC-8 and OWL 2 reasoning and querying capabilities. In PelletSpatial, spatial relations are separated from OWL 2 relations providing a hybrid reasoner for both spatial and thematic data. Spatial relations are managed as an RCC-8 constraint network. Conjunctive query answering in PelletSpatial requires two phases: *a*) evaluating spatial query atoms over the constraint network by employing a path-consistency algorithm, and *b*) further constraining the set of bindings such that the non-spatial query atoms are satisfied. Compared to the RDF$^{\text{i}}$ framework, PelletSpatial corresponds to RDF$^{\text{i}}$ databases with a conjunction of TCL-constraints as a global constraint. Compared to our extension of SPARQL, the query language of PelletSpatial computes certain answers for SELECT queries using only the operators AND and FILTER with conjunctions of TCL-constraints allowed as expressions in FILTER graph patterns. The representational and querying power of RDF$^{\text{i}}$ when $\mathcal{L}$ is PCL is greater than the one of PelletSpatial since PCL is more expressive than TCL. However, PelletSpatial offers OWL representation and reasoning that is not offered by RDF$^{\text{i}}$.

Likewise, the DL reasoner RacerPro[11] can associate an ABox with a spatial representation layer, called *substrate*, to provide spatial reasoning facilities [86]. RacerPro implements the RCC substrate that offers representation and querying facilities for RCC networks. The RCC substrate offers two sets of RCC relations: the set of RCC-5 relations and the set of RCC-8 relations. Fixing the RCC substrate to be used, one can then use it to assert topological relations between ABox individuals. Moreover, disjunctions of these relations can be used to represent indefinite knowledge regarding the spatial relation of two

[11] http://www.racer-systems.com/

individuals. RacerPro can check for consistency of RCC networks as well as query knowledge entailed by an RCC network, the last feature of which is not present in PelletSpatial.

For efficiency reasons, the aforementioned spatial DL reasoners 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 of interest to this important semantic web area as well.

Recently there has also been an increased interest in extending the expressivity of Datalog, so that the resulting language is appropriate for handling incomplete information of two kinds: indefinite knowledge and missing values. Of particular importance are the works by Calí et al. [16, 17] and Leone et al. [5]. Calí et al. [16] proposed the extension Datalog$^{\pm}$ as a unified framework for query answering and reasoning with incomplete data. The three main extensions to Datalog are the provision of existential variables in the head of rules, equalities, as well as the falsum (in symbols $\bot$). Such extensions are enough to handle the expressiveness of other formalisms for knowledge representation and reasoning, such as description logics (e.g., the DL-Lite, $\mathcal{EL}$, and F-Logic Lite families), data exchange formalisms (e.g., tuple-generating dependencies), and graph query languages (e.g., SPARQL). Leone et al. [5] go one step further and extend Datalog$^{\vee}$ [23] (i.e., an extension of Datalog, called Disjunctive Datalog, that allows disjunctions in the head of rules) so that variables appearing in the head of rules can be existentially quantified. The resulting extension is denoted by Datalog$^{\vee,\exists}$ and makes it more expressive than Datalog$^{\pm}$. Leone et al. [5] provide a semantics for Datalog$^{\vee,\exists}$, study which fragments are decidable, study the problem of query answering for such fragments, as well as the corresponding computational complexity. It turns out that the expressivity of Datalog$^{\vee,\exists}$ is very high and it can naturally encode advanced ontology properties such as role transitivity, role hierarchy, role inverse, and concept inclusion axioms with union concepts on the right hand side.

The appearance of existential variables in the head of rules can be used to represent null values, since, after elimination of quantifiers from the rules, those variables play the role of Skolem constants. In addition, permitting rules with disjunctive heads gives the ability to represent missing values ranging over a finite set of values. This kind of incompleteness has been studied extensively in the literature of incomplete relational databases [41, 30, 3, 42] and their logical counterparts studied by Reiter [79], indefinite databases [68], incomplete deductive databases [32, 40, 45], and logic programming [61, 84]. Therefore, we expect that the problem of combining such formalisms with concrete domains so that the resulting extension is decidable will be of interest to the research community of incomplete information in the near future, since as noted above, decidability is very easy to lose when combining DLs with concrete domains even for the simple case of the $\mathcal{ALC}$ language with general TBoxes [62].

Incomplete information has been studied a lot in logic programming as well [61, 84, 22, 88]. Approaches that consider extensions of rules with disjunctive heads [84] and existential quantification [88] or allow variables playing the role of marked null values in facts [22] are similar to the corresponding extensions of Datalog [24, 16, 17, 5]. With the exception of existential quantification, it is easy to see that the rest of the features considered for capturing incomplete information (i.e., classical negation, negation as failure, disjunctive heads) are part of the answer set programming (ASP) paradigm [59, 60]. In this respect, incomplete information had already been studied much earlier [28, 27] in the context of non-monotonic reasoning, before the term ASP had been coined.

The use of constraints for representing knowledge has also been pioneered by constraint logic programming (CLP) [43, 66]. In CLP, variables appearing in the body of rules can be further constrained by constraints attached to the body. Evaluation of such programs can be done using the traditional top-down and bottom-up evaluation methods equipped with a solver for processing the constraints of rules. Typically, such a solver is based on constraint propagation and other techniques developed in the context of constraint satisfaction. Languages diPCL/dePCL and RCL defined in Section 3 are only some examples

of constraint languages that can be employed in such programs. Regarding the capturing of incomplete information, there have also been works aiming at combining non-monotonic reasoning and CLP. Notable works are those by Dix and Stolzenburg [21] who consider disjunctive logic programs extended with inequality constraints, and Marek and Truszczyński [65] and more recently Mellarkod et al. [67] who study how CLP and ASP can be integrated in general. This last combination of the two paradigms comes under the title constraint answer set programming (CASP) and is very fruitful both from a theoretical point of view and a practical alike, out of which a number of CASP solvers have sprung up. The recent article of Lierler [58] discusses the state-of-the-art in CASP relating the two perspectives.

Clearly, all approaches to incomplete information that are based on rules go beyond RDF$^{\mathrm{i}}$ and are more expressive. On the other hand, since the RDF$^{\mathrm{i}}$ framework is parameterized with a constraint language, it provides a very expressive framework that can handle incomplete information in a general way. The only requirement RDF$^{\mathrm{i}}$ imposes on the constraint language is the existence of an equality predicate.

Although related work in relational databases has been reviewed throughout the article, in the following, we list some prominent and more recent works on incomplete information. To a large extent, the study of incomplete information in databases is being kept active due to the contributions of Libkin and his colleagues [54, 55, 29, 56, 57, 13, 82] who are broadly concerned about the following two topics: *a*) A theory of incomplete information that would unify different data models (e.g., relational tables and XML documents [54, 82] or graph data models [13]) and semantics of incompleteness (e.g., certain answers and representation systems, logical theories, information ordering, combined with closed and open world assumptions [56, 55]), and cases in which the developed theory could result in efficient query evaluation procedures [29, 82]. *b*) Practical issues arising with respect to the management of incomplete information from the adopted industry standards, such as the SQL query language [56, 57].

11. Conclusions and Future Work

In this work, we proposed RDF$^{\mathrm{i}}$, a framework that extends RDF with the ability to capture incomplete information using a first-order constraint language $\mathcal{L}$. We gave the formal semantics of RDF$^{\mathrm{i}}$ and formally defined how queries expressed in an extension of SPARQL that modifies the FILTER operator so that constraints of $\mathcal{L}$ can also be included in the condition part can be evaluated over RDF$^{\mathrm{i}}$ databases. Following the work on incomplete relational databases by Imieliński and Lipski [41] and Grahne [30], we first showed that the monotone fragment of CONSTRUCT queries forms a representation system for RDF$^{\mathrm{i}}$ and then defined the corresponding certainty problem for RDF$^{\mathrm{i}}$ and SPARQL. Last, we demonstrated the usefulness of RDF$^{\mathrm{i}}$ in geospatial Semantic Web applications and compared the modeling capabilities of RDF$^{\mathrm{i}}$ with related formalisms found in the literature.

Our future work focuses on the following: 1) exploration of other fragments of SPARQL that can be used to define a representation system for RDF$^{\mathrm{i}}$, 2) identification of subclasses of the various spatial and temporal constraint languages $\mathcal{L}$ we considered for which the certainty problem is tractable, and 3) extension of the RDF$^{\mathrm{i}}$ framework so that incomplete information arises in the subject and object positions of triples. Regarding this last item, there is already a related line of research on querying incomplete graph data [13]. Therefore, it would be interesting to investigate how RDF$^{\mathrm{i}}$ relates to this body of work.

12. Acknowledgments

This work has been funded by the FP7 project TELEIOS (257662). We are very grateful to all of the reviewers for their constructive comments which improved the article in terms of technical merit and presentation.

Appendix A. Proof of Theorem 8.15

Before going into the details of the proof, we need to show a proposition that the proof employs. The proposition shows that the result of applying a valuation

to the join of two possibly compatible conditional mappings is the same as applying first the valuation to the conditional mappings and then computing their join as in standard RDF.

Proposition A.1. Let $v : U \to C$ be a valuation and $\mu_1 = (\nu_1, \theta_1)$, $\mu_2 = (\nu_2, \theta_2)$ be two possibly compatible conditional mappings such that $\mu_1 \bowtie \mu_2 = (\nu_3, \theta_3)$. Then

$$v(\mu_1 \bowtie \mu_2) = v(\mu_1) \bowtie v(\mu_2)$$

whenever these mappings are defined (i.e., whenever $\mathbf{M}_{\mathcal{L}} \models v(\theta_3)$ and therefore $\mathbf{M}_{\mathcal{L}} \models v(\theta_1)$ and $\mathbf{M}_{\mathcal{L}} \models v(\theta_2)$).

The proof follows easily from the definition of join for conditional mappings and is omitted.

Proof for $\mathcal{Q}^{C'}_{AUF}$

The proof that the triple $\langle \mathcal{D}, Rep, \mathcal{Q}^{C'}_{AUF} \rangle$ is a representation system continues here by showing that for every valuation v such that $\mathbf{M}_{\mathcal{L}} \models v(\phi)$ equality $v([\![P]\!]_D) = [\![P]\!]_{v(D)}$ holds. This equation is proved in the following using induction on the structure of graph patterns P of $\mathcal{Q}^{C'}_{AUF}$.

P is (s, p, o) (base case):

We shall prove that $v([\![P]\!]_D) = [\![P]\!]_{v(D)}$. Let $\mu \in v([\![P]\!]_D)$. Then, there exists a conditional mapping $\mu' = (\nu', \theta') \in [\![P]\!]_D$ such that $v(\mu') = \mu$ and $\mathbf{M}_{\mathcal{L}} \models v(\theta')$. We now distinguish two cases corresponding to Definition 6.13 (1):

(i) In this case $o \in C$. Therefore, $dom(\nu')$ does not contain any special query variable, hence the application of v to μ' leaves ν' unchanged. In other words $\mu = v(\mu') = \nu'$.

Now we have two cases corresponding to the two sets making up $[\![P]\!]_D$.

If $\mu = \nu'$ is an element of the first set, then $\mu'(P) \in G$. Since $\mathbf{M}_{\mathcal{L}} \models v(\theta')$, this is written as $v(\mu'(P)) \in v(G)$, and because $\mathbf{M}_{\mathcal{L}} \models v(\phi)$, this is

equivalent to $v(\mu'(P)) \in v(D)$. Since also $v(\mu') = \mu$, we have $\mu(P) \in v(D)$ and hence $\mu \in [\![P]\!]_{v(D)}$.

If $\mu = \nu'$ is an element of the second set then θ' is $\theta \wedge (\_l \text{ EQ } o)$. Since $\mathbf{M}_{\mathcal{L}} \models v(\theta')$, we have $\mathbf{M}_{\mathcal{L}} \models v(\theta)$ and $\mathbf{M}_{\mathcal{L}} \models v(\_l \text{ EQ } o)$. From the second set of Definition 6.13 (1) that makes up $[\![P]\!]_D$, we have $((\mu(s), \mu(p), \_l), \theta) \in G$. Since $\mathbf{M}_{\mathcal{L}} \models v(\theta)$, we can apply v to the above and get

$$v((\mu(s), \mu(p), \_l)) \in v(G).$$

Since also $\mathbf{M}_{\mathcal{L}} \models v(\phi)$ and $\mathbf{M}_{\mathcal{L}} \models v(\_l \text{ EQ } o)$ we get

$$(\mu(s), \mu(p), o) \in v(D)$$

which is equivalently written as $\mu(P) \in v(D)$ or $\mu \in [\![P]\!]_{v(D)}$.

(ii) In this case $o \in I \cup L \cup V$. Therefore $\nu'(o) \in I \cup B \cup L \cup U \cup C$ and $\mu'(P) \in G$. Since $\mathbf{M}_{\mathcal{L}} \models v(\theta')$, we can apply v to the previous relation and get $v(\mu'(P)) \in v(G)$. Because also $\mathbf{M}_{\mathcal{L}} \models v(\phi)$, we have $v(\mu'(P)) \in v(D)$. The latter fact together with the fact that $v(\mu') = \mu$ gives that $\mu(P) \in v(D)$ and hence $\mu \in [\![P]\!]_{v(D)}$.

This establishes the fact that $v([\![P]\!]_D) \subseteq [\![P]\!]_{v(D)}$. The other direction of the proof is similar and goes as follows.

Let $\mu \in [\![P]\!]_{v(D)}$. Then, $\mu(P) \in v(D)$. We now distinguish two cases corresponding to Definition 6.13 (1):

(i) In this case $o \in C$. Then, $dom(\mu)$ does not contain any special query variable. Since $\mu(P) \in v(D)$, there exists a conditional triple $((\mu(s), \mu(p), x), \theta) \in G$ such that $\mathbf{M}_{\mathcal{L}} \models v(\theta)$ and $v(x) = o$.

Now, we have two cases for x corresponding to the two sets making up $[\![P]\!]_D$ in Definition 6.13 (1):

- x is o. Then, a conditional mapping $\mu' = (\mu, \theta)$ is an element of the first set, i.e., $\mu' \in [\![P]\!]_D$. Since $\mathbf{M}_{\mathcal{L}} \models v(\theta)$, we can apply v to

relation $\mu' \in [\![P]\!]_D$ and get $v(\mu') \in v([\![P]\!]_D)$. Because $dom(\mu)$ does not contain any special query variable the application of v to μ' leaves μ' unchanged. Therefore, $v(\mu') \in v([\![P]\!]_D)$ becomes $\mu \in v([\![P]\!]_D)$.

- x is $\_l$. Then, a conditional mapping $\mu' = (\mu, \theta \wedge \_l \text{ EQ } o)$ is an element of the second set, i.e., $\mu' \in [\![P]\!]_D$. Since $v(\_l) = v(x) = o$, we have $\mathbf{M}_{\mathcal{L}} \models v(\_l \text{ EQ } o)$. Because also $\mathbf{M}_{\mathcal{L}} \models v(\theta)$, it holds that $\mathbf{M}_{\mathcal{L}} \models (\theta \wedge \_l \text{ EQ } o)$, and hence we can apply v to relation $\mu' \in [\![P]\!]_D$ and get $v(\mu') \in v([\![P]\!]_D)$. Because $dom(\mu)$ does not contain any special query variable the application of v to μ' leaves μ' unchanged. Therefore, $v(\mu') \in v([\![P]\!]_D)$ becomes $\mu \in v([\![P]\!]_D)$.

(ii) In this case $o \in I \cup L \cup V$. We have two cases to consider.

If $o \in I \cup L \cup V_n$, then $dom(\mu)$ does not contain any special query variable. Since $\mu(P) \in v(D)$, there exists a conditional triple $(\mu(P), \theta) \in G$ such that $\mathbf{M}_{\mathcal{L}} \models v(\theta)$. By the first set making up $[\![P]\!]_D$, the conditional mapping $\mu' = (\mu, \theta)$ is an element of $[\![P]\!]_D$, that is, $\mu' \in [\![P]\!]_D$. Since $\mathbf{M}_{\mathcal{L}} \models v(\theta)$, we can apply valuation v to this expression and get $v(\mu') \in v([\![P]\!]_D)$ that is equivalent to $\mu \in v([\![P]\!]_D)$, since the application of v to μ' leaves μ' (and μ) unchanged.

Now if $o \in V_s$, there exists a conditional mapping $\mu' = (\nu', \theta)$ such that μ' and μ are possibly compatible, $dom(\mu') = dom(\mu)$, and $\mathbf{M}_{\mathcal{L}} \models v(\theta)$. The conditional mapping μ' is such that either $\nu' = \mu$ or $\nu'(x) = \mu(x)$ for every $x \in dom(\mu) \setminus \{o\}$ and $\nu'(o) \in U$ with $v(\nu'(o)) = \mu(o)$. In either case $\mu(P) \in v(D)$ implies $v(\mu'(P)) \in v(D)$, from which eliminating v, we get $\mu'(P) \in G$, or equivalently $\mu' \in [\![P]\!]_D$. Applying v to the last expression we have that $v(\mu') \in v([\![P]\!]_D)$ and thus $\mu \in v([\![P]\!]_D)$.

Inductive step:

- P is P_1 AND P_2.

We have $v([\![P_1]\!]_D) = [\![P_1]\!]_{v(D)}$ and $v([\![P_2]\!]_D) = [\![P_2]\!]_{v(D)}$ from the inductive hypothesis. We will prove that $v([\![P_1 \text{ AND } P_2]\!]_D) = [\![P_1 \text{ AND } P_2]\!]_{v(D)}$.

Let $\mu \in v([\![P_1 \text{ AND } P_2]\!]_D)$. Therefore there exists a conditional mapping $\mu' = (\nu', \theta') \in [\![P_1 \text{ AND } P_2]\!]_D$ such that $\mu = v(\mu')$ and $\mathbf{M}_{\mathcal{L}} \models v(\theta')$. Because $[\![P_1 \text{ AND } P_2]\!]_D = [\![P_1]\!]_D \bowtie [\![P_2]\!]_D$, there exist possibly compatible conditional mappings $\mu'_1 = (\nu'_1, \theta'_1)$ and $\mu'_2 = (\nu'_2, \theta'_2)$ such that $\mu' = \mu'_1 \bowtie \mu'_2$, $\mu'_1 \in [\![P_1]\!]_D$, and $\mu'_2 \in [\![P_2]\!]_D$. Because of Proposition A.1 and the fact that $\mathbf{M}_{\mathcal{L}} \models v(\theta')$, we have

$$\mu = v(\mu') = v(\mu'_1 \bowtie \mu'_2) = v(\mu'_1) \bowtie v(\mu'_2).$$

Since $\mathbf{M}_{\mathcal{L}} \models v(\theta')$ it also holds $\mathbf{M}_{\mathcal{L}} \models v(\theta'_1)$ and $\mathbf{M}_{\mathcal{L}} \models v(\theta'_2)$. Therefore, $v(\mu'_1) \in v([\![P_1]\!]_D)$ and $v(\mu'_2) \in v([\![P_2]\!]_D)$. Notice also that because μ'_1 and μ'_2 are possibly compatible, $v(\mu'_1)$ and $v(\mu'_2)$ are compatible. Therefore,

$$v(\mu'_1) \bowtie v(\mu'_2) \in v([\![P_1]\!]_D) \bowtie v([\![P_2]\!]_D)$$

which is equivalent to $\mu \in v([\![P_1]\!]_D) \bowtie v([\![P_2]\!]_D)$. From the equalities of the inductive hypothesis, we now get $\mu \in \left([\![P_1]\!]_{v(D)} \bowtie [\![P_2]\!]_{v(D)}\right)$ which is equivalent to $\mu \in [\![P_1 \text{ AND } P_2]\!]_{v(D)}$.

This proof establishes that

$$v([\![P_1 \text{ AND } P_2]\!]_D) \subseteq [\![P_1 \text{ AND } P_2]\!]_{v(D)}.$$

The other direction of the proof is similar and goes as follows.

Let μ be a mapping such that $\mu \in [\![P_1 \text{ AND } P_2]\!]_{v(D)}$. Then $\mu \in \left([\![P_1]\!]_{v(D)} \bowtie [\![P_2]\!]_{v(D)}\right)$, which due to the inductive hypothesis gives us

$$\mu \in v([\![P_1]\!]_D) \bowtie v([\![P_2]\!]_D).$$

Therefore, there exist compatible mappings $\mu_1 \in v([\![P_1]\!]_D)$ and $\mu_2 \in v([\![P_2]\!]_D)$ such that $\mu = \mu_1 \bowtie \mu_2$. Thus, there exist conditional mappings $\mu'_1 = (\nu'_1, \theta'_1) \in [\![P_1]\!]_D$ and $\mu'_2 = (\nu'_2, \theta'_2) \in [\![P_2]\!]_D$ such that $\mu_1 = v(\mu'_1)$, $\mu_2 = v(\mu'_2)$, $\mathbf{M}_{\mathcal{L}} \models v(\theta'_1)$ and $\mathbf{M}_{\mathcal{L}} \models v(\theta'_2)$. Notice also that μ'_1 and μ'_2 are possibly compatible.

From Proposition A.1 and the fact that $\mu = \mu_1 \bowtie \mu_2$, we have

$$\mu = \mu_1 \bowtie \mu_2 = v(\mu'_1) \bowtie v(\mu'_2) = v(\mu'_1 \bowtie \mu'_2).$$

Because $\mu'_1 \in [\![P_1]\!]_D$, $\mu'_2 \in [\![P_2]\!]_D$, and μ'_1, μ'_2 are possibly compatible, we have

$$\mu'_1 \bowtie \mu'_2 \in [\![P_1]\!]_D \bowtie [\![P_2]\!]_D.$$

Now let $\mu' = (\nu', \theta')$ be a conditional mapping such that $\mu' = \mu'_1 \bowtie \mu'_2$. Since $\mathbf{M}_\mathcal{L} \models v(\theta'_1)$ and $\mathbf{M}_\mathcal{L} \models v(\theta'_2)$, the definition of join of two compatible mappings gives us $\mathbf{M}_\mathcal{L} \models v(\theta')$. Therefore we can apply the valuation v to μ' and get

$$v(\mu') \in v([\![P_1]\!]_D \bowtie [\![P_2]\!]_D).$$

From this and the fact that $v(\mu') = v(\mu'_1 \bowtie \mu'_2) = \mu$ we get $\mu \in v([\![P_1 \text{ AND } P_2]\!]_D)$.

- P is P_1 UNION P_2.

  We have $v([\![P_1]\!]_D) = [\![P_1]\!]_{v(D)}$ and $v([\![P_2]\!]_D) = [\![P_2]\!]_{v(D)}$ from the inductive hypothesis. We will prove that

  $$v([\![P_1 \text{ UNION } P_2]\!]_D) = [\![P_1 \text{ UNION } P_2]\!]_{v(D)}.$$

  A mapping μ is in $[\![P_1 \text{ UNION } P_2]\!]_{v(D)}$ iff $\mu \in [\![P_1]\!]_{v(D)} \cup [\![P_2]\!]_{v(D)}$, which due to the inductive hypothesis is equivalent to $\mu \in v([\![P_1]\!]_D) \cup v([\![P_2]\!]_D)$, which can be seen to be equivalent to $\mu \in v([\![P_1]\!]_D \cup [\![P_2]\!]_D)$, which is equivalent to $\mu \in v([\![P_1 \text{ UNION } P_2]\!]_D)$.

- P is P_1 FILTER R.

  We have $v([\![P_1]\!]_D) = [\![P_1]\!]_{v(D)}$ from the inductive hypothesis. We will prove that

  $$v([\![P_1 \text{ FILTER } R]\!]_D) = [\![P_1 \text{ FILTER } R]\!]_{v(D)}$$

  Without loss of generality, we give the proof only for the case of filters that are atomic $\mathcal{L}$-constraints (Definition 6.16). Let μ be in $[\![P_1 \text{ FILTER } R]\!]_{v(D)}$. By definition, this is equivalent to $\mu \in [\![P_1]\!]_{v(D)}$ and $\mu \models R$. From the inductive hypothesis, we now have $\mu \in v([\![P_1]\!]_D)$. Thus, there exists a conditional mapping $\mu' = (\nu', \theta') \in [\![P_1]\!]_D$ such that $v(\mu') = \mu$ and $\mathbf{M}_\mathcal{L} \models v(\theta')$.

Let now $\mu_1 = (\nu', \theta_1)$ be a conditional mapping with $\theta_1 = \theta' \wedge \nu'(R)$. Because $\mu \models R$, we have $\mathbf{M}_{\mathcal{L}} \models \mu(R)$. Therefore $\mathbf{M}_{\mathcal{L}} \models v(\nu'(R))$ since $v(\mu') = \mu$. Now notice that because $\mathbf{M}_{\mathcal{L}} \models v(\nu'(R))$ and $\mathbf{M}_{\mathcal{L}} \models v(\theta')$, we have $\mathbf{M}_{\mathcal{L}} \models v(\theta_1)$. Therefore $v(\mu_1)$ is well defined and we have $v(\mu_1) = v(\mu') = \mu$.

The way μ_1 and μ' have been defined above, together with the definition of the evaluation of FILTER graph patterns give us

$$\mu_1 \in [\![P_1 \text{ FILTER } R]\!]_D.$$

We can apply valuation v to the above relation and get $v(\mu_1) \in v([\![P_1 \text{ FILTER } R]\!]_D)$ that is equivalent to $\mu \in v([\![P_1 \text{ FILTER } R]\!]_D)$.

This proof establishes that

$$[\![P_1 \text{ FILTER } R]\!]_{v(D)} \subseteq v([\![P_1 \text{ FILTER } R]\!]_D).$$

The other direction of the proof is similar and goes as follows.

Let μ be a mapping in $v([\![P_1 \text{ FILTER } R]\!]_D)$. Then there exists a conditional mapping $\mu_1 = (\nu_1, \theta_1) \in [\![P_1 \text{ FILTER } R]\!]_D$ such that $v(\mu_1) = \mu$ and $\mathbf{M}_{\mathcal{L}} \models v(\theta_1)$. Therefore, from the definition of FILTER evaluation there exists a conditional mapping $\mu_2 = (\nu_1, \theta_2)$ such that $\mu_2 \in [\![P_1]\!]_D$, where $\theta_1 = \theta_2 \wedge \nu_1(R)$. Since $\mathbf{M}_{\mathcal{L}} \models v(\theta_1)$, it holds that $\mathbf{M}_{\mathcal{L}} \models v(\theta_2)$ and $\mathbf{M}_{\mathcal{L}} \models v(\nu_1(R))$. Thus $v(\mu_2) = v(\mu_1) = \mu \in v([\![P_1]\!]_D)$. Now using the inductive hypothesis, we have $\mu \in [\![P_1]\!]_{v(D)}$. Because $\mathbf{M}_{\mathcal{L}} \models v(\nu_1(R))$ and $\mu = v(\mu_1)$, we have $\mathbf{M}_{\mathcal{L}} \models \mu(R)$. Thus we also have $\mu \models R$. Hence $\mu \in [\![P_1 \text{ FILTER } R]\!]_{v(D)}$.

The above proves that the triple $\langle \mathcal{D}, Rep, \mathcal{Q}_{AUF}^{C'} \rangle$ is a representation system.

Proof for $\mathcal{Q}_{WD}^{C'}$

The proof that the triple $\langle \mathcal{D}, Rep, \mathcal{Q}_{WD}^{C'} \rangle$ is also a representation system is the same as the previous one, while it differs only in the inductive step for the OPT operator. Thus, in this case, P is P_1 OPT P_2.

We have $v([\![P_1]\!]_D) = [\![P_1]\!]_{v(D)}$ and $v([\![P_2]\!]_D) = [\![P_2]\!]_{v(D)}$ from the inductive hypothesis. We need to prove $v([\![P_1 \text{ OPT } P_2]\!]_D) = [\![P_1 \text{ OPT } P_2]\!]_{v(D)}$ or equivalently

$$v([\![P_1 \text{ OPT } P_2]\!]_D) = ([\![P_1]\!]_{v(D)} \bowtie [\![P_2]\!]_{v(D)}) \cup ([\![P_1]\!]_{v(D)} \setminus [\![P_2]\!]_{v(D)}). \tag{A.1}$$

Let $\mu \in v([\![P_1 \text{ OPT } P_2]\!]_D)$. Then, there exists a conditional mapping $\mu' = (\nu', \theta') \in [\![P_1 \text{ OPT } P_2]\!]_D$ such that $\mu = v(\mu')$ and $\mathbf{M}_{\mathcal{L}} \models v(\theta')$. Since $[\![P_1 \text{ OPT } P_2]\!]_D = [\![P_1]\!]_D \bowtie [\![P_2]\!]_D = ([\![P_1]\!]_D \bowtie [\![P_2]\!]_D) \cup ([\![P_1]\!]_D \setminus [\![P_2]\!]_D)$, then

$$\mu' \in ([\![P_1]\!]_D \bowtie [\![P_2]\!]_D) \text{ or } \mu' \in ([\![P_1]\!]_D \setminus [\![P_2]\!]_D).$$

For the former case, i.e., $\mu' \in ([\![P_1]\!]_D \bowtie [\![P_2]\!]_D)$ the proof is the same as in the proof of Theorem 8.15, hence we finally get that $\mu \in ([\![P_1]\!]_{v(D)} \bowtie [\![P_2]\!]_{v(D)})$ and thus from formula (A.1) we have $\mu \in [\![P_1 \text{ OPT } P_2]\!]_{v(D)}$. For the latter case, i.e., $\mu' \in ([\![P_1]\!]_D \setminus [\![P_2]\!]_D)$, and the definition of difference for sets of conditional mappings we distinguish two sub-cases:

1. $\mu' \in [\![P_1]\!]_D$ and for all $\mu_2 \in [\![P_2]\!]_D$, μ' and μ_2 are not possibly compatible. Since $\mathbf{M}_{\mathcal{L}} \models v(\theta')$, we can apply valuation v to μ' and get

   $$\mu = v(\mu') \in v([\![P_1]\!]_D).$$

   Since also every conditional mapping μ_2 of $[\![P_2]\!]_D$ is not compatible to μ' — this is implied from not being possibly compatible to μ' — $v(\mu')$ is not compatible to every mapping $\mu'' \in v([\![P_2]\!]_D)$. Therefore, $v(\mu') \in (v([\![P_1]\!]_D) \setminus v([\![P_2]\!]_D))$ which from our hypothesis is equivalent to $\mu \in ([\![P_1]\!]_{v(D)} \setminus [\![P_2]\!]_{v(D)})$. Hence, from formula (A.1) we have $\mu \in [\![P_1 \text{ OPT } P_2]\!]_{v(D)}$.

2. μ' is the conditional mapping (ν', θ') and there exists a conditional mapping $\mu'' = (\nu', \theta) \in [\![P_1]\!]_D$ such that

   - μ'' is not compatible to some mappings of $[\![P_2]\!]_D$ and
   - for the rest of mappings $\mu_i = (\nu_i, \theta_i) \in [\![P_2]\!]_D$, μ'' and μ_i are possibly compatible and

   $$\theta' = \theta \wedge \left(\theta_i \supset \bigvee_{x \in dom(\mu'') \cap dom(\mu_i) \cap V_s} \neg(\mu''(x) \text{ EQ } \mu_i(x)) \right).$$

Since $\mathbf{M}_{\mathcal{L}} \models v(\theta')$, we can apply valuation v to μ' and get

$$\mu = v(\mu') \in v([\![P_1]\!]_D \setminus [\![P_2]\!]_D)$$

which can be written as

$$\mu = v(\mu') \in (v([\![P_1]\!]_D) \setminus v([\![P_2]\!]_D)) .$$

To see this, notice that the above relation holds if and only if $v(\mu') \in v([\![P_1]\!]_D)$ and it is not compatible to every mapping $v(\mu_2)$ of $v([\![P_2]\!]_D)$. Since $\mathbf{M}_{\mathcal{L}} \models v(\theta')$ it holds $\mathbf{M}_{\mathcal{L}} \models v(\theta)$ and thus $v(\mu') = v(\mu'') \in v([\![P_1]\!]_D)$. Let us now take a mapping μ_2 in $[\![P_2]\!]_D$. Then, *a*) either μ'', and consequently μ', is not compatible to μ_2, or *b*) μ'', and consequently μ', is possibly compatible to μ_2. For the first case $v(\mu')$ is also not compatible to $v(\mu_2)$. For the second case $v(\mu')$ is also not compatible to $v(\mu_2)$. To see this, notice that $v(\mu')$ and $v(\mu_2)$ become compatible only when $v(\mu'(x)) = v(\mu_2(x))$ for $x \in dom(\mu') \cap dom(\mu_2)$. In such cases, however, $\mathbf{M}_{\mathcal{L}} \not\models \theta'$ and thus $v(\mu') \notin v([\![P_1]\!]_D)$.

Continuing the proof, from our hypothesis, the relation

$$\mu = v(\mu') \in (v([\![P_1]\!]_D) \setminus v([\![P_2]\!]_D))$$

now becomes $\mu \in ([\![P_1]\!]_{v(D)} \setminus [\![P_2]\!]_{v(D)})$ and thus from formula (A.1) we get $\mu \in [\![P_1 \text{ OPT } P_2]\!]_{v(D)}$.

This proves that $v([\![P_1 \text{ OPT } P_2]\!]_D) \subseteq [\![P_1 \text{ OPT } P_2]\!]_{v(D)}$. The other direction of the proof is similar.

Appendix B. Proof for Proposition 8.14

Let $[\![q]\!]_D$ be the RDF$^{\text{i}}$ database $D' = (G', \phi)$ where

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

Then $v([\![q]\!]_D)$ is the RDF graph $v(D')$ where

$$v(D') = \bigcup_{\mu=(\nu,\theta)\in[\![P]\!]_D} \Big\{ v(t) \mid (t,\theta) \in \mu\big(f_\mu(E)\big) \text{ and } t \in \big((I \cup B) \times I \times T\big) \text{ and } \mathbf{M}_{\mathcal{L}} \models v(\theta) \Big\}. \tag{B.1}$$

Likewise, let $[\![q]\!]_{v(D)}$ be the RDF graph H. According to the definition of the evaluation of CONSTRUCT queries on RDF graphs [74], H is the set

$$H = \bigcup_{\mu\in[\![P]\!]_{v(D)}} \big\{ \mu(f_\mu(E)) \cap ((I \cup B) \times I \times T^C \big\}. \tag{B.2}$$

To prove our proposition, we have to show that $H = v(D')$.

Let $t \in H$ be an RDF triple. Then, there exists a mapping $\mu \in [\![P]\!]_{v(D)}$ such that

$$t \in (\mu(f_\mu(E)) \cap ((I \cup B) \times I \times T^C)). \tag{B.3}$$

From our assumption that $v([\![P]\!]_D) = [\![P]\!]_{v(D)}$, we have $\mu \in v([\![P]\!]_D)$. Therefore, there exists a conditional mapping $\mu' = (\nu', \theta') \in [\![P]\!]_D$ such that $\mathbf{M}_{\mathcal{L}} \models v(\theta')$ and $\mu = v(\mu')$, and hence, relation (B.3) can equivalently be written as

$$t \in \Big\{ v(t') \mid (t',\theta') \in \mu'(f_\mu(E)) \text{ and } v(t') \in \big((I \cup B) \times I \times T^C\big) \Big\}.$$

Notice now that for a conditional triple t, relation $t \in \big((I \times B) \times I \times T\big)$ is equivalent to relation $v(t) \in \big((I \times B) \times I \times T^C\big)$. This equivalence holds because v is a total function from U to C. Therefore, the above relation can be equivalently written as

$$t \in \Big\{ v(t') \mid (t',\theta') \in \mu'(f_\mu(E)) \text{ and } t' \in \big((I \cup B) \times I \times T\big) \Big\}. \tag{B.4}$$

From (B.1) and since $\mathbf{M}_{\mathcal{L}} \models v(\theta')$ and $\mu' \in [\![P]\!]_D$, we have

$$\Big\{ v(t') \mid (t',\theta') \in \mu'(f_{\mu'}(E)) \text{ and } t' \in \big((I \cup B) \times I \times T\big) \Big\} \subseteq v(D').$$

Observe now that from relation (B.4) and by fixing the renaming function $f_{\mu'}$ to rename blank nodes like f_μ does, we get $t \in v(D')$. Note that this fixing respects Definition 6.26, since $blank(D) = blank(G) = blank(v(D))$.

Hence, we showed that every triple of H is a triple of $v(D')$.

The other direction of the proof is similar and goes as follows. Let $t \in v(D')$. Then, there exists a conditional mapping $\mu = (\nu, \theta) \in [\![P]\!]_D$ and a conditional triple $(t', \theta) \in G'$ such that $\mathbf{M}_{\mathcal{L}} \models v(\theta)$ and $v(t') = t$. From (B.1) we then have

$$(t', \theta) \in \mu(f_\mu(E)) \text{ and } t' \in \big((I \cup B) \times I \times T\big). \tag{B.5}$$

Since $\mathbf{M}_{\mathcal{L}} \models v(\theta)$, $v(\mu)$ is defined and thus we have $v(\mu) \in v([\![P]\!]_D)$, which from our assumption that $v([\![P]\!]_D) = [\![P]\!]_{v(D)}$, we get $\mu' = v(\mu) \in [\![P]\!]_{v(D)}$. Thus, applying valuation v to (B.5), we get

$$v(t') \in v\big(\mu(f_\mu(E))\big) \text{ and } v(t') \in \big((I \cup B) \times I \times T^C\big).$$

Since $\mu' = v(\mu)$ and $v(t') = t$, the above relation becomes

$$t \in \big(\mu'(f_\mu(E)) \cap ((I \cup B) \times I \times T^C)\big).$$

Working as previously, we set f_μ to behave as $f_{\mu'}$ and hence the above relation becomes

$$t \in \big(\mu'(f_{\mu'}(E)) \cap ((I \cup B) \times I \times T^C)\big).$$

From the above and because of (B.2) and $\mu' \in [\![P]\!]_{v(D)}$, we get $t \in H$.

References

[1] Serge Abiteboul and Oliver M. Duschka. Complexity of answering queries using materialized views. In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '98, pages 254–263, 1998.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] Serge Abiteboul, Paris Kanellakis, and Gösta Grahne. On the representation and querying of sets of possible worlds. *Theoretical Computer Science*, 78(1):159 – 187, 1991.

[4] Serge Abiteboul, Luc Segoufin, and Victor Vianu. Representing and Querying XML with Incomplete Information. *ACM Transactions on Database Systems (TODS)*, 31(1):208–254, 2006.

[5] Mario Alviano, Wolfgang Faber, Nicola Leone, and Marco Manna. Disjunctive datalog with existential quantifiers: Semantics, decidability, and complexity issues. *Theory and Practice of Logic Programming*, 12:701–718, 7 2012.

[6] Renzo Angles and Claudio Gutierrez. The expressive power of SPARQL. In *Proceedings of the 7th International Semantic Web Conference*, ISWC '08, pages 114–129, 2008.

[7] Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014.

[8] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *Proceedings of the 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '99, pages 68–79, 1999.

[9] Marcelo Arenas, Mariano Consens, and Alejandro Mallea. Revisiting blank nodes in RDF to avoid the semantic mismatch with SPARQL. In *W3C Workshop: RDF Next Steps, Palo Alto, CA*, 2010.

[10] Marcelo Arenas and Jorge Pérez. Querying semantic web data with SPARQL. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '11, pages 305–316, 2011.

[11] Franz Baader and Philipp Hanschke. A scheme for integrating concrete domains into concept languages. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI '91, pages 452–457, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[12] Pablo Barceló, Leonid Libkin, Antonella Poggi, and Cristina Sirangelo. XML with incomplete information. *Journal of the ACM (JACM)*, 58(1):4:1–4:62, 2010.

[13] Pablo Barceló, Leonid Libkin, and Juan L. Reutter. Querying regular graph patterns. *Journal of the ACM (JACM)*, 61(1):8:1–8:54, 2014.

[14] Brandon Bennett. Modal logics for qualitative spatial reasoning. *Logic Journal of the IGPL*, 4(1):23–45, 1996.

[15] Ronald J. Brachman and Hector J. Levesque. *Knowledge Representation and Reasoning*. Morgan Kaufmann Series in Artificial Intelligence Series. Elsevier, 2004.

[16] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *Proceedings of the 28th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '09, pages 77–86, New York, NY, USA, 2009. ACM.

[17] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. Datalog$^{\pm}$: A unified approach to ontologies and integrity constraints. In *Proceedings of the 12th International Conference on Database Theory*, ICDT '09, pages 14–30, New York, NY, USA, 2009. ACM.

[18] Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '03, pages 260–271, 2003.

[19] Matteo Cristani and Nicoletta Gabrielli. Practical issues of description logics for spatial reasoning. In *Benchmarking of Qualitative Spatial and Temporal Reasoning Systems, Papers from the 2009 AAAI Spring Sympo-*

sium, Technical Report SS-09-02, Stanford, California, USA, March 23-25, 2009, pages 5–10, 2009.

[20] Claire David, Leonid Libkin, and Filip Murlak. Certain answers for XML queries. In *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '10, pages 191–202, 2010.

[21] Jürgen Dix and Frieder Stolzenburg. A framework to incorporate nonmonotonic reasoning into constraint logic programming. *The Journal of Logic Programming*, 37(1–3):47 – 76, 1998.

[22] Fangqing Dong and Laks V.S. Lakshmanan. Intuitionistic interpretation of deductive databases with incomplete information. *Theoretical Computer Science*, 133(2):267 – 306, 1994.

[23] Thomas Eiter, Michael Fink, Gianluigi Greco, and Domenico Lembo. Repair localization for query answering from inconsistent databases. *ACM Transactions on Database Systems (TODS)*, 33(2):10:1–10:51, 2008.

[24] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 9 1997.

[25] Herbert B. Enderton. *A mathematical introduction to logic*. Academic Press, 1972.

[26] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89 – 124, 2005.

[27] Michael Gelfond. Logic programming and reasoning with incomplete information. *Annals of Mathematics and Artificial Intelligence*, 12(1-2):89–116, 1994.

[28] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3-4):365–385, 1991.

[29] Amélie Gheerbrant, Leonid Libkin, and Cristina Sirangelo. Naïve evaluation of queries over incomplete databases. *ACM Transactions on Database Systems*, 39(4):31, 2014.

[30] Gösta Grahne. *The Problem of Incomplete Information in Relational Databases*, volume 554 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1991.

[31] Gösta Grahne. Incomplete information. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 1405–1410. Springer US, 2009.

[32] John Grant and Jack Minker. Answering queries in indefinite databases and the null value problem. *Advances in Computing Research*, 3:247–267, 1986.

[33] Alasdair J. G. Gray, Jason Sadler, Oles Kit, Kostis Kyzirakos, Manos Karpathiotakis, Jean-Paul Calbimonte, Kevin Page, Raúl Garcéa-Castro, Alex Frazer, Ixent Galpin, Alvaro A. A. Fernandes, Norman W. Paton, Oscar Corcho, Manolis Koubarakis, David De Roure, Kirk Martinez, and Asunción Gómez-Pérez. A semantic sensor web for environmental decision support applications. *Sensors*, 11(9):8855–8887, 2011.

[34] Claudio Gutierrez, Carlos A. Hurtado, Alberto O. Mendelzon, and Jorge Pérez. Foundations of semantic web databases. *Journal of Computer and System Sciences (JCSS)*, 77(3):520–541, 2011.

[35] Claudio Gutierrez, Carlos A. Hurtado, and Alejandro Vaisman. Introducing Time into RDF. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 19(2):207–218, 2007.

[36] Volker Haarslev, Carsten Lutz, and Ralf Möller. A description logic with concrete domains and a role-forming predicate operator. *Journal of Logic and Computation*, 9(3):351–384, 1999.

[37] Patrick Hayes and Peter F. Patel-Schneider. RDF 1.1 Semantics. W3C Recommendation 25 February 2014.

[38] Aidan Hogan, Marcelo Arenas, Alejandro Mallea, and Axel Polleres. Everything you always wanted to know about blank nodes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 27(1), 2014.

[39] Carlos A. Hurtado and Alejandro A. Vaisman. Reasoning with Temporal Constraints in RDF. In *Proceedings Workshop on Principles and Practice of Semantic Web Reasoning*, pages 164–178, 2006.

[40] Tomasz Imieliński. Incomplete deductive databases. *Annals of Mathematics and Artificial Intelligence*, 3(2-4):259–293, 1991.

[41] Tomasz Imieliński and Witold Lipski, Jr. Incomplete information in relational databases. *Journal of the ACM (JACM)*, 31(4):761–791, 1984.

[42] Tomasz Imieliński and Kumar V. Vadaparty. Complexity of query processing in databases with OR-objects. In *Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 29-31, 1989, Philadelphia, Pennsylvania, USA*, pages 51–65, 1989.

[43] Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *The Journal of Logic Programming*, 19/20:503–581, 1994.

[44] Paris C. Kanellakis, Gabriel M. Kuper, and Peter Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences (JCSS)*, 51(1):26 – 52, 1995.

[45] Quinzheng Kong and Graham Chen. On deductive databases with incomplete information. *ACM Transactions on Information Systems*, 13(3):354–370, 7 1995.

[46] Manolis Koubarakis. Complexity results for first-order theories of temporal constraints. In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, KR '94, pages 379–390, 1994.

[47] Manolis Koubarakis. Database models for infinite and indefinite temporal information. *Information Systems*, 19(2):141 – 173, 1994.

[48] Manolis Koubarakis. The complexity of query evaluation in indefinite temporal constraint databases. *Theoretical Computer Science*, 171(1–2):25 – 60, 1997.

[49] Manolis Koubarakis and Kostis Kyzirakos. Modeling and querying metadata in the semantic sensor web: The model stRDF and the query language stSPARQL. In *Proceedings of the 7th Extended Semantic Web Conference, ESWC '10*, pages 425–439, 2010.

[50] Kostis Kyzirakos, Manos Karpathiotakis, and Manolis Koubarakis. Strabon: A Semantic Geospatial DBMS. In *Proceedings of the 11th International Semantic Web Conference - Volume Part I*, ISWC '12, pages 295–311, 2012.

[51] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 233–246, 2002.

[52] Sanjiang Li, Weiming Liu, and Shengsheng Wang. Qualitative constraint satisfaction problems: An extended framework with landmarks. *Artificial Intelligence*, 201(0):32 – 58, 2013.

[53] Leonid Libkin. Data exchange and incomplete information. In *Proceedings of the 25th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '06, pages 60–69, 2006.

[54] Leonid Libkin. Incomplete information and certain answers in general data models. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '11, pages 59–70, 2011.

[55] Leonid Libkin. Certain answers as objects and knowledge. In Chitta Baral, Giuseppe De Giacomo, and Thomas Eiter, editors, *Principles of Knowledge*

Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR '14, Vienna, Austria, July 20-24, 2014. AAAI Press, 2014.

[56] Leonid Libkin. Incomplete data: What went wrong, and how to fix it. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '14, pages 1–13, New York, NY, USA, 2014. ACM.

[57] Leonid Libkin. SQL's three-valued logic and certain answers. In Marcelo Arenas and Martín Ugarte, editors, *18th International Conference on Database Theory, ICDT 2015, March 23-27, 2015, Brussels, Belgium*, volume 31 of *LIPIcs*, pages 94–109. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[58] Yuliya Lierler. Relating constraint answer set programming languages and algorithms. *Artificial Intelligence*, 207:1 – 22, 2014.

[59] Vladimir Lifschitz. Answer set planning. In *Logic Programming: The 1999 International Conference, Las Cruces, New Mexico, USA, November 29 - December 4, 1999*, pages 23–37, 1999.

[60] Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1–2):39 – 54, 2002. Knowledge Representation and Logic Programming.

[61] Yuan Liu. Null values in definite programs. In *Proceedings of the 1990 North American Conference on Logic Programming*, pages 273–287, 1990.

[62] Carsten Lutz. Description logics with concrete domains—a survey. In *Advances in Modal Logic, AiML '02*, Toulouse, France, 2002. Final version appeared in Advanced in Modal Logic Volume 4, 2003.

[63] Carsten Lutz and Maja Miličić. A Tableau Algorithm for Description Logics with Concrete Domains and General TBoxes. *Journal of Automated Reasoning*, 38(1-3):227–259, 2007.

[64] Alejandro Mallea, Marcelo Arenas, Aidan Hogan, and Axel Polleres. On blank nodes. In *Proceedings of the 10th International Conference on The Semantic Web - Volume Part I*, ISWC '11, pages 421–437, 2011.

[65] Victor W. Marek and Miroslaw Truszczyński. Stable models and an alternative logic programming paradigm. In Krzysztof R. Apt, Victor W. Marek, Mirek Truszczyński, and David S. Warren, editors, *The Logic Programming Paradigm*, Artificial Intelligence, pages 375–398. Springer Berlin Heidelberg, 1999.

[66] Kim Marriott and Peter J. Stuckey. *Introduction to Constraint Logic Programming*. MIT Press, Cambridge, MA, USA, 1998.

[67] Veena S. Mellarkod, Michael Gelfond, and Yuanlin Zhang. Integrating answer set programming and constraint logic programming. *Annals of Mathematics and Artificial Intelligence*, 53(1-4):251–287, 2008.

[68] Jack Minker. On indefinite databases and the closed world assumption. In D.W. Loveland, editor, *6th Conference on Automated Deduction*, volume 138 of *Lecture Notes in Computer Science*, pages 292–308. Springer Berlin Heidelberg, 1982.

[69] Charalampos Nikolaou and Manolis Koubarakis. Querying Linked Geospatial Data with Incomplete Information. In *Proceedings of the 5th Terra Cognita Workshop on Foundations, Technologies and Applications of the Geospatial Web*, pages 51–61, 2012.

[70] Charalampos Nikolaou and Manolis Koubarakis. Incomplete Information in RDF. In *Proceedings of the 8th International Conference on Web Reasoning and Rule Systems, RR '13*, pages 138–152, 2013.

[71] Open Geospatial Consortium Inc. GeoSPARQL - A geographic query language for RDF data. OpenGIS Implementation Standard, 2010.

[72] Özgür L. Özçep and Ralf Möller. Scalable geo-thematic query answering. In *Proceedings of the 11th International Conference on The Semantic Web - Volume Part I*, ISWC '12, pages 658–673, 2012.

[73] Özgür L. Özçep and Ralf Möller. Combining DL-Lite with spatial calculi for feasible geo-thematic query answering. In *Proceedings of the 2012 International Workshop on Description Logics, DL '12, Rome, Italy, June 7-10, 2012*, 2012.

[74] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics of SPARQL. Technical report, Univ. de Chile, 2006.

[75] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and Complexity of SPARQL. *ACM Transactions on Database Systems (TODS)*, 34(3):16:1–16:45, 2009.

[76] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation 15 January 2008.

[77] Michael O. Rabin. Decidable theories. In *Handbook of mathematical logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 595–629. North-Holland, 1977.

[78] David A. Randell, Zhan Cui, and Anthony G. Cohn. A spatial logic based on regions and connection. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, KR '92, pages 165–176, 1992.

[79] Raymond Reiter. Towards a logical reconstruction of relational database theory. In Michael L. Brodie, John Mylopoulos, and Joachim W. Schmidt, editors, *On Conceptual Modelling*, Topics in Information Systems, pages 191–238. Springer New York, 1984.

[80] Jochen Renz. A canonical model of the region connection calculus. *Journal of Applied Non-Classical Logics*, 12(3-4):469–494, 2002.

[81] Jochen Renz and Bernhard Nebel. On the complexity of qualitative spatial reasoning: A maximal tractable fragment of the region connection calculus. *Artificial Intelligence*, 108(1–2):69 – 123, 1999.

[82] Cristina Sirangelo. *Representing and Querying Incomplete Information: a Data Interoperability Perspective*. Habilitation à diriger des recherches, Ecole Normale Supérieure de Cachan, 12 2014.

[83] Markus Stocker and Evren Sirin. PelletSpatial: A Hybrid RCC-8 and RDF/OWL Reasoning and Query Engine. In *Proceedings of the 5th International Workshop on OWL: Experiences and Directions*, OWLED '09, 2009.

[84] Bonnie Traylor and Michael Gelfond. Representing null values in logic programming. In Anil Nerode and Yu.V. Matiyasevich, editors, *Logical Foundations of Computer Science*, volume 813 of *Lecture Notes in Computer Science*, pages 341–352. Springer Berlin Heidelberg, 1994.

[85] Ron van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *Journal of Computer and System Sciences (JCSS)*, 54(1):113 – 135, 1997.

[86] Michael Wessel and Ralf Möller. Flexible software architectures for ontology-based information systems. *Journal of Applied Logic*, 7(1):75 – 99, 2009. Special Issue: Empirically Successful Computerized Reasoning.

[87] Frank Wolter and Michael Zakharyaschev. Qualitative spatiotemporal representation and reasoning: A computational perspective. In Gerhard Lakemeyer and Bernhard Nebel, editors, *Exploring Artificial Intelligence in the New Millennium*, pages 175–215. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[88] Jia-Huai You, Heng Zhang, and Yan Zhang. Disjunctive logic programs with existential quantification in rule heads. *Theory and Practice of Logic Programming*, 13:563–578, 7 2013.