TELEIOS

FP7-257662

Deliverable

D4.2

An implementation of a temporal and spatial extension of RDF and SPARQL on top of MonetDB - Phase II

Konstantina Bereta, Kallirroi Dogani, George Garbis, Stella Giannakopoulou, Manolis Koubarakis, Kostis Kyzirakos, Charalampos Nikolaou, Michael Sioutis, Panayiotis Smeros, Bakogiannis Dimitrios, Foufoulas Ioannis, Smaragdakis Ioannis,

and

Consortium Members

March 22, 2013

Status: Final
Scheduled Delivery Date: 28 February, 2013

Executive Summary

In this deliverable we present the second phase of the implementation of our temporal and spatial extensions of RDF and SPARQL, called stRDF, RDF^i and stSPARQL, in the system Strabon. This follows deliverable D4.1., in which we documented the first implementation phase of the system Strabon.

First, we present the valid time dimension of the data model stRDF and the query language stSPARQL and their implementation in Strabon. The valid time functionality, together with the geospatial features of Strabon, make Strabon the first spatio-temporal RDF store with many features and excellent performance as shown by our experimental evaluation.

Second, we investigate the problem of extending existing relational database systems with reasoning capabilities for incomplete spatial information. In this context, we implement a well-known algorithm for checking the consistency of RCC-8 constraint networks in PostgreSQL and MonetDB and contrast it with the state of the art qualitative spatial reasoners. We have chosen these two systems since these are currently the backends supported by Strabon. Our ultimate goal here is to prepare the technological ground for developing scalable query processing algorithms for the RDF^i framework on top of our geospatial RDF store Strabon. The same tehcnologies can be used to enable query processing for the topology vocabulary extension of GeoSPARQL in Strabon.

Document Information

Contract Number	FP7-257662	**Acronym**	TELEIOS
Full title	TELEIOS: Virtual Observatory Infrastructure for Earth Observation Data		
Project URL	`http://www.earthobservatory.eu/`		
EU Project Officer	Francesco Barbato		

Deliverable	**Number**	D4.2	**Name**	An implementation of a temporal and spatial extension of RDF and SPARQL on top of MonetDB - Phase II			
Task	**Number**	T4.2	**Name**	Query processing and optimization for a temporal and spatial extension of RDF and SPARQL on top of MonetDB - phase II: indefinite geospatial information			
Work package	**Number**			WP4			
Date of delivery	**Contract**	M30 (Feb. 2013)		**Actual**	23 February 2013		
Status	Draft ☐ Final ☑						
Nature	Prototype ☑ Report ☐						
Distribution Type	Public ☑ Restricted ☐						
Responsible Partner	NKUA						
QA Partner	CWI						
Contact Person	Manolis Koubarakis						
	Email	koubarak@di.uoa.gr		**Phone**	+30 210 727 5213	**Fax**	+30 210 727 5214

Project Information

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number FP7-257662. The Beneficiaries in this project are:

Partner	Acronym	Contact
National and Kapodistrian University of Athens Department of Informatics and Telecommunications	NKUA	Prof. Manolis Koubarakis National and Kapodistrian University of Athens Department of Informatics and Telecommunications Panepistimiopolis, Ilissia, GR-15784 Athens, Greece Email: (koubarak@di.uoa.gr) Tel: +30 210 7275213, Fax: +30 210 7275214
Fraunhofer Institute for Computer Graphic Research	Fraunhofer	Dr. Eva Klien Fraunhofer Institute for Computer Graphic Research Fraunhofer Strasse 5, D-64283 Darmstadt, Germany Email: (eva.klien@igd.fraunhofer.de) Tel: +49 6151 155 412, Fax: +49 6151 155 444
German Aerospace Center The Remote Sensing Technology Institute Photogrammetry and Image Analysis Department Image Analysis Team	DLR	Prof. Mihai Datcu German Aerospace Center The Remote Sensing Technology Institute Oberpfaffenhofen, D-82234 Wessling, Germany Email: (mihai.datcu@dlr.de) Tel: +49 8153 28 1388, Fax: +49 8153 28 1444
Stichting Centrum voor Wiskunde en Informatica Database Architecture Group	CWI	Prof. Martin Kersten Stichting Centrum voor Wiskunde en Informatica P.O. Box 94097, NL-1090 GB Amsterdam, Netherlands Email: (martin.kersten@cwi.nl) Tel: +31 20 5924066, Fax: +31 20 5924199
National Observatory of Athens Institute for Space Applications and Remote Sensing	NOA	Dr. Charis Kontoes National Observatory of Athens Institute for Space Applications and Remote Sensing Vas. Pavlou and I. Metaxa, GR 152 36 Athens, Greece Email: (kontoes@space.noa.gr) Tel: +30 210 8109186, Fax: +30 210 6138343
Advanced Computer Systems A.C.S S.p.A	ACS	Mr. Ugo Di Giammatteo Advanced Computer Systems A.C.S S.p.A Via Della Bufalotta 378, RM-00139 Rome, Italy Email: (udig@acsys.it) Tel: +39 06 87090944, Fax: +39 06 87201502

Contents

List of Figures

List of Tables

1. Introduction

In this deliverable we present the second phase of the implementation of our temporal and spatial extensions of RDF and SPARQL, called stRDF, RDF^i and stSPARQL, in the system Strabon. This follows deliverable D4.1., in which we documented the first implementation phase of the system Strabon and the additional features that were implemented to satisfy the requirements and demands of two TELEIOS use cases: the Fire monitoring application [KPMm10, KPM$^+$12] and the virtual observatory for TerraSAR-X [SDMD10, DDK$^+$12].

The deliverable is structured in two main parts. In the first part we present the valid time dimension of the data model stRDF and the query language stSPARQL and their implementation in Strabon [KKK12b]. Following the recent interest of the semantic web researchers to incorporate temporal features in RDF and SPARQL, we present Strabon as the first spatio-temporal RDF store with with many features and excellent performance as shown by our experimental evaluation. In the second part of the deliverable we investigate the problem of extending existing relational database systems with reasoning capabilities for incomplete spatial information. In this context, we implement a well-known algorithm for checking the consistency of RCC-8 constraint networks in PostgreSQL and MonetDB and contrast it with the state of the art qualitative spatial reasoners. Our ultimate goal here is to prepare the technological ground for developing scalable query processing algorithms for the RDF^i framework on top of our geospatial RDF store Strabon. The same tehcnologies can be used to enable query processing for the topology vocabulary extension of GeoSPARQL in Strabon.

The contributions of this deliverable are the following:

- We present the valid time dimension of the data model stRDF and the query language stSPARQL. In this way, stSPARQL becomes the most expressive query language proposed for linked spatiotemporal data, going beyond the recent OGC standard GeoSPARQL which has no support for valid time of triples.
- We present our implementation of this valid time component in Strabon and a preliminary experimental evaluation comparing it with other systems offering similar functionalities.
- We motivate the need for the development of query processing techniques for incomplete spatial information in RDF using real-world example datasets and queries. We show that some of the answers to the queries cannot be computed by current geospatial RDF stores and point out the need for using constraint propagation algorithms developed in the area of qualitative spatial reasoning.
- We concentrate on the RCC-8 calculus for reasoning about topological relations among regions and implement the path consistency algorithm of [RN01] in two relational database systems, PostgreSQL and MonetDB. We consider different implementations (one using an SQL program, one using the MonetDB column-based abstractions, and one that exposes the implementation of [RN01] as a user-defined function) and evaluate them experimentally comparing them with the reference implementation of the path consistency algorithm for RCC-8 as presented in [RN01].
- We make the case for a new generation of RCC-8 reasoners implemented in Python, a general-purpose, interpreted high-level programming language which enables rapid application development, and making use of advanced Python environments, such as PyPy[1], utilizing trace-based just-in-time (JIT) compilation techniques [BCFR09, BCF$^+$11]. We present such a reasoner, called PyRCC8, and compare it to other well-known reasoners from the literature [RN01, GWW08, SS09a].

[1] http://pypy.org/

- We go beyond PyRCC8 to develop a second reasoner called PyRCC8$\bigtriangledown$. Given a network with only tractable RCC-8 relations, PyRCC8$\bigtriangledown$ can solve it very efficiently by making its underlying constraint graph chordal and running path consistency on this sparse graph instead of the completion of the given network. In the same way, it uses partial path consistency as a consistency checking step in backtracking algorithms for networks with arbitrary RCC-8 relations resulting in very improved pruning for sparse networks while incurring a penalty for dense networks.

Moreover, we provide detailed installation instructions for each one of the implementations that have been developed in the context of WP4 and are described in this deliverable.

The structure of this deliverable is the following. Chapter 2 presents the valid time component of the data model stRDF, the query language stSPARQL, their implementation in Strabon and the first evaluation results. Next, Chapter 3 motivates the need for new query processing techniques for RDF with incomplete spatial information using algorithms from the area of constraint satisfaction. It presents the state of the art reasoners of this area targeted to the RCC-8 calculus and then, inspired by these reasoners, discusses and contrasts the implementations of a path consistency algorithm inside a database system with the state of the art reasoners. Then, Chapter 4 concludes this deliverable and outlines our plan for future work. Last, Appendix A contains instructions for installing and using the temporal component of Strabon, while Appendix B contains instructions for downloading, installing, and running the implementations of Chapter 3. Appendix C and D contain two research papers related to the work presented in Chapter 3 of the deliverable. In particular, Appendix C considers efficient algorithms for checking the consistency of chordal RCC-8 networks. Appendix D considers the problem of representing and querying incomplete information in RDF in a general way.

2. Representing and querying the valid time of triples in Strabon

In this chapter, stRDF and stSPARQL are revisited in order to study alternative representations for the valid time of a triple. All relevant approaches concerning the conceptual level, the logical level, and the implementation level have been investigated, leading to our final design and implementation decisions which are described in the following sections.

The introduction of time in data models and query languages has been the subject of extensive research in the field of relational databases [Sno95, DDL02]. Three distinct kinds of time were introduced and studied: *user-defined* time which has no special semantics (e.g., January 1st, 1963 when John has his birthday), *valid* time which is the time a fact is true in the application domain (e.g., the time 2000-2012 when John is a professor) and *transaction* time which is the time when a fact is current in the database (e.g., the system time that gives the exact period when the tuple representing that John is a professor from 2000 to 2012 is current in the database). In these research efforts, many temporal extensions to SQL92 were proposed, leading to the query language TSQL2, the most influential query language for temporal relational databases proposed at that time. However, although the research output of the area of temporal relational databases has been impressive, TSQL2 did not make it into the SQL standard and the commercial adoption of temporal database research had been very slow. It is only recently that commercial relational database systems started offering SQL extensions for temporal data, such as IBM DB2, Oracle Workspace manager, and Terradata. Also, in December 2011, the latest standard of SQL (SQL:2011) was published. Its most important new feature is the support for temporal tables, i.e., tables with rows associated with one or two temporal periods (representing valid time and transaction time respectively but without following the approach of TSQL2).

Compared to the relational database case, little research has been done to extend the RDF data model and the query language SPARQL with temporal features. Gutierrez et al. [GHV05] were the first to propose a formal extension of the RDF data model with valid time support. Perry [Per08] proposed an extension of SPARQL, called SPARQL-ST, that allows one to query spatial and temporal data.

In the same direction, Lopes et al. integrated valid time support in the general framework that they have proposed in [LPSZ10] for adding annotations to RDF triples. Finally, Tappolet and Bernstein [TB09] have proposed the language τ-SPARQL for querying the valid time of triples, showed how to transform τ-SPARQL into standard SPARQL (using named graphs), and briefly discussed an index that can be used for query evaluation.

Following the ideas of Perry [Per08], our group proposed a formal extension of RDF, called stRDF, and the corresponding query language stSPARQL for the representation and querying of temporal and spatial data using linear constraints [KK10a]. stRDF and stSPARQL were later redefined in [KKK12b] so that geometries are represented using the Open Geospatial consortium standards Well-Known-Text (WKT) and Geography Markup Language (GML). Both papers [KK10a] and [KKK12b] mention briefly the temporal dimension of stRDF, but none of the papers goes into details. Similarly, the version of the system Strabon presented in [KKK12b], which implements stRDF and stSPARQL, has not so far implemented the temporal dimension of this data model and query language. In this paper we remedy this situation by introducing all the details of the temporal dimension of stRDF and stSPARQL and implementing it in Strabon.

The contributions of this work are the following:

- We fully present the valid time dimension of the data model stRDF and the query language stSPARQL.

- We discuss our implementation of this valid time component in Strabon, which in this way becomes a spatiotemporal RDF store.
- We evaluate the performance of our implementation on two large real-world datasets and compare it to that of a Prolog implementation of the query language AnQL[1] and a naive implementation based on the native store of Sesame which we extended with valid time support. Our results show that Strabon outperforms the other implementations.

This chapter is structured as follows. In Section 2.1 we introduce the temporal dimension of the data model stRDF and in Section 2.2 we present the temporal features of the query language stSPARQL. In Section 2.3 we describe how we extended the current implementation of stSPARQL, i.e., the system Strabon, with valid time support. In Section 2.4 we evaluate our implementation experimentally and compare it with other related implementations. In Section 2.5 we present related work in this field.

2.1 Valid time representation in the data model stRDF

In this section we describe the valid time dimension of the data model stRDF presented in [KKK12b]. The *time line* assumed is the value space of the datatype `xsd:dateTime` of XML-Schema. Two kinds of time primitives are supported: time instants and time periods. A *time instant* is an element of the time line. A *time period* (or simply period) is an expression of the form $[B,E)$, $(B,E]$, (B,E), or $[B,E]$ where B and E are time instants called the *beginning* and the *ending* of the period respectively. Syntactically, time periods are represented by literals of the new datatype `strdf:period` that we introduce in stRDF. The value space of `strdf:period` is the set of all time periods covered by the above definition. The lexical space of `strdf:period` is trivially defined from the lexical space of `xsd:dateTime` and the closed/open period notation introduced above.

Figure 2.1 shows all the new datatypes defined by the model stRDF in [KKK12b] and this paper. The prefix `strdf` stands for `http://strdf.di.uoa.gr/ontology` where one can find all the relevant datatype definitions underlying the model stRDF.

We also assume the existence of temporal constants `NOW` and `UC` inspired from the literature of temporal databases [CDI+97]. `NOW` represents the current time and can be used in stSPARQL queries to be introduced in Section 2.2. `NOW` can appear in the beginning or the ending point of a period. The temporal constant `UC` means "Until Changed" and denotes that the validity time of a period has not ended yet and we do not know when it ends.

Values of the datatype `strdf:period` can be used as objects of a triple to represent *user-defined time*. In addition, they can be used to represent *valid times* of temporal triples which are defined as follows. A *temporal triple* is an expression of the form `s p o t.` where `s p o.` is an RDF triple and `t` is a time instant or a time period called the *valid time* of a triple. An *stRDF graph* is a set of triples or temporal triples. In other words, some triples in an stRDF graph might not be associated with a valid time.

Example 1. *The following stRDF graph consists of temporal triples that represent the land cover use of an area in Spain for the time periods [2000, 2006) and [2006, UC) and triples which encode other information about this area, such as its code and the WKT serialization of its geometry extent. In this and following examples, namespaces are omitted for brevity.*

```
corine:Area_4 rdf:type corine:Area .
corine:Area_4 corine:hasID "EU-101324" .
corine:Area_4 corine:hasLandUse corine:naturalGrassland
```

[1] `http://anql.deri.org/`

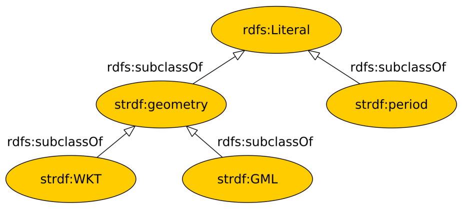


Figure 2.1: The datatypes of stRDF

```
               "[2006-01-01T00:00:00, UC)"^^strdf:period.
corine:Area_4 corine:hasLandUse corine:coniferousForest .
               "[2000-01-01T00:00:00, 2006-01-01T00:00:00)"^^strdf:period.
corine:Area_4 corine:hasGeometry
"POLYGON ((-0.662 42.345, -0.661 42.344, ...,-0.663 42.332))"^^strdf:WKT.
```

The stRDF graph provided above in N-QUADS format[2] has been extracted from a publicly available dataset provided by the European Environmental Agency (EEA) that contains the changes in the Corine Land Cover dataset for the time period [2000, 2006] for various European areas. So, according to this dataset the area `corine:Area_4` has been a coniferous forest area before 2006, when the newer version of Corine showed it to be natural grassland. Since the Corine Land cover dataset has not been updated since 2006, `UC` is used to denote the persistence of land use values into the future. The last triple of the stRDF graph provides a simplified representation of the WKT serialization of the area. This dataset will be used in our examples but also in the experimental evaluation of Section 2.4.

2.2 Querying valid times using the query language stSPARQL

The spatial features of the query language stSPARQL have been presented in [KKK12b] and [KKK$^+$12a]. In this section we will introduce for the first time all the details of the valid time dimension of stSPARQL. In order to provide valid time support, the query language stSPARQL extends SPARQL 1.1 as follows:

- Temporal triple patterns are introduced for querying temporal triples.
- Temporal extension functions are defined in order to express temporal relations between temporal data values (i.e., periods and instants).
- The temporal constants `NOW` and `UC` can be used in queries.

We now explain these temporal features of stSPARQL in more detail giving examples that illustrate the usefulness of the language.

Temporal triple patterns. A *temporal triple pattern* is an expression of the form `s p o t`, where `s p o` is a triple pattern and `t` is a time period or a variable.

Temporal extension functions. stSPARQL extends SPARQL 1.1 with 10 functions that express temporal relations between periods like in Allen's Interval Algebra. These functions are

[2]`http://sw.deri.org/2008/07/n-quads/`

the following: `strdf:before`, `strdf:after`, `strdf:during`, `strdf:starts`, `strdf:finishes`, `strdf:contains`, `strdf:meets`, `strdf:overlaps`, `strdf:isMetBy` and `strdf:cotemporal`.

Apart from the functions that express temporal relations between periods, stSPARQL offers a set of functions that construct new periods from existing ones. These functions are the following:

- `strdf:period strdf:period_intersect(period p1, period p2)`: This function returns the temporal intersection of period `p1` with period `p2`.
- `strdf:period strdf:period_union(period p1, period p2)`: This function returns the temporal union of period `p1` with period `p2`.
- `strdf:period strdf:minus(period p1, period p2)`: This function returns the temporal difference of period `p1` temporally minus period `p2`.

Also, we define the following operations that return the starting and ending points of a period:

- `xsd:dateTime strdf:period_start(period p)`: This function returns the starting point of the period `p`.
- `xsd:dateTime strdf:period_end(period p)`: This function returns the ending point of a period `p`.

Finally, stSPARQL defines the following functions that compute temporal aggregates:

- `strdf:period strdf:intersectAll(set of period p)`: Returns a period that is the intersection of the set of input periods.
- `strdf:period strdf:maxDuration(set of period p)`: Returns the period with the maximum duration of the set of periods given as input.
- `strdf:period strdf:maximalPeriod(set of period p)`: Constructs a period that begins with the minimum beginning timestamp and the maximum ending timestamp of the set of periods given as input.

The query language stSPARQL, being an extension of SPARQL 1.1, supports the temporal extension functions defined above in the SELECT, FILTER and HAVING clause of a query. A complete reference of the temporal extension functions of stSPARQL is available on the web[3].

In the rest of this section, we give some representative examples that demonstrate the expressive power of stSPARQL.

Example 2. Temporal selection and temporal constants. *Return how the landscape of each area mentioned in the dataset has evolved until now.*

```
SELECT ?clc1990 ?clc2000 ?clc2006
WHERE {?clcArea rdf:type corine:Area;
                corine:hasID ?id;
                corine:hasLandUse ?clc1990
               "[1990-01-01T00:00:00,2000-01-01T00:00:00)"^^strdf:period;
                corine:hasLandUse ?clc2000
               "[2000-01-01T00:00:00,2006-01-01T00:00:00)"^^strdf:period;
                corine:hasLandUse ?clc2006
               "[2006-01-01T00:00:00,NOW)"^^strdf:period.}
```

[3] `http://www.strabon.di.uoa.gr/stSPARQL#temporals`

This query is a temporal selection query that uses an extended Turtle syntax that we have devised in order to support temporal patterns. In this extended syntax, the fourth element is optional and it represents the valid time of the triple pattern. Constants of the `strdf:period` datatype are used as the fourth element in the temporal triple patterns in order to retrieve the land cover of a Corine area for the requested periods. The temporal constant `NOW` is also used to retrieve the current land cover of the area.

Example 3. Temporal join and spatial metric function. *Compute the area occupied by coniferous forests that were burnt.*

```
SELECT ?clc (SUM(strdf:area(?geo)) AS ?totalArea)
WHERE {?clc rdf:type corine:Area;
            corine:hasLandUse corine:coniferousForest ?t1;
            corine:hasLandUse corine:burntArea ?t2;
            clc:hasGeometry ?geo .
       FILTER(strdf:before(?t1,?t2))} GROUP BY ?clc
```

In this query, a temporal join is performed by using the temporal extension function `strdf:before` to ensure that areas included in the result set were covered by coniferous forests *before* they were burnt. The query also uses the spatial metric function `strdf:area` in the SELECT clause of the query that computes the area of a geometry. The aggregate function `SUM` of SPARQL 1.1 is used to compute the total area occupied by burnt coniferous forests.

Example 4. Temporal join and spatial selection. *Return the evolution of the landscape of the area overlapping with the following polygon:*

```
"POLYGON ((-0.66 42.34,...,-0.662 42.33))"^^strdf:WKT .
```

```
SELECT ?clc1 ?t1 ?clc2 ?t2
WHERE {?clc rdf:type corine:Area;
             corine:hasLandUse ?clc1 ?t1;
             corine:hasLandUse ?clc2 ?t2;
             clc:hasGeometry ?geo .
       FILTER(strdf:contains(?geo,
              "POLYGON(-0.66 42.34, ...,-0.662 42.33)"^^strdf:WKT))
       FILTER(strdf:before(?t1,?t2))}
```

The query described above performs a temporal join and a spatial selection. The spatial selection checks whether the geometry representation of the area is contained by a corine land cover area, so that they share the same landscape. The temporal join is performed to express the temporal evolution of this landscape.

Example 5. Update statement with temporal joins and period constructor. *Coalescing of temporal triples.*

```
UPDATE {?area corine:hasLandUse ?clc ?coalesced}
WHERE {SELECT (?clcArea AS ?area) ?clc
              (strdf:period_union(?t1,?t2) AS ?coalesced)
       WHERE {?clcArea rdf:type corine:Area;
                       corine:hasLandUse ?clc ?t1;
                       corine:hasLandUse ?clc ?t2 .
              FILTER(strdf:meets(?t1,?t2) || strdf:overlaps(?t1,?t2))}}
```

In this update, we perform an operation called *coalescing* in the literature of temporal relational databases: two temporal triples with exactly the same subject, predicate and object, and periods that overlap or meet each other can be “joined" into a single triple with valid time the union of the periods of the original triples [BSS96].

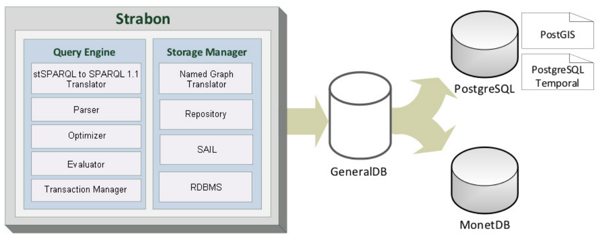


Figure 2.2: Architecture of the system Strabon enhanced with valid time support

2.3 Implementation of valid time support in Strabon

Figure 2.2 shows the architecture of the system Strabon presented in [KKK12b], as it has been extended for valid time support. We have added new components and extended existing ones as we explain below.

As described in [KKK12b], Strabon has been implemented by extending Sesame[4] 2.6.3 and using an RDBMS as a backend. Currently, PostgreSQL and MonetDB can be used as backends. To offer support for the valid time dimension of stSPARQL discussed in this document, a temporally enabled is used as back-end and the following new components have been added to Strabon:

- Named Graph Translator: This component is added to the storage manager and translates the temporal triples of stRDF to standard RDF triples following the named graphs approach of [TB09] as we discuss below.
- stSPARQL to SPARQL 1.1 Translator: This component is added to the query engine so that temporal triple patterns are translated to triple patterns as we discuss below.
- PostgreSQL Temporal: This is a temporal extension of PostgreSQL which defines a `PERIOD` datatype and implements a set of temporal functions. This datatype and its associated functions come very handy for the implementation of the valid time suport in Strabon as we will see below. PostgreSQL Temporal also allows the use of a GiST index on `PERIOD` columns. Using this add-on, PostgreSQL becomes "temporally enabled" as it adds support for storing and querying `PERIOD` objects and for evaluating temporal functions. For functions that were not directly supported by PostgreSQL Temporal, we extended the temporally enabled database with the implementation of these functions as extension functions of PostgreSQL Temporal.

Storing temporal triples. When a user wants to store stRDF data in Strabon, she makes them available in the form of a N-QUADS document. This document is decomposed into temporal triples and each temporal triple is processed separately by the storage manager as follows:

- The temporal triple is translated into the named graph representation as follows: a URI is created and is assigned to a named graph that corrresponds to the validity period of the triple. To ensure that every distinct valid time of a temporal triple corresponds to exactly one named graph, the URI of the graph is constructed using the literal representation of the valid time annotation. The triple is stored in the named graph identified by this URI. Then, the URI of the named graph is associated to its corresponding valid time by storing the following triple in the default graph: (`g, strdf:hasValidTime, t`)) where `g` is the URI of the graph and `t` is the corresponding valid time.

  For example, temporal triple

[4]`http://www.openrdf.org/`

```
corine:Area_4 corine:hasLandUse corine:naturalGrassland
             "[2000-01-01, 2006-01-01)"^^strdf:period
```

will be translated into the following standard RDF triples:

– the triple `corine:Area_4 corine:hasLandUse corine:naturalGrassland` will be stored in the named graph `corine:2000-01-01_2006-01-01`

– the triple

```
corine:2000-01-01_2006-01-01 strdf:hasValidTime
              "[2000-01-01, 2006-01-01)"^^strdf:period
```

will be stored in the default graph.

- For the temporal literals found during data loading, we deviate from the default behaviour of Sesame by storing the instances of the `strdf:period` datatype in a table with schema *period_values(id int, value period)*. The attribute *id* is used to assign a unique identifier to each period and associate it to its RDF representation as a typed literal. The attribute *value* is a temporal column of the `PERIOD` datatype defined in PostgreSQL Temporal. In addition, we construct an R-tree-over-GiST index on the *value* column.

Querying temporal triples. Let us now explain how the query engine of Strabon presented in [KKK12b] has been extended to evaluate temporal triple patterns. When a temporal triple pattern is encountered, the query engine of Strabon executes the following steps:

- The stSPARQL to SPARQL 1.1 Translator converts the temporal triple patterns to a set of triple patterns as follows. A temporal triple pattern `s p o t` is translated into the following graph pattern:

  ```
  GRAPH ?g { s p o } .
  ?g strdf:hasValidTime t .
  ```

  where `s`, `p`, `o` are RDF terms or variables and `t` is either a variable or an instance of the datatypes `strdf:period` or `xsd:dateTime`.

- If a temporal extension function is present, the evaluator incorporates the table *period_values* to the query tree and it is declared that the arguments of the temporal function will be retrieved from the *period_values* table. In this way, all temporal extension functions are evaluated using PostgresSQL Temporal.

- Finally, the RDBMS evaluation module has been extended so that the execution plan directed by the logical level of Strabon is translated into suitable SQL statements. The temporal extension functions are respectively mapped into SQL statements using the functions and operators provided by PostgreSQL Temporal.

2.4 Evaluation

For the experimental evaluation of our system, we used two different datasets:

- the GovTrack dataset[5], which consists of RDF data about US Congress. This dataset was created by Civic Impulse, LLC[6] and contains information about politicians, bills and voting records.

[5]`http://www.govtrack.us/data/rdf/`

[6]`http://www.civicimpulse.com/`

- the Corine Land Cover changes dataset that represents changes for the period (2000, 2006), which we have already introduced in Section 2.1.

Both datasets had to be transformed into N-QUADS before they could be used in our evaluation.

The GovTrack dataset contain temporal information in the form of instants and periods, but in standard RDF format using reification. So, in the pre-processing step we transformed the dataset into N-QUADS format. For example the 5 triples

```
congress_people:A000069 politico:hasRole _:node17d3oolkdx1 .
_:node17d3oolkdx1 time:from _:node17d3oolkdx2 .
_:node17d3oolkdx1 time:to _:node17d3oolkdx3 .
_:node17d3oolkdx2 time:at "2001-01-03"^^xs:date .
_:node17d3oolkdx3 time:at "2006-12-08"^^xs:date .
```

were transformed into a single quad:

```
congress_people:A000069 politico:hasRole _:node17d3oolkdx1
                               "[2001-01-03, 2006-12-08]"^^strdf:period.
```

The transformed dataset has a total number of 7,900,905 triples, 42,049 of which have periods as valid time and 294,636 have instants.

The Corine Land Cover changes dataset for the time period [2000, 2006) is publicly available in the form of shapefiles and it contains the areas that have changed their landscape between the years 2000 and 2006. Using this dataset, we created a new dataset in N-QUADS form which has information about geographic regions such as: unique identifiers, geometries and periods when regions have a landcover. The dataset contains 717,934 temporal triples whose valid time is represented using the `strdf:period` datatype. It also contains 1,076,901 triples without valid times. Using this dataset, we were able to perform temporal and spatial stSPARQL queries, similar to the ones that we provided in 2.2 as examples.

Our experiments were conducted on an Intel Xeon E5620 with 12MB L3 caches running at 2.4 GHz. The system has 24GB of RAM and 4 disks of striped RAID (level 5) and the operating system installed is Ubuntu 12.04. We ran our queries three times on cold and warm caches, for which we ran each query once before measuring the response time.

We evaluate the performance of our system in terms of query response time. We compute the response time for each query posed by measuring the elapsed time from query submission till a complete iteration over the results had been completed. We also investigate the scalability of our system with respect to database size and complexity of queries. Finally, we compare our results with a Prolog implementation of AnQL[7] and a naive implementation based on the Sesame native store which we extended by adding valid time support and temporal extension functions implemented in Java. A similar implementation has been used as a baseline in [KKK12b] where we evaluated the geospatial features of Strabon.

We have conducted 3 different experiments and we compared the performance of Strabon to the naive implementation in all three experiments. The Prolog implementation of the temporal domain of AnQL only implements the temporal functions `beforeAll` and `beforeAny` and does not provide geospatial support, it took part only in the first two experiments. Twenty queries were executed in total. Only one query is shown here; the rest are omitted due to space considerations. However, all datasets and the queries that we used in our experimental evaluation are publicly available[8].

[7] http://anql.deri.org/
[8] http://www.strabon.di.uoa.gr/temporal-evaluation/experiments.html

Table 2.1: Composition of the GovTrack datasets in valid time representations

Total triples and quads	periods	instants
1,000	300	300
10,000	3000	3000
100,000	30000	30000
1,000,000	42,049	294,636
8,000,000	42,049	294,636

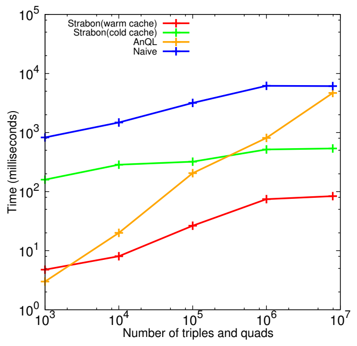


Figure 2.3: Query response time with respect dataset size

Experiment 1. In this experiment we ran the same query against a number of subsets of the GovTrack dataset of various size, as we wanted to test the scalability of our implementation with respect to the dataset size. To achieve this, we created 5 instances of the GovTrack dataset, each one with exponential increasing number of triples and quads. The query that is evaluated against these datasets is the following:

```
#Q1: Return the ids and optionally the names of members
of Congress after the period 2000-2004.

SELECT distinct ?x ?name
WHERE {
   ?x gov:hasRole> ?term ?t .
   OPTIONAL { ?x  foaf:name ?name . }
   FILTER(strdf:after(?t,"[2000-01-01T00:00:00, 2004-12-31T23:59:59]"^^strdf:period))
   }
```

Figure 2.3 shows the results of this experiment. As the dataset size increases, more periods need to be processed and as expected, the query response time grows. The respective results when the caches are warm are far better, as the intermediate results are available in main memory, so we have less I/O requests. One can observe that Strabon achieves better scalability than the other two systems, although the Prolog implementation of AnQL performs better for small datasets of up to 100.000 triples. The poor performance of the naive implementation compared to Strabon is reasonable, as Strabon evaluates the temporal extension functions in the RDBMS level using the respective functions of PostgreSQL Temporal and a GiST index on period values, while in the case of the naive implementation a full scan over all literals is required.

Experiment 2. We carried out this experiment to measure the scalability of our implementation with respect to queries of varying complexity. The complexity of a query depends on the number and the type of the graph patterns it contains and their selectivity. We posed a set of queries against our biggest dataset (GovTrack), that contains approximately 8 millions of temporal and non-temporal triples and we increased the number of triple patters in each query.

Table 2.2: System performance in experiment 2

System	Q1	Q2	Q3	Q4	Q5	Q6	Q7
Strabon (cold caches)	448	242	710	1156	1537	480	432
Strabon (warm caches)	196	47	334	720	1113	30	29
AnQL	2028	1715	4275	4379	5802	6913	7472
Naive	945	2104	2197	2241	2271	400	397

Table 2.3: System performance in experiment 3

System	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
Strabon (cold caches)	485	381	377	370	250	248	248	285	376	268	368	532	792
Strabon (warm caches)	217	212	209	207	82	84	81	208	200	3	13	12	208
Naive	6206	6148	6162	6196	6097	6248	6388	6332	6258	404	2868	2388	200705

Table 2.2 shows the results of this experiment. Strabon scales better than the naive and the AnQL implementation for the same reasons as in Figure 2.3. First, in Q2, we have a temporal triple pattern and a temporal selection on its valid time. Then, Q3 is formed by adding a temporal join to Q2. Then Q4 and Q5 are formed by adding some more graph patterns to Q3. The extra graph patterns we add are characterized by low selectivity. This means that they match large graphs of the dataset, so the reponse time increases because in most cases the intermediate results do not fit in the main memory blocks that are available, requiring more I/O requests, and because of the fact that the additional graph patterns do not decrease the number of the intermediate results. In the queries Q6 and Q7 we added graph patterns with high selectivity and the response time descreased. This happened because highly selective graph patterns eliminate many results from the result set and produce less intermediate results to be processed in the query processing workflow. The respective response times in warm caches are far better, as expected. What is interesting in this case, is that while in cold caches the reponse time slightly increases from the query Q6 to the query Q7, in warm caches it decreases. This happens because with warm caches, the computational effort is widely reduced and the response time is more dependent of the number of the intermediate results which are produced. The query Q7 does produce less intermediate results, because the graph pattern inserted is more selective.

Experiment 3. This experiment was carried out in two parts and its results are shown in Table 2.3:

- testing the temporal operators: We posed temporal queries against the 8 million instance of the GovTrack dataset in order to test some temporal operators in the FILTER clause of the query. The queries posed (Q8-Q16) are identical and they differ only on the temporal function used in the FILTER statement of the query.
- Using the Corine Land Cover changes 2000-2006 dataset: Four queries (Q17-Q20) were posed against this dataset. It contains more temporal triples, but there are only two different period values as the valid time representations of triples. Query response time is increased in the query Q20, as it performs a temporal join and a temporal selection.

It is clear from Table 2.3 that Strabon with warm caches is again the winner in both parts of this experiment.

2.5 Related Work

The `strdf:period` datatype of stRDF is similar to the period datatypes used in the few commercial relational database systems that have good support for time e.g., the Teradata database[9], DB2[10] and Oracle Workspace Manager[11]. These systems implement many of the features that have been introduced in SQL:2011.

To the best of our knowledge, the only commercial RDF store that has comprehensive support for time is AllegroGraph[12]. AllegroGraph defines temporal datatypes, such as time instants and intervals and supports a set of temporal functions, including those that are based on Allen's interval calculus. The temporal component of AllegroGraph captures only user-defined time, but since the temporal datatypes and stSPARQL extension functions that we introduced in Sections 2.1 and 2.2 can be used to represent and query user-defined time, AllegroGraph and Strabon offer comparable functionalities on this dimension.

Gutierrez et al. [GHV07, GHV05] were the first to propose a formal extension of the RDF data model with valid time support. [GHV07] also introduces the concept of *anonymous timestamps* in general temporal RDF graphs, i.e., graphs containing quads of the form $(s, p, o)[t]$ where t is a timestamp or an anonymous timestamp x stating that the triple (s, p, o) is valid in some unknown time point x. [HV06] subsequently extends the concept of general temporal RDF graphs of [GHV07] so that one is allowed to express temporal constraints involving anonymous timestamps.

Perry [Per08] proposed an extension of SPARQL, called SPARQL-ST, for representing and querying spatiotemporal data. The main idea of [Per08] is to incorporate geospatial information to the temporal RDF graph model of [GHV07]. The query language SPARQL-ST adds two new types of variables, namely spatial and temporal ones, to the standard SPARQL variables. Temporal variables (denoted by a # prefix) are mapped to time intervals and can appear in the fourth position of a quad as described in [GHV07]. Furthermore, in SPARQL-ST two special filters are introduced: `SPATIAL FILTER` and `TEMPORAL FILTER`. These filters are used to filter the results with spatial and temporal constraints (OGC Simple Feature Access topological relations and distance for the spatial part, and Allen's interval calculus [All83] for the temporal part).

The temporal dimension of stRDF and stSPARQL presented in this paper is in the same spirit as [Per08], but it takes place in a language with a much more mature geospatial component based on OGC standards than the language in [Per08]. In addition, stSPARQL offers a richer set of functions for querying valid times as we have discussed in Section 3. Unfortunately, the implementation of the ideas in [Per08] could not be made available to us to compare with our own implementation.

With the temporal dimension presented in this paper, stSPARQL also becomes more expressive than the recent OGC standard GeoSPARQL [ogc12]. While now stSPARQL can represent and query geospatial data that change over time, GeoSPARQL only supports static geospatial data. An interesting contribution of GeoSPARQL is the definition of small upper level ontologies that give the users vocabulary for modeling geospatial data through well-known GIS concepts such as feature, geometry etc. It would be nice to revisit the definition of these ontologies in the light of valid time of triples as modeled in stSPARQL i.e., to define small upper level ontologies for modeling spatiotemporal data. Another contribution of GeoSPARQL which is relevant to the work presented in this paper is its Topology vocabulary extension. This extension allows one to assert topological relations between two regions as RDF triples (e.g., region A is inside region B) and query/reason with them using well-known calculi for topological reasoning such as RCC-8 [RCC92b]. In a similar spirit, one can extend stSPARQL with a similar component so that in addition to the topology vocabulary, one can offer vocabulary for binary temporal relations (e.g., relations from Allen's Interval Algebra etc.) that can be asserted and queried/reasoned with.

[9] `http://www.teradata.com/`
[10] `http://www-01.ibm.com/software/data/db2/`
[11] `http://www.oracle.com/technetwork/database/enterprise-edition/index-087067.html`
[12] `http://www.franz.com/agraph/allegrograph/`

Tappolet and Bernstein in [TB09] introduce a storage format for temporal RDF graphs using named graphs. According to this approach, queries expressed in the temporally enhanced query language τ-SPARQL are translated into standard SPARQL and are executed on the stored named graphs. [TB09] also suggests the use of an indexing structure for speeding access to temporal data. However, temporal functions like the ones we are using in stSPARQL are not supported. As we have explained in Section 2.3, we are using the same approach as [TB09] and rely on a translation to named graphs for our implementation. Unfortunately, the implementation of [TB09]could not be made available to us to compare it with our own. An efficient index structure, named tGRIN, is also proposed by [PUS08].

[Gra10] presents another approach for extending RDF with temporal features, using a temporal element that captures more than one time dimensions. [Gra10] also proposes a temporal extension of SPARQL, named T-SPARQL, which is based on TSQL2. Furthermore, [Mot12] presents a logic-based approach for extending RDF and OWL with valid time and the query language SPARQL for querying and reasoning with RDF, RDFS and OWL2 temporal graphs. To the best of our knowledge, no public implementation of [Gra10] and [Mot12] exists that we could use to test our work against.

[BP10] introduces SOWL, an extension of OWL, for representing spatial and spatio-temporal information and evolution in time. SOWL incorporates spatio-temporal reasoning in SOWL for inferring spatio-temporal information from representations in the underlying ontology model.

3. Database Techniques for Incomplete Spatial Information

In this chapter, we investigate the possibilities of using existing relational database systems to implement reasoning capabilities for incomplete spatial information. Our ultimate goal is to prepare the technological ground for developing scalable query processing algorithms for the framework RDF^i originally defined in deliverable D2.3 [KNF11]. The most recent version of RDF^i is presented in Appendix D. As we will show, our investigation is also useful for solving the problem of query answering for the topology vocabulary extension of GeoSPARQL.

The organization of the chapter is as follows. Section 3.1 motivates the need for our query processing techniques using several real world datasets and queries, some of the answers of which cannot be computed by current geospatial RDF stores. It turns out that appropriate query processing techniques should employ such algorithms as the ones that have been developed in the area of constraint satisfaction and in particular in qualitative spatial reasoning. Therefore, Sections 3.2 and 3.3 contain background information and terminology useful in these areas. Then, Section 3.4 presents the state of the art qualitative spatial reasoners focusing on the RCC-8 calculus. Next, in Section 3.5 we present our implementation of the path consistency algorithm for the RCC-8 calculus as presented in [RN01] both in PostgreSQL and MonetDB database systems. Last, Section 3.6 performs an experimental evaluation of our implementation and compares them with the state of the art RCC-8 reasoners.

3.1 Querying Incomplete Spatial Information

In this section we motivate the need for the development of appropriate algorithms for querying incomplete spatial information. We start from a simple RDF database and a SPARQL query that deal with definite geospatial information and then embellish them by adding qualitative spatial information. Using real examples and queries that are similar to the ones that our partner, the National Observatory of Athens, employs in its use case, we shed light on the challenges that an implementation of RDF^i or the topology vocabulary extension of GeoSPARQL has to address and point to technological solutions that have to be followed.

Let us start with an example containing definite quantitative geospatial information. We will follow the syntax of GeoSPARQL in all examples of this section.

Example 6. Triples representing the town of Olympia and a burnt area product produced by the processing chain of the National Observatory of Athens.

```
noa:Town rdfs:subClassOf geo:Feature.

geonames:264637 a noa:Town;
                geonames:name "Olympia";
                owl:sameAs dbpedia:Olympia_Greece;
                geo:hasGeometry ex:OlympiaPolygon.

ex:OlympiaPolygon a sf:Polygon;
                  geo:asWKT "POLYGON((21.5 18.5, 23.5 18.5,
                  23.5 21, 21.5 21, 21.5 18.5))"^^geo:wktLiteral.

noa:BurntArea rdfs:subClassOf geo:Feature.
```

```
noa:ba1 a noa:BurntArea;
        geo:hasGeometry ex:ba1Polygon.

ex:ba1Polygon a sf:Polygon;
              geo:asWKT "POLYGON((20 20, 20 22, 22 22, 22 20,
              20 20))"^^geo:wktLiteral.
```

The above triples represent some information about the Greek town Olympia and a burnt area close to Olympia. This information is typical in the TELEIOS use case led by the National Observatory of Athens. The triples include an approximation of their geometries modified for the purposes of our example.

We now give an example of a GeoSPARQL query over the triples of Example 6 that can be used to answer questions concerning the geometries of some objects.

Example 7. Return the names of towns that have been affected by fires.

```
SELECT ?name
WHERE { ?town a noa:Town;
              geonames:name ?name;
              geo:hasGeometry ?townpoly.
        ?townpoly geo:asWKT ?townGeo.
        ?ba   a noa:BurntArea;
              geo:hasGeometry ?bapoly.
        ?bapoly geo:asWKT ?baGeo.
        FILTER(geof:sfIntersects(?townGeo,?baGeo))
}
```

The above query is expressed in GeoSPARQL using the geometry topology extension. This query checks whether the Boolean function `geof:sfIntersects` holds between the geometry literals related to towns and burnt areas.

Let us now see how we can augment the kind of geospatial information of Example 6 so that we have qualitative spatial information. This is illustrated below using the modeling possibilities offered by the topology vocabulary extension of GeoSPARQL.

Example 8. Triples representing administrative geography information about the community of Olympia, the municipality of Olympia, and the region of West Greece.

```
gag:Olympia rdf:type gag:Community;
            rdfs:label "Ancient Olympia".

gag:OlympiaMunicipality rdf:type gag:Municipality;
                        rdfs:label "Municipality of Ancient Olympia".

gag:WestGreece rdf:type gag:Region;
          rdfs:label "Region of West Greece".

gag:OlympiaMunicipality geo:sfContains gag:Olympia.

gag:WestGreece geo:sfContains gag:OlympiaMunicipality.
```

According to the Greek Administrative Geography[1] (namespace gag in the example), Greece is divided into 7 decentralized administrations, 13 regions and 325 municipalities. Communities, such as Ancient Olympia, are parts of municipalities (in this case the municipality with the same name). The municipality of Ancient Olympia is part of the region of West Greece. The administrative geography of Greece has been published as linked data by our group at the Greek Linked Open Data portal[2].

The last two of the above triples are used to assert the topological relations that hold between these three administrative divisions of Greece using vocabulary from the topology extension of GeoSPARQL.

Considering the triples of Example 8 above, one can then formulate various queries involving qualitative spatial information. This is illustrated in the following example.

Example 9. Find the administrative region that contains the community of Ancient Olympia.

```
SELECT ?d
WHERE {
     ?d rdf:type gag:Region.
     ?d geo:sfContains gag:Olympia.
}
```

The answer to the previous query is displayed below:

?d
gag:WestGreece

GeoSPARQL does not tell us how to compute this answer which needs reasoning about the transitivity of relation geo:sfContains. One option would be to have rules that express the transitivity of RCC-8 relations [BP10] while another would be to consult an RCC-8 reasoner based on constraint networks as in description logic reasoners PelletSpatial [WM09] and RacerPro [SS09b].

Let us now modify the information represented in Example 6 to reflect the process that NOA follows during the summer season for real-time fire monitoring.

Example 10. Triples representing the town of Olympia and a hotspot product produced by the processing chain of the National Observatory of Athens.

```
noa:Town rdfs:subClassOf geo:Feature.

geonames:264637 a noa:Town;
                geonames:name "Olympia";
                owl:sameAs dbpedia:Olympia_Greece;
                geo:hasGeometry ex:OlympiaPolygon.

ex:OlympiaPolygon a sf:Polygon;
                  geo:asWKT "POLYGON((21.5 18.5, 23.5 18.5,
                  23.5 21, 21.5 21, 21.5 18.5))"^^geo:wktLiteral.

noa:f1 a noa:Fire.
```

[1] http://en.wikipedia.org/wiki/Kallikratis_reform
[2] http://linkedopendata.gr/

```
noa:Hotspot rdfs:subClassOf geo:Feature.

noa:hotspot1 a noa:Hospot;
             geo:hasGeometry ex:h1Polygon;
             geo:sfContains noa:f1.

ex:h1Polygon a sf:Polygon;
              geo:asWKT "POLYGON((<3x3-rectangle>))"^^geo:wktLiteral.
```

The above triples represent some information about the Greek town Olympia like it has been done in Example 6. The difference is that now the exact geometry of the burnt area that is close to Olympia is not known. The only information we have is the existence of a fire the extent of which is limited by a certain geometry, that is, it is known that the geometry of the fire, although unknown, is inside the 3km by 3km rectangle of the geometry of a hotspot. This is in fact a real example given the low spatial resolution of the MSG/SEVIRI sensor utilized by NOA.

Let us now take Example 8 and add to it information about the geometry of the administrative region of West Greece. The geometries of such regions are provided by the geodata portal of the Government of Greece[3] and have been transformed into linked data by us. The triples are shown in the following example.

Example 11. Triples representing the geometry of the administrative region of West Greece.

```
gag:Region rdfs:subClassOf geo:Feature.

gag:WestGreece rdf:type gag:Region;
               geo:hasGeometry ex:wgpoly.

ex:wgpoly a sf:Polygon;
            geo:asWKT "MULTIPOLYGON (((25.9 41, 25 41,...
                      22.2 43, 22.4 43)))"^^geo:wktLiteral.
```

The triples above give the actual geometry of the administrative region of West Greece using the modeling of GeoSPARQL.

Let us know see how easy it is to answer queries over the combined information from Examples 10, 8, and 11. This is illustrated in the following example.

Example 12. Return the fires inside the administrative region of West Greece.

```
SELECT ?f
WHERE {
    ?f a noa:Fire.
    ?h a noa:Hotspot;
       geo:sfContains ?f;
       geo:hasGeometry ex:h1Polygon.
    ex:h1Polygon geo:asWKT ?hGeo.
    gag:WestGreece geo:hasGeometry ?wgpoly.
    ?wgpoly geo:asWKT ?WGgeo.
    FILTER(geo:sfWithin(?WGgeo,?hGeo))
}
```

[3] http://geodata.gov.gr/

The above query uses both the geometry and the topology vocabulary extensions of GeoSPARQL. The geometry extension is used in the `FILTER` statement (boolean function `geo:sfWithin`), while the topology vocabulary extension is used in the triple pattern containing the `geo:sfContains` property. The query searches for hotspots contained in the administrative region of West Greece (`FILTER` expression) and then selects the fires that are contained in these hotspots (triple pattern matching using the `geo:sfContains` property).

Even though the GeoSPARQL query given in Example 12 is a legitimate one, it takes into account the modeling followed in the input triples making it hard to write for larger and more complex datasets. In contrast, one would be tempted to write down a simpler version of that query as given in the example below.

Example 13. Return the fires inside the administrative region of West Greece.

```
SELECT ?f
WHERE {
    ?f a noa:Fire.
    gag:WestGreece geo:sfContains ?f.
}
```

The GeoSPARQL query above conveys in a direct way the meaning of the query we gave in natural language and it uses only the topology extension. However, the specification of GeoSPARQL does not propose any algorithm for computing the answer to such a query, although the answer is entailed by the triples of Examples 10, 8, and 11. An algorithm for computing such an entailment needs to deal with both qualitative and quantitative spatial information. The algorithm should derive the relation `geo:sfContains` between `gag:WestGreece` and `noa:hotspot1` and then include it in the computation of the transitive closure for relation `geo:sfContains`, and thus deriving the triple `gag:WestGreece geo:sfContains noa:f1`.

3.2 Preliminaries

In this section we define the concepts of constraint networks and their corresponding constraint graphs, relation algebra, composition, weak composition, algebraic closure (a-closure), and various notions of local consistency. More details can be found in [RN07].

3.2.1 Constraint Networks and Relation Algebra

Knowledge about entities or about the relationships between entities is often given in the form of *constraints*. Given a set of m variables $\mathcal{V} = \{x_1, \ldots, x_m\}$ over a domain $\mathcal{D}$, an *n-ary constraint* consists of an n-ary relation $R_i \subseteq \mathcal{D}^n$ and an n-tuple of variables $\langle x_{i_1}, \ldots, x_{i_n} \rangle$, written $R_i(x_{i_1}, \ldots, x_{i_n})$. For binary constraints, we will also use the infix notation $x_1 R_i x_2$. A (partial) *instantiation* f of variables to values is a (partial) function from the set of variables $\mathcal{V}$ to the set of values $\mathcal{D}$. We say that an instantiation f *satisfies the constraint* $R_i(x_{i_1}, \ldots, x_{i_n})$ if and only if $f(x_{i_1}), \ldots, f(x_{i_n}) \in R_i$.

A *constraint satisfaction problem* (CSP) consists of a set of variables $\mathcal{V}$ over a domain $\mathcal{D}$ and a set of constraints Θ. The intention is to find a *solution* which is an instantiation such that all constraints in Θ are satisfied. In this paper we restrict ourselves to binary CSPs, i.e., CSPs where only binary constraints are used. A binary CSP can be represented by a *constraint network* which is a labelled digraph where each node is labelled by a variable x_i or by the variable index i and each directed edge is labelled by a binary relation. We will use the notation R_{ij} to denote the relation constraining the variable pair $\langle x_i, x_j \rangle$. A *constraint graph* for such a problem is an undirected graph where the nodes represent variables and two nodes are linked to represent the *existence* of a constraint which involves these variables. At most a single edge will join two nodes, even if they are related by more than one constraint. By overloading notation, we also use R_{ij} to denote the constraint $R_{ij}(x_i, x_j)$ itself. A CSP is *consistent* if it has a solution. If the domain of the variables is finite, CSPs can be solved by *backtracking* over the ordered domains of the single variables. If the domain of the variables is infinite, as is the case with RCC-8, backtracking over the domain is not possible and other methods have to be applied. Infinite domains is the main difference of spatial or temporal CSPs to normal CSPs. For instance, there are infinitely many time points or temporal intervals on the time line and infinitely many regions in a two or three dimensional space.

One way of dealing with infinite domains is using constraints over a finite set of binary relations, by employing a *relation algebra* [LM94]. A *relation algebra* consists of a set of binary relations R which is closed under several operations on relations and contains some particular relations. The operations are union ($\cup$), intersection ($\cap$), composition ($\circ$), complement ($\bar{\cdot}$), and conversion ($\breve{\cdot}$), where conversion is defined as $\breve{R} \overset{def}{=} \{\langle x, y \rangle | \langle y, x \rangle \in R\}$ and composition is defined as $R \circ S \overset{def}{=} \{\langle x, y \rangle | \exists z : \langle x, z \rangle \in R \wedge \langle z, y \rangle \in S\}$. The particular binary relations mentioned above are the empty relation $\emptyset$ which does not contain any pair, the universal relation $*$ which contains all possible pairs, and the identity relation Id which contains all pairs of identical elements in a set of constraints. We assume that a set of constraints Θ contains one constraint for each pair of variables involved in Θ, i.e., if no information is given about the relation holding between two variables x_i and x_j, then the universal relation $*$ constrains the pair, i.e., $R_{ij} = *$. Another assumption that we make is that whenever a constraint R_{ij} between x_i and x_j is in Θ, the converse relation constrains x_j and x_i , i.e., $\breve{R_{ij}} = R_{ji}$.

Of particular interest are relation algebras that are based on finite sets of *jointly exhaustive and pairwise disjoint* (JEPD) relations. JEPD relations are sometimes called atomic, basic, or base relations. We refer to them as base relations. Since any two entities are related by exactly one of the base relations, they can be used to represent definite knowledge with respect to the given level of granularity. Indefinite knowledge can be specified by unions of possible base relations. For these relation algebras, the universal relation is the union over all base relations. Converse, complement, intersection and union of relations can easily be obtained by performing the corresponding set theoretic operations. Composition of base relations has to be computed using the semantics of the relations. Composition of unions of base relations can be obtained by computing only the union of the composition of the base relations. Usually, compositions of the base relations are pre-computed and stored in a *composition table.*

3.2.2 Composition and Weak Composition

According to the definition of composition, we have to look at an infinite number of tuples in order to compute composition of base relations, which is clearly not feasible. Fortunately, many domains such as points or intervals on a time line are ordered or otherwise well-structured domains and composition can be computed using the semantics of the relations. However, for domains such as arbitrary spatial regions that are not well structured and where there is no common representation for the entities we consider, computing the true composition is not feasible and composition has to be approximated by using *weak composition* [RL05]. Weak composition ($\diamond$) of two relations S and T for a set of base relations $\mathcal{B}$ is defined as the strongest relation $R \in 2^{\mathcal{B}}$ which contains $S \circ T$, or formally, $S \diamond T \overset{def}{=} \{R_i \in \mathcal{B} | R_i \cap (S \circ T) \neq \emptyset\}$. The advantage of weak composition is that we stay within the given set of relations $R = 2^{\mathcal{B}}$ while applying the algebraic operators, as R is by definition closed under weak composition, union, intersection, and converse.

3.2.3 Path Consistency and Algebraic Closure

Because of the high complexity of deciding consistency, different forms of local consistency and algorithms for achieving local consistency were introduced. Local consistency is used to prune the search space by eliminating local inconsistencies. In some cases local consistency is even enough for deciding consistency. The path consistency algorithm is one of the central methods to solve constraint networks in qualitative spatial and temporal calculi. Path consistency is implemented by introducing rules defining compositions and intersections of supported relations. Path consistency approximates consistency and realises *forward checking* in a backtracking algorithm, by checking the consistency of all triples of relations and eliminating relations that are impossible, through iteravely performing the following operation:

$$\forall i, j, k \; R_{ij} \leftarrow R_{ij} \cap (R_{ik} \circ R_{kj})$$

until a fixed point $\overline{R}$ is reached. If $R_{ij} = \emptyset$ for a pair (i, j) then R is inconsistent, otherwise $\overline{R}$ is *path consistent.*

When weak composition differs from composition, we cannot apply the path consistency algorithm as it requires composition and not just weak composition. We can, however, replace the composition operator in the path consistency algorithm with the weak composition operator. The resulting algorithm is called the algebraic closure algorithm [LR04] which makes a network algebraically closed or a-closed. Algebraic closure is essentially path consistency with weak composition.

3.2.4 Varieties of k-consistency

Freuder [Fre78] generalised path consistency and the weaker notion of arc consistency to k-consistency: A CSP is k-consistent, if for every subset $\mathcal{V}_k \subset \mathcal{V}$ of k variables the following holds: for every instantiation of $k-1$ variables of $\mathcal{V}_k$ that satisfies all constraints of C that involve only these $k-1$ variables, there is an instantiation of the remaining variable of $\mathcal{V}_k$ such that all constraints involving only variables of $\mathcal{V}_k$ are satisfied. So if a CSP is k-consistent, we know that each consistent instantiation of $k-1$ variables can be extended to any k-th variable. A CSP is strongly k-consistent, if it is i-consistent for every $i \leq k$. If a CSP with n variables is strongly n-consistent (also called *globally consistent*) then a solution can be constructed incrementally without backtracking. 3-consistency is equivalent to path consistency, 2-consistency is equivalent to arc consistency.

3.3 Qualitative Spatial Reasoning

Qualitative Spatial Reasoning is based on qualitative abstractions of spatial aspects of the common-sense background knowledge, on which our human perspective on the physical reality is based. Space has become a significant research area within the field of Qualitative Reasoning, and, more generally, in the Knowledge Representation and Reasoning community.

The main reasons why non-precise, qualitative spatial information may be useful are the following:

- Only partial information may be available (e.g. we may know that one region is *disconnected* from another without knowing the precise geometry of the regions)
- General constraints holding among geographical objects are often most naturally stated in qualitative terms (e.g. we may wish to state that one region is *part of* another region)

Table 3.1: Definition of the various relations of RCC. Relations in bold are included in RCC-8

Relation	Description	Definition
$C(x,y)$	connects with	primitive relation
$\mathbf{DC}(x,y)$	disconnected	$\neg C(x,y)$
$P(x,y)$	part	$\forall z[C(z,x) \rightarrow C(z,y)]$
$PP(x,y)$	proper part	$P(x,y) \wedge \neg P(y,x)$
$\mathbf{EQ}(x,y)$	equals	$P(x,y) \wedge P(y,x)$
$O(x,y)$	overlaps	$\exists z[P(z,x) \wedge P(z,y)]$
$\mathbf{PO}(x,y)$	partially overlaps	$O(x,y) \wedge \neg P(x,y) \wedge \neg P(y,x)$
$DR(x,y)$	discrete	$\neg O(x,y)$
$\mathbf{TPP}(x,y)$	tangential proper part	$PP(x,y) \wedge \exists z[EC(z,x) \wedge EC(z,y)]$
$\mathbf{EC}(x,y)$	externally connected	$C(x,y) \wedge \neg O(x,y)$
$\mathbf{NTPP}(x,y)$	non-tangential proper part	$PP(x,y) \wedge \neg\exists z[EC(z,x) \wedge EC(z,y)]$
$Pi(x,y)$	part inverse	$P(y,x)$
$PPi(x,y)$	proper part inverse	$PP(y,x)$
$\mathbf{TPPi}(x,y)$	tangential proper part inverse	$TPP(y,x)$
$\mathbf{NTPPi}(x,y)$	non-tangential proper part inverse	$NTPP(y,x)$

The challenge, then, arises to provide calculi which allow the represenation and reasoning of spatial knowledge, without resorting to the traditional quantitative techniques prevalent in, for example, the computer graphics or computer vision communities.

Qualitative Spatial Reasoning is an important subproblem in many application areas, such as geographical information systems (GIS), robotic navigation, high level vision, the semantics of spatial prepositions in natural languages, engineering design, commonsense reasoning about physical situations, and specifying visual language syntax and semantics [Coh97]. Recently, the Semantic Web community[4] has been making efforts to enhance models and query languages involved with qualitative reasoning capabilities [KK10b, Ope12].

3.3.1 The Region Connection Calculus

The Region Connection Calculus (RCC) [RCC92a] is the dominant topological approach to representing and reasoning about topological relations. RCC abstractly describes regions that are non-empty regular subsets of some topological space, by their possible relations to each other. RCC is based on a single primitive relation between spatial regions, relation C [Cla81]. The intended topological interpretation of $C(a,b)$, where a and b are spatial regions, is that a and b are connected if and only if their topological closures share a common point. This primitive can be used to define many predicates and functions which capture interesting and useful topological distinctions [Got94, Got96]. Table 3.1 shows the definitions of the RCC relations in terms of the basic relation C. Of particular importance are those relations that form a set of jointly exhaustive and pairwise disjoint relations, which are also denoted base relations. Base relations have the property that exactly one of them holds between any two spatial regions. These relations form the RCC-8 calculus and are depicted in Figure 3.1. The relations of RCC-8 are the following: disconnected (DC), externally connected (EC), equal (EQ), partially overlapping (PO), tangential proper part (TPP), tangential proper part inverse (TPPi), non-tangential proper part (NTPP), and non-tangential proper part inverse (NTPPi).

[4] `http://www.w3.org/standards/semanticweb/`

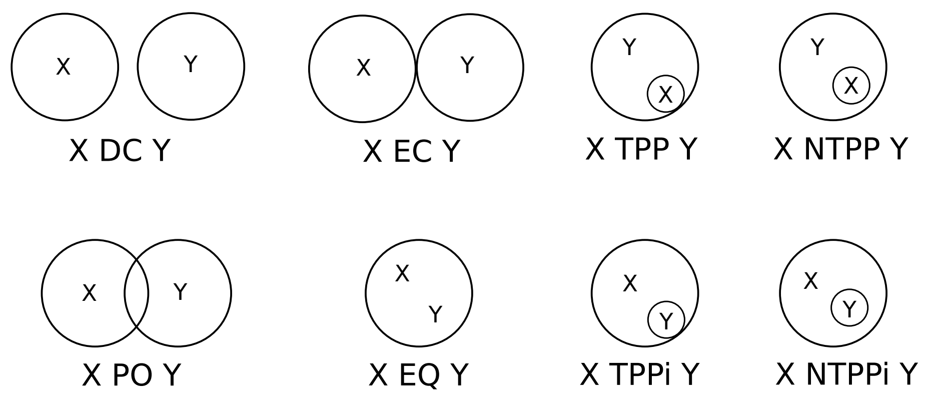


Figure 3.1: The relations of RCC-8

3.3.2 Path Consistency Algorithm

Reasoning about topological spatial relations is done using the well-known path consistency algorithms for the respective calculi [RN99b, RN07]. Path consistency is implemented by introducing rules defining compositions[5] and intersections of supported relations until a fixed point is reached or until an inconsistency is detected. Having defined the composition between any pair of spatial relations, the path consistency algorithm is implemented by iteravely performing the following operation[6]:

$$\forall i, j, k \; R_{ij} \leftarrow R_{ij} \cap (R_{ik} \diamond R_{kj})$$

The above iterative operation is applied until a fixed point is $\overline{R}$ is reached (i.e., operation does not yield new inferences) or until the empty set is reached, implying inconsistency. Computing $\overline{R}$ is done in $O(n^3)$ time.

Algorithm 1 depicts the path consistency algorithm of [RN01]. This algorithm takes as input a set Θ of binary constraints and outputs an equivalent set which is path consistent. If such a set does not exist, the algorithm fails. For this purpose, a queue that contains all possible paths (i, j, k) is created. In each iteration, a path is extracted and deleted from the queue, in order to be revised by function $REVISE(i, j, k)$. Function $REVISE(i, j, k)$ performs the operation

$$M_{ij} \leftarrow M_{ij} \cap (M_{ik} \circ M_{kj})$$

and returns true if M_{ij} is revised and false otherwise. The whole procedure continues until a fixed point $\overline{M}$ is reached. If an empty M_{ij} occurs, then the algorithm fails.

In general, reasoning in the RCC-8 calculus is a NP-complete problem [RN99a]. This means that for very large instances, it is unlikely to decide consistency of a set of spatial formulas in polynomial time. However, there are subsets of RCC-8 which are tractable, and as a result, the consistency problem for them can be decided in polynomial time. These subsets are $\hat{H}8$, $Q8$ (160 relations) and $C8$ (158 relations). For these subsets path consistency is sufficient for deciding consistency.

3.4 State of the Art Reasoners for the RCC-8 Calculus

In this section we briefly review the state of the art qualitative spatial reasoners. We strictly focus on reasoners based on RCC-8, because it is the calculus of choice in extending Strabon with qualitative spatial information.

[5]The composition of any two RCC-8 relations is shown in Table 3.2. The composition of RCC-8 is actually a weak composition [RL05].

[6]Symbol $\diamond$ denotes the weak composition of two relations and R_{xy} denotes the spatial relation holding between regions x and y

Table 3.2: Composition of RCC-8 relations

⋄	**DC**	**EC**	**PO**	**TPP**	**NTPP**	**TPPi**	**NTPPi**	**EQ**
DC	*	DC, EC, PO, TPP, NTPP	DC, EC, PO, TPP, NTPP	DC, EC, PO, TPP, NTPP	DC, EC, PO, TPP, NTPP	DC	DC	DC
EC	DC, EC, PO, TPPi, NTPPi	DC, EC, PO, TPP, TPPi, EQ	DC, EC, PO, TPP, NTPP	EC, PO, TPP, NTPP	PO, TPP, NTPP	DC, EC	DC	EC
PO	DC, EC, PO, TPPi, NTPPi	DC, EC, PO, TPPi, NTPPi	*	PO, TPP, NTPP	PO, TPP, NTPP	DC, EC, PO, TPPi, NTPPi	DC, EC, PO, TPPi, NTPPi	PO
TPP	DC	DC,EC	DC, EC, PO, TPP, NTPP	TPP, NTPP	NTPP	DC, EC, PO, TPP, TPPi, EQ	DC, EC, PO, TPPi, NTPPi	TPP
NTPP	DC	DC	DC, EC, PO, TPP, NTPP	NTPP	NTPP	DC, EC, PO, TPP, NTPP	*	NTPP
TPPi	DC, EC, PO, TPPi, NTPPi	EC, PO, TPPi, NTPPi	PO, TPPi, NTPPi	PO, TPP, TPPi, EQ	PO, TPP, NTPP	TPPi, NTPPi	NTPPi	TPPi
NTPPi	DC, EC, PO, TPPi, NTPPi	PO, TPPi, NTPPi	PO, TPPi, NTPPi	PO, TPPi, NTPPi	PO, TPP, NTPP, TPPi, NTPPi, EQ	NTPPi	NTPPi	NTPPi
EQ	DC	EC	PO	TPP	NTPP	TPPi	NTPPi	EQ

3.4.1 Renz's RCC-8 Solver

In [RN01], Renz et al. describe the path consistency implementation for RCC-8 consistency checking. They employ a $n \times n$ matrix M that holds the spatial relations between all n different regions, including the universal relations. A queue is also implemented as an array of triple pairs (i, j, k),(k, i, j) for regions that represent a path that needs to be revised. The inverse of a spatial relation is precomputed and is assigned directly to the corresponding matrix index when needed. Renz's solver is implemented in C.

3.4.2 PelletSpatial

PelletSpatial [SS09a], as opposed to Renz's solver, ignores universal relations between spatial regions, thus, processing a sparse matrix and iterating over fewer nodes. Further, the queue implementation in [SS09a] doesn't keep track of triples (k, i, j) which correspond to a path with incoming edge for i. Given an edge i, j, the authors look for nodes k and update i, k (which corresponds to triple (i, j, k)). Incoming edges for i are processed by keeping both i, j and j, i in their queue. On revising j, i, the authors look for nodes k and update j, k as well as k, j (which corresponds to triple (k, i, j)). The inverse of a spatial relation is computed on the fly using recursion. PelletSpatial is implemented in Java.

Algorithm 1: PATH-CONSISTENCY

```
Input  : A set $\Theta$ of binary constraints over the variables $x_1, x_2, ..., x_n$ of $\Theta$ represented by an
         $n \times n$ matrix $M$
Output: path-consistent set equivalent to $\Theta$; fail, if such a set does not exist

$Q := \{(i, j, k), (k, i, j) \mid 1 \leqslant i, j, k \leqslant n, i < j, k \neq i, k \neq j\}$;
   ($i$ indicates the $i$-th variable of $\Theta$. Analogously for $j$ and $k$)
while $Q \neq \emptyset$ do
    select and delete a path $(p, r, q)$ from $Q$;
    if REVISE$(p, r, q)$ then
        if $M_{pq} = \emptyset$ then
            return fail;
        else
            $Q := Q \cup \{(p, q, s), (s, p, q) \mid 1 \leqslant s \leqslant n, s \neq p, s \neq q\}$;
```

Algorithm 2: REVISE(i, k, j)

```
Input        : three labels $i$, $k$ and $j$ indicating the variables $x_i, x_j, x_k$ of $\Theta$
Output       : true, if $M_{ij}$ is revised; false otherwise
Side Effects: $M_{ij}$ and $M_{ji}$ revised using the operations $\cap$ and $\circ$ over
               the constraints involving $x_i, x_k$ and $x_j$

oldM := $M_{ij}$;
$M_{ij} := M_{ij} \cap (M_{ik} \circ M_{kj})$;
if (oldM = $M_{ij}$) then
    return false;
$M_{ji} := M_{ij}^{\smile}$;
return true;
```

3.4.3 Generic Qualitative Reasoner

Generic Qualitative Reasoner (GQR) [GWW08] is a solver for binary qualitative constraint networks, that supports arbitrary binary constraint calculi developed for spatial and temporal reasoning, such as the calculi from the RCC family, and Allen's interval algebra. The main innovation of GQR whith respect to the previously mentioned reasoners, is that it uses a hand-written queue put together from generic components (parts of the C++ Standard Template Library STL), implemented as a virtual class, allowing for easy and convenient extensions. GQR also employs hash tables and different methods for creating and handling precomputations of composition and inverse tables. GQR is implemented in C++ and makes extensive use of templates.

3.4.4 PyRCC8

PyRCC8[7] is an open source, efficient QSR for RCC-8 written in Python. PyRCC8 is implemented using PyPy, a fast, compliant implementation of the Python 2 language. To the best of our knowledge, PyRCC8 is the first implementation of a QSR on top of a trace-based JIT compiler. Previous implementations have used either static compilers, e.g., Renz's solver [RN01] and GQR [GWW08], or method-based JIT compilers, e.g., the RCC-8 reasoning module of the description logic reasoner PelletSpatial [SS09a]. The advantage of trace-based JIT compilers is that they can discover optimization opportunities in common dynamic execution paths, that are not apparent to a static compiler or a method-based JIT compiler [BCFR09, BCF+11]. PyRCC8 offers a path consistency algorithm for solving tractable RCC-8 networks and a backtracking-based

[7] `http://pypi.python.org/pypi/PyRCC8/`

algorithm for general networks. Both algorithms draw on the original ideas of [RN01], but offer some more interesting features that we discuss as follows. PyRCC8's path consistency algorithm processes only meaningful arcs, i.e., arcs that do not correspond to the universal relation[8], thus, doing also fewer consistency checks. Regarding concistency of general RCC-8 networks, PyRCC8 is the only reasoner we know that offers an iterative counterpart of the recursive backtracing algorithm. Additionaly, PyRCC8 precomputes the converse of relations to avoid time consuming and exhaustive repetition of converse computations for large datasets.

The design and implementation of PyRCC8 has been carried out in the context of the TELEIOS project and a preliminary version and comparison with the reasoners discussed above has been published in [SK12]. An extended version of this work has already been submitted to a journal, a copy of which is given in Appendix C at the end of this deliverable. Therefore, the interested reader may find additional information and more details about the technical work underlying the development of PyRCC8 there.

3.5 Enabling RCC-8 Reasoning in a Database

In this section we report on a number of implementations of a path consistency algorithm for the RCC-8 calculus in relational database systems. We consider both the PostgreSQL and MonetDB databases in our implementations. As a first step towards enabling RCC-8 reasoning in a database system, we transfer the implementation of the path consistency algorithm for the RCC-8 calculus as presented in [RN01] inside the codebase of MonetDB (Section 3.5.1). Then, we investigate whether such an algorithm could be expressed following the relational model and the procedural extensions of SQL. For this purpose, we consider both PostgreSQL and MonetDB and develop a SQL program that implements the path consistency algorithm (Section 3.5.2).Last, we go deeper by implementing a path consistency algorithm using the application programming interface of MonetDB. This implementation has been based on the SQL implementation mentioned previously, but uses the primitives that the kernel of MonetDB provides, namely the Binary Associative Table (BAT) data structure and operations on them (Section 3.5.3).

The ultimate goal of this section is to end-up with a database system extended with reasoning facilities for the RCC-8 calculus. This goal is the stepping stone towards the development of query processing techniques for the stSPARQL and GeoSPARQL query languages enabling them to answer queries involving qualitative and quantitative spatial information, such as the ones mentioned in the motivation section of this work (Section 3.1).

Notice that the implementations of this section are targeted towards checking constraint networks for path consistency. Thus, these implementations cannot solve the full problem of consistency in RCC-8 networks. This is intended since the problem of deciding whether such a constraint network is consistent has been shown to be NP-complete [RN99a]. Thus, we only target the tractable fragments of the RCC-8 calculus [RN99a] which have been show to be solvable by path consistency algorithms.

3.5.1 Transferring Renz's solver to MonetDB

This implementation of the path consistency algorithm uses exactly the same C code as the Renz solver path consistency function does. The structures are also the same as the structures of the original Renz algorithm. This means that there is a $n \times n$ matrix M for holding the spatial relations between all n different regions, and another matrix of equal size for marking the pairs of regions to be revised for consistency. However, the input of the algorithm, that is the RCC-8 constraint network, is represented as a relational table with three attributes `u`, `v`, `r` denoting the

[8]The result of the composition of any relation with the universal relation is the universal relation.

spatial relation `r` through which nodes `u` and `v` are related. In the terminology of MonetDB, these attributes correspond to three Binary Associative Tables (BAT), the internal datastructures of MonetDB for representing a column of a relational table, and we will refer to them as BAT `u`, BAT `v`, and BAT `r`. For a more detailed description of how an input constraint network is represented using a relational table, see the following section.

Transferring Renz's solver to MonetDB consists of three phases. In the first phase, the constraint network represented in BATs `u`, `v` and `r` are imported to matrix M. The next step is the execution of the Renz path consistency algorithm. If the network is consistent, matrix M contains new values that represent the complete graph. In final step, MonetDB is updated with these values, but only includes relations r which connect two regions i and j, where $i < j$. This happens because, if a region i is connected with j through relation r, then j is connected with i through the inverse relation r_i. Also, pairs of regions that are connected through the relation `DALL` are not transferred to MonetDB.

Detailed information to get the source code of this implementation and instructions on how to invoke it are provided in Section B.1 of the Appendix.

3.5.2 The Path Consistency Algorithm as a SQL program

In this section we discuss the implementaion of the path consistency algorithm as presented in Section 3.3.2 as a SQL program. Before getting into the details of the implementation, we introduce the reader to the encoding we use for representing a set of RCC-8 relations and then how we model an RCC-8 constraint network under the relational model.

Encoding of RCC-8 relations

Table 3.3 shows how the RCC-8 relations are encoded. Like the implementation of [RN01], we have opted to encode RCC-8 relations using a binary encoding. In particular, a RCC-8 relation corresponds to an integer number which is expressed in a power of 2. This encoding scheme has the following advantages: a) the encoding of a *set* of RCC-8 relations corresponds to a single integer number which can be computed by calculating the bitwise OR of the relations it contains and b) computing the intersection of two or more sets of RCC-8 relations reduces to computing the bitwise AND of the binary encodings of these sets. For example, the set of relations $\{\mathrm{DC}, \mathrm{TPP}, \mathrm{EQ}\}$ is encoded to the binary number 10001001 which corresponds to the integer number 137. This number is the result of computing the bitwise OR of the binary encodings 00000001, 00001000, and 10000000 of relations $\mathrm{DC}, \mathrm{TPP}$ and EQ respectively. Likewise, the set of all basic RCC-8 relations, denoted by DALL, is encoded to the binary number 11111111 which corresponds to the integer number 255. Computing the bitwise AND between this encoding and the encoding of the previous set results in the binary number 10001001 which is the encoding for set $\{\mathrm{DC}, \mathrm{TPP}, \mathrm{EQ}\}$ we computed previously.

Modeling of a RCC-8 constraint network using the relational model

An RCC-8 constraint network N is modeled in the relational model as a ternary relation `R(u int, v int, r int)`. If N contains a labeled edge $R(x, y)$ between nodes x and y, then the relational table `R` contains a tuple `<x, y, r>` where `x`, `y`, `r` are appropriate integers encoding the nodes x, y, and relation R.

The composition table for RCC-8 relations is precomputed in a ternary relational table `rcc8_-trans(r1 int, r2 int, comp int)` as well. In this relational table, `r1`, `r2`, and `comp` are sets of

Table 3.3: Encoding of RCC-8 relations

Relation	**Integer encoding**	**Binary encoding**
DC	1	00000001
EC	2	00000010
PO	4	00000100
TPP	8	00001000
NTPP	16	00010000
TPPi	32	00100000
NTPPi	64	01000000
EQ	128	10000000

RCC-8 relations with `comp` corresponding to the composition of the set of relations `r1` with the set of relations `r2`. The result of composition for two sets of relations is the set union of the result of the composition of each pair of RCC-8 relations from each set (see Table 3.2).

Let us now see an example of an RCC-8 constraint network[9] and the corresponding modeling as a relational table.

Example 14. Two houses are connected via a road. Each house is located on an own property. The first house possibly touches the boundary of the property; the second one surely does not. What can we infer about the relation of the second property to the road?

Let us first write down the spatial relations that hold between the objects of the previous description. These are the following:

```
house1 DC house2
house1 {TPP, NTPP} property1
house1 {DC, EC} property2
house1 EC road
house2 {DC, EC} property1
house2 NTPP property2
house2 EC road
property1 {DC, EC} property2
road {DC, EC, TPP, TPPi, PO, EQ, NTPP, NTPPi} property1
road {DC, EC, TPP, TPPi, PO, EQ, NTPP, NTPPi} property2
```

The last two relations above relate the road to the two properties of the houses. The set of relations that is used correspond to the DALL relation, i.e., the set of all basic RCC-8 relations. Notice that if there is no information about how two regions are spatially related, then DALL is the proper set of relations to assume.

The corresponding RCC-8 constraint network is given in the following figure. In this network, the nodes represent the objects mentioned in the example. The encoding of the objects to integers is given in Table 3.4.

[9]This example is taken from Wikipedia: `http://en.wikipedia.org/wiki/Region_Connection_Calculus`

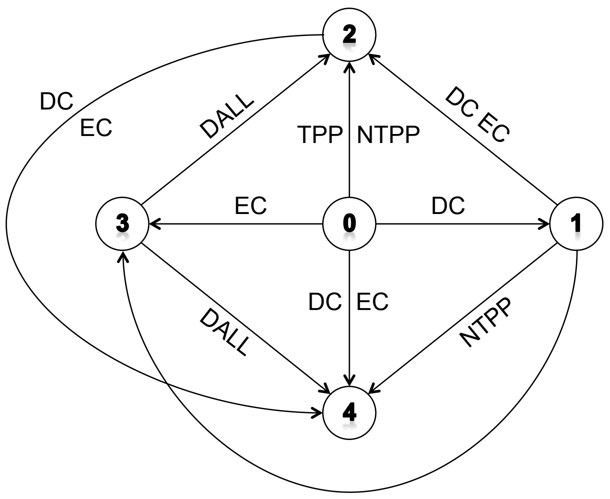


Table 3.4: Integer encoding of objects mentioned in Example 14

Object	**Integer encoding**
house1	0
house2	1
property1	2
road	3
property2	4

In the following the relational table `R` corresponding to the above RCC-8 constraint network is given.

u	**v**	**r**	**set of RCC-8 relations**
0	1	1	$\{\text{DC}\}$
0	2	24	$\{\text{TPP}, \text{NTPP}\}$
0	3	2	$\{\text{EC}\}$
0	4	3	$\{\text{DC}, \text{EC}\}$
1	2	3	$\{\text{DC}, \text{EC}\}$
2	4	3	$\{\text{DC}, \text{EC}\}$
1	4	16	$\{\text{NTPP}\}$
1	3	2	$\{\text{EC}\}$
3	4	255	DALL
3	2	255	DALL

The implementations we consider in the following assume that if any two nodes of an input RCC-8 constraint network are connected, then they have to be connected in both directions. Notice that if an RCC-8 network N contains an edge $R(x, y)$, then also we know the label for the edge (y, x) (the inverse). Table 3.5 shows the inverse relation for each one of the basic RCC-8 relations. If region x is related through relation R to region y, i.e., relation $R(x, y)$ holds, then Table 3.5 gives us the relation through which region y is related to region x, i.e., relation $RI(y, x)$.

Returning back to Example 14, the relational table produced by the addition of the inverses for each edge is given in Table 3.6.

Implementation in MonetDB

Figure 3.2 depicts the SQL program that implements the path consistency algorithm in MonetDB. The algorithm takes as input a constraint network R and iteratively performs the operation:

$$\forall i, j, k \; R_{ij} \leftarrow R_{ij} \cap (R_{ik} \circ R_{kj})$$

Table 3.5: Inverse relations for RCC-8 basic relations

Initial relation $R(x,y)$	**Inverse relation** $RI(y,x)$
DC	DC
EC	EC
PO	PO
TPP	TPPi
NTPP	NTPPi
TPPi	TPP
NTPPi	NTPP
EQ	EQ

Table 3.6: Relational table R corresponding to Example 14 populated with inverse relations

u	**v**	**r**	**set of RCC-8 relations**
0	1	1	{DC}
0	2	24	{TPP, NTPP}
0	3	2	{EC}
0	4	3	{DC, EC}
1	2	3	{DC, EC}
2	4	3	{DC, EC}
1	4	16	{NTPP}
1	3	2	{EC}
3	4	255	DALL
3	2	255	DALL
1	0	1	{DC}
2	0	96	{TPPI, NTPPI}
3	0	2	{EC}
4	0	3	{DC, EC}
2	1	3	{DC, EC}
4	2	3	{DC, EC}
4	1	64	{NTPPI}
3	1	2	{EC}
4	3	255	DALL
2	3	255	DALL

The composition between relations R_{ik} and R_{kj} is computed using the table `rcc8_trans`, which is the precomputed composition table for RCC-8 relations as described in Section 3.5.2. At the end of every loop, the algorithm investigates whether a fixed point $\overline{R}$ is reached, in order to terminate the loop. Below, a more detailed explanation of the code is given:

- Lines 27–31: Self join of table R, in order to create a relation which contains all possible paths of length two that result from the input constraint network. This is implemented by finding edges that share a common node.
- Lines 25–33: Computation of the composition between every two relations R_{ik} and R_{kj} by joining with table `rcc8_trans`.
- Lines 23–37: Intersection of edges that come from different paths and additional intersection with edge R_{ij} . For this purpose, the `bit_and` aggregate function was implemented as an extension to MonetDB, which is described in Section B.5 as well. Line 37 contains a full outer join because some edges may have not been involved in a path and we want to keep them.
- Lines 9–38: A new table `itR` is created, which contains the new edges. This table serves as the result of the current iteration and the input to the next one.
- Lines 39–40: Keep the difference of tables `itR` and R to the variable that determines whether the loop will be terminated or not.

Detailed information to get the source code of this implementation and instructions on how to invoke it are provided in Section B.2 of the Appendix.

Implementation in PostgreSQL

Figure 3.3 depicts the SQL program that implements the path consistency algorithm in PostgreSQL. The code is similar to that of MonetDB. The differences that exist are due to the dissimilarities of the SQL that the two database systems support.

Detailed information to get the source code of this implementation and instructions on how to invoke it are provided in Section B.3 of the Appendix.

3.5.3 Implementation of an RCC-8 module in MonetDB

We have also done a native implementation of the path consistency algorithm in MonetDB as an RCC-8 module. This implementation is native to the MonetDB in the sense that the path consistency algorithm is implemented using the application programming interface of MonetDB. This implementation has been based on the SQL implementation mentioned previously, but uses the primitives that the kernel of MonetDB provides, namely the Binary Associative Table (BAT) data structure and the associated operations (e.g., select, join, semi-join, mirror) on them.

Detailed information to get the source code of this implementation and instructions on how to invoke it are provided in Section B.4 of the Appendix.

3.6 Experimental Evaluation

In this section we evaluate experimentally and compare the performance of the implementations of the path consistency algorithm as depicted in Table 3.7. Notice that the implementation corresdonding to PyRCC8$\bigtriangledown$ uses the notion of partial path consistency instead of the notion of path consistency that is employed in all the others.

```
CREATE FUNCTION pc ()
RETURNS TABLE (edge_count INTEGER)
BEGIN
    DECLARE TABLE itR (u INTEGER, v INTEGER, r INTEGER);
    DECLARE delta int;
    SET delta = (SELECT count(*) FROM R);
    WHILE (delta <> 0)
    DO
      INSERT INTO itR (u, v, r) (
        SELECT DISTINCT
        CASE WHEN R.u IS NULL
             THEN T.u ELSE R.u
        END,
        CASE WHEN R.v IS NULL
             THEN T.v ELSE R.v
        END,
        CASE WHEN (R.r IS NOT NULL AND T.r is NOT NULL)
             THEN R.r & T.r
             WHEN R.r IS NULL
             THEN T.r ELSE R.r
        END
        FROM
            (SELECT TRANS.x as u, TRANS.z as v, bit_and(comp) as R
             FROM
                 (SELECT TR.x, TR.z, comp

                  FROM
                       (SELECT TR1.u as x, TR1.v as y, TR2.v as z,

                                     TR1.r as rxy, TR2.r as ryz
                        FROM
                            R as TR1 JOIN R as TR2
                                 ON (TR1.v = TR2.u AND TR1.u <> TR2.v)

                       ) as TR JOIN "rcc8_trans"
                                 ON (TR.rxy = s AND TR.ryz = t)

                 ) AS TRANS
                 GROUP BY TRANS.x, TRANS.z
            ) AS T
            FULL OUTER JOIN R ON (R.u = T.u AND R.v = T.v)
        );
        SET delta = (SELECT count(*) FROM
                        (SELECT * FROM itR EXCEPT (SELECT * FROM R)) AS DIFF);
        IF delta <> 0
        THEN
            DELETE FROM R;
            -- update according to new computation
            INSERT INTO R (SELECT * FROM itR);
        END IF;
        -- initialize intermediate result
        DELETE FROM itR;
    END WHILE;
    RETURN ((SELECT count(*) FROM R));
END;
```

Figure 3.2: Path consistency algorithm as SQL program in MonetDB

```
CREATE FUNCTION pc ()
RETURNS integer as $$
DECLARE
        delta int;
BEGIN
CREATE TABLE itR (f int, t int, rel int);
delta := count(*) FROM R;
WHILE delta <> 0
LOOP
        -- compute transitive closure: refine R, keep unaffected edges, and add
            new edges
        INSERT INTO itR (
        SELECT DISTINCT
        CASE WHEN R.u is Null
             THEN T.u
             ELSE R.u
        END,
        CASE WHEN R.v is Null
             WHEN T.v
             ELSE R.v
        END,
        CASE WHEN (R.r is NOT Null AND T.r is NOT Null)
             THEN R.r & T.r
             WHEN R.r is Null
             THEN T.r
             ELSE R.r
        END
        FROM
          (SELECT TRANS.x as u, TRANS.z as v, bit_and(comp) as r
           FROM
            (SELECT TR.x, TR.z, comp
             FROM
               (SELECT TR1.u as x, TR1.v as y, TR2.v as z, TR1.r as rxy, TR2.r
                   as ryz
                 FROM R as TR1 JOIN R as TR2
                 ON (TR1.v = TR2.u AND TR1.u <> TR2.v)
               ) as TR JOIN "rcc8_trans"
               ON (TR.rxy = s AND TR.ryz = t)
            ) AS TRANS
            GROUP BY TRANS.x, TRANS.z
          ) AS T
        FULL OUTER JOIN R ON (R.u = T.u AND R.v = T.v)
        );
        delta := count(*) FROM (SELECT * FROM itR EXCEPT (SELECT * FROM R)) AS
            DIFF;
        IF delta <> 0
        THEN
                TRUNCATE R;
                INSERT INTO R (SELECT * FROM itR);
        END IF;
        TRUNCATE itR; -- initialize intermediate result
END LOOP;
DROP TABLE itR;
RETURN count(*) FROM R;
END;
```

Figure 3.3: Path consistency algorithm as SQL program in PostgreSQL

Table 3.7: Codenames of different implementations of the path consistency algorithm

Codename	Detailed description	Short description
PostgreSQL	Section 3.5.2	Implementation of PC algorithm for RCC-8 as a SQL program in PostgreSQL
MonetDB-SQL	Section 3.5.2	Implementation of PC algorithm for RCC-8 as a SQL program in MonetDB
MonetDB-ext	Section 3.5.3	Implementation of PC algorithm for RCC-8 using the MonetDB API
MonetDB-Renz	Section 3.5.1	Transfer of Renz's implementation in MonetDB
Renz	Section 3.4.1	Original implementation of the path consistency algorithm of [RN01] by Jochen Renz
PyRCC8	Section 3.4.4 and [SK12]	Implementation of an RCC-8 reasoner in python
PyRCC8$\triangledown$	[SK12]	Extension of the above to run PPC instead of PC

3.6.1 Hardware setup

Our experiments were conducted on an Intel Xeon E5620 with 12MB L3 cache running at 2.4 GHz. The system has 48GB of RAM and 4 disks of striped RAID (level 5). The operating system is Ubuntu 12.04 (Precise).

3.6.2 Datasets

In the following we describe the datasets we used to evaluate our implementations.

The Administrative Geography of Great Britain

The comparison of the different implementations was performed using the Administrative geography (admingeo) of Great Britain dataset. The admingeo dataset describes the hierarchy of administrative units and the topological relations among them. Therefore, the corresponding ontology includes classes that represent the administrative units of Great Britain, and properties that describe the qualitative topological relations. Two ontologies, the Geometry Ontology and the Spatial Relations Ontology are used to provide geospatial vocabulary. These ontologies describe abstract geometries and basic spatial (equivalent to RCC-8) relations respectively. Boundary-Line, a polygon dataset of areas defined by electoral and administrative boundaries, was then used to generate the topological relations in RDF based on the provided names and boundary information. The resulting dataset was combined with addresses, roads, land use, height and other datasets available in RDF. The constraint network that the dataset comprises is consistent and contains more than 10000 nodes and nearly 80000 relations. For the experiments, the initial dataset was divided into smaller datasets of different size, according to the number of relations that they contain.

Greek Administrative Geography

The Greek Administrative Geography (GAG) dataset describes the administrative divisions of Greece (prefecture, municipality, district, etc.). It has been populated with relevant data that are available in the Greek open government data portal. For each administrative unit in the ontology

Table 3.8: Dataset characteristics

Dataset		#nodes	#edges
admingeo	100	53	100
	500	186	500
	1000	326	1000
	2000	560	2000
	5000	1175	5000
	10000	2118	10 000
	20000	3947	20 000
	25000	4710	25 000
	30000	5504	30 000
	35000	6251	35 000
	40000	7009	40 000
	45000	7771	45 000
	50000	8492	50 000
	55000	9149	55 000
	60000	9791	60 000
	65000	10 521	65 000
	70000	11 150	70 000
	75000	11 727	75 000
	77000	11 761	76 996
	full	11 762	77 907
gag		1458	2176
nuts		2236	3176
gadm-geov.		276 728	590 443
gadm		42 750	159 600

(e.g., a municipality) various pieces of information are available (e.g., population and geographical boundaries).

In the context of this deliverable, following the methodology of [GDH08], we populated the GAG dataset with topological information using the relations of the RCC-8 calculus. We followed the following procedure. For each administrative level, we checked whether the geometries of two administrative units touch, in order to infer the EC topological relation of RCC-8 for them. Then, for each pair of consecutive administrative levels, we exploited the existing relation `belongs_To` that holds between them to infer the NTPP or TPP topological relation of RCC-8. To decide which one of these two relations hold, we checked whether the boundaries of the two regions intersect. The population of the GAG dataset with topological information was done solely by the evaluation of stSPARQL queries on Strabon. The part of the GAG dataset with topological information contains 1139 relations and 420 regions.

Global Administrative Areas

The Global Administrative Areas dataset (GADM) contains the geometries of administrative areas of all countries. GADM consists of three subdivisions of administrative areas. The top level contains countries and the two lower levels are subdivisions for each country. The dataset comes in various formats, such as ESRI shape files, KML/KMZ files, or RData (for use in the statistical project R). We transformed ESRI shape files in RDF and populated the resulting data with topological information using the methodology described in Section 3.6.2.

The GADM dataset is also available by the Geovocab[10] project, which provides vocabularies for geospatial modelling. Geovocab offers GADM in RDF format[11] and has also populated it with topological information providing links to existing spatial datasets, with the ultimate goal of enhancing the integration of spatial information in the semantic web. However, Geovocab has used the topological relations of the RCC-5 calculus. Based on this dataset, we have transformed it into one which contains relations of RCC-8 calculus. This was possible because the dataset contained only the RCC-5 relations EQ, PPI, and PP, which can be represented in RCC-8 using the set of relations $\{\mathrm{EQ}\}$, $\{\mathrm{TPPI}, \mathrm{NTPPI}\}$ and $\{\mathrm{TPP}, \mathrm{NTPP}\}$ respectively. We refer to the resulting dataset as GADM-Geovocab.

Nomenclature of Territorial Units for Statistics

The Nomenclature of Territorial Units for Statistics[12] (NUTS) is a hierarchical system defined by the Eurostat office of the European Union for dividing the economic territory of the EU for the purpose of collecting, developing, and harmonising EU regional statistics, analysing EU regions based on socio-economic factors, and framing of EU regional policies. NUTS is divided in four levels NUTS-0, NUTS-1, NUTS-2, and NUTS-3. NUTS-0 comprises EU countries (e.g., Germany, Greece). NUTS-1 comprises the major socio-economic regions (e.g., North Rhine-Westphalia, Central Greece). NUTS-2 comprises the basic regions for the application of regional policies (e.g., Berlin, Attica), while NUTS-3 comprises small regions for specific diagnoses (e.g., Hamburg, Evros).

Geovocab has published this data in RDF as linked data[13] using the NeoGeo RDF Vocabulary for GeoData[14]. Again, the topological relations of this dataset contains RCC-5 relations. We used the same procedure as above to transform them in RCC-8 relations.

Table 3.8 summarizes the characteristics of the aforementioned datasets.

[10] `http://geovocab.org/`
[11] `http://gadm.geovocab.org/`
[12] `http://epp.eurostat.ec.europa.eu/portal/page/portal/nuts_nomenclature/introduction`
[13] `http://nuts.geovocab.org/`
[14] `http://geovocab.org/doc/neogeo/`

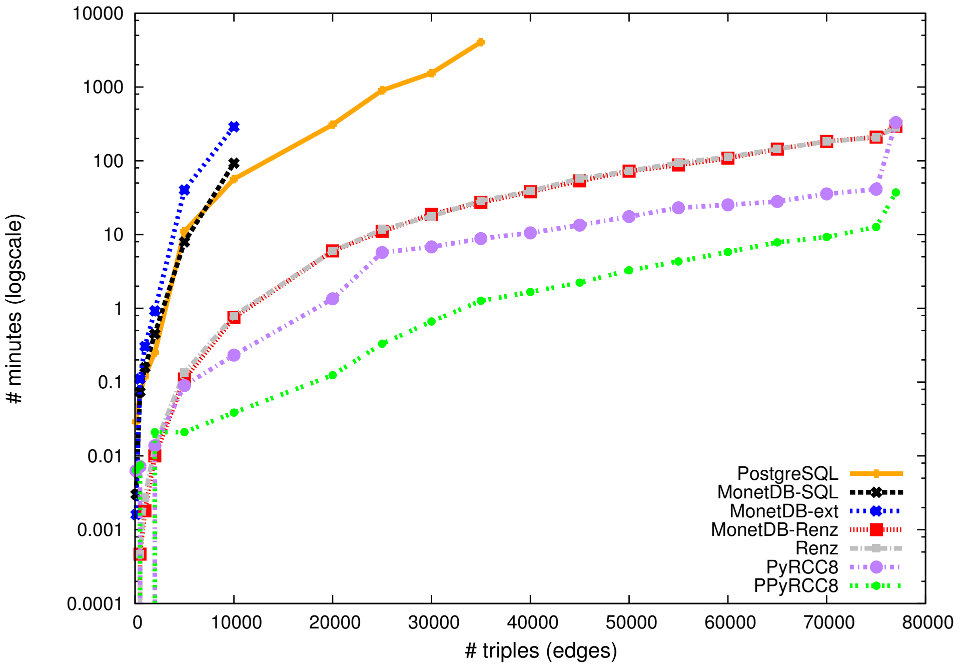


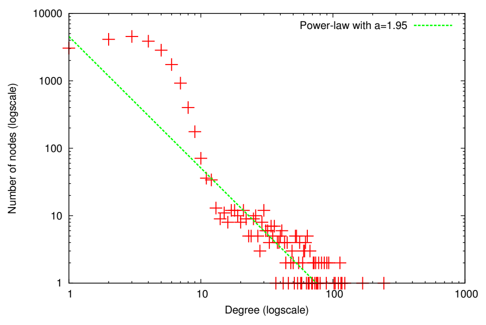
Figure 3.4: Performance of consistency checking for the admingeo RDF dataset

3.6.3 Evaluation

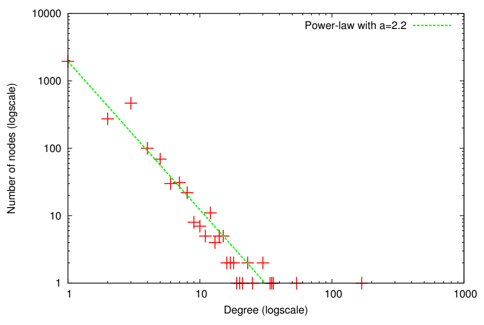
Figure 3.4 shows the performance of consistency checking on the admingeo RDF dataset. We observe that all implementations that are based on the relational model, i.e., PostgreSQL, MonetDB-SQL, and MonetDB-ext, have the worst performance with the exception of MonetDB-Renz. This is reasonable since the MonetDB-Renz implementation models a constraint network as a two dimensional array for running the path consistency algorithm, while it uses the relational model to store the constraint network. Among these implementations, PostgreSQL outperforms both the relational-based implementations in MonetDB, that is, MonetDB-SQL and MonetDB-ext. For these implementations, after the limit of 10000 relations, no results can be obtained due to the fact that the memory requirements for MonetDB exceed the available main memory, forcing the algorithm to make use of the swap space. Regarding MonetDB-ext and MonetDB-SQL, MonetDB-SQL performs better. This is justified since MonetDB always maps a SQL query to an equivalent program in its internal assembly-like language (called MAL) and uses this program to optimize query evaluation. Thus, the MonetDB-ext implementation, since it implements the path consistency algorithm using the BAT data structure and the associated operations on BATs, could only be as good as MonetDB-SQL.

Regarding the performance of the other implementations shown in Figure 3.4, the best performance for checking consistency of RCC-8 networks can be obtained using the reasoners PyRCC8 and PyRCC8$\triangledown$. Another observation is that the Renz and MonetDB-Renz implementations behave similarly regarding performance. This is to be expected since MonetDB-Renz implementation runs internally the code of Renz behaving in this respect as a wrapper of Renz using the relational model.

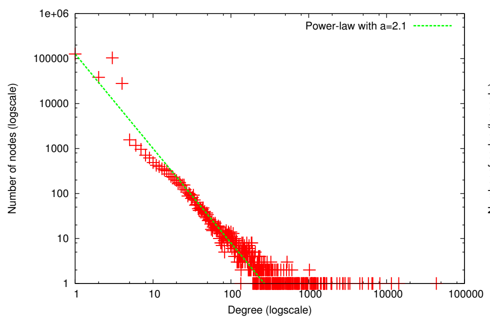
Regarding the performance of consistency checking for larger datasets than the dataset `admingeo`, we observed that even PyRCC8$\triangledown$, which exposes the best performance among the state of the art qualitative spatial reasoners, cannot handle them. An example is the `gadm-geovocab` dataset which is an RCC-8 network with around 600 thousands edges and 45 thousands nodes. The last four implementations of Figure 3.4 allocate a two-dimensional array to represent the input constraint network. This array is of size N^2 where N is the number of nodes of the input constraint network.



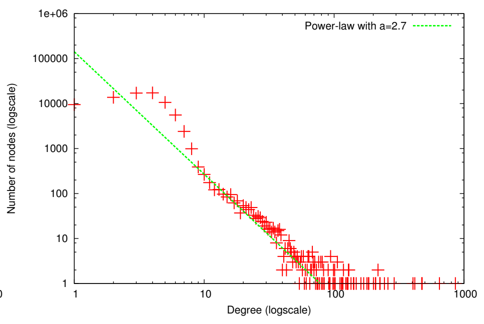
(a) `admingeo`



(b) `nuts`



(c) `gadm-geovocab`



(d) `gadm`

Figure 3.5: Degree distributions of various datasets

Thus, even for medium-sized graphs, these implementations fail to run, a drawback that is not present in the implementations PostgreSQL, MonetDB-SQL, and MonetDB-ext, which not only can load the input constraint graph, but can also complete some iterations of the algorithm.

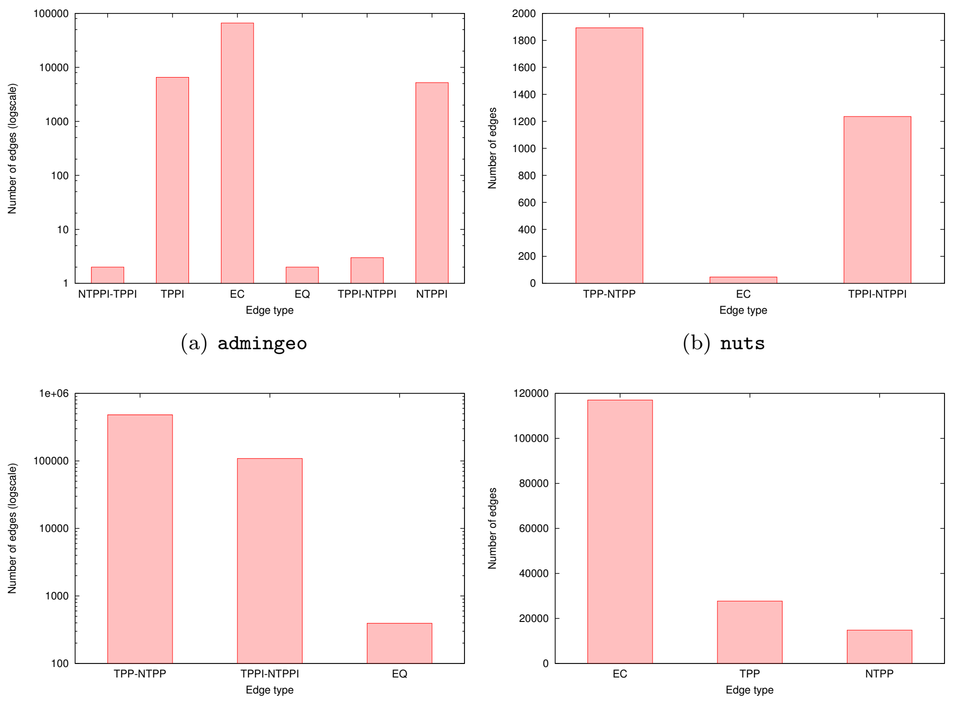
The most interesting and intriguing observation for such graphs is that they are very sparse, hence representations of the input graph that are based on two-dimensional arrays are not appropriate at all. This fact is well depicted in Figure 3.5 where the degree distribution among the nodes of the various datasets of Table 3.8 is shown. Figure 3.5 shows that all these RCC-8 constraint networks are sparse and follow a power-law distribution[15] which has been shown to fit the Internet web graph and the Twitter follower graph [FFF99, GLG+12]. The primary characteristic of such graphs is that most of the nodes have very low degree, while only a small number of the nodes have high degree number leading to star-shaped graphs. In particular, if the degree distribution among the nodes of a graph follows a power-law distribution, then the number of nodes with degree number x can be approximated by the following power-law function:

$$f(x) = \frac{c}{x^{\alpha}} \text{ with } \alpha > 0 \tag{3.1}$$

In the above function, c and α are parameter values which are empirically chosen. When the values for c and α are equal to 1, then this function becomes the well-known Zipf function. For natural graphs, the value of α has been shown to be around 2 [FFF99, GLG+12]. Moreover, the value of parameter α determines the skeweness of the degree distribution among the nodes of the graph. Higher values for α imply that the graph has lower density, that is, the ratio of edges to nodes is small. As the value for α decreases, the graph density increases, thus the number of nodes with high degree number increases as well [GLG+12].

One more observation that calls for optimization is the kind of edges that these real constraint graphs comprise. Figure 3.6 shows the number of edges grouped by type for the datasets of Table

[15]One could admittedly argue that the `admingeo` dataset does not fit this distribution.



(a) `admingeo` (b) `nuts`

(c) `gadm-geovocab` (d) `gadm`

Figure 3.6: Number of edges grouped by type for various datasets

3.8. With the exception of the `gadm` dataset, in all others the most popular edges are those that relate two regions with a containment relation, that is, TPP, NTPP, their inverses, or a disjunction of these. In addition, the presence of EC relations is also high. In particular, the high frequency of the containment edges calls for further optimization as has been done in the case of temporal reasoning [GS95, DGA01].

4. Conclusions

In this report we presented the second phase of the implementation of a temporal and spatial extension of RDF and SPARQL on top of MonetDB. First, we described how we extended the data model RDF and the query language SPARQL with the valid time dimension and how Strabon was extended resspectively to support this functionality. In this context, we also provided the results of our first experimental evaluation, where Strabon competed with implementations offering similar functionalities and the results are very encouraging.

Second we investigated different kinds of implementations of the path consistency algorithm for checking the consistency of tractable fragments of RCC-8 calculus in a database system, such as PostgreSQL and MonetDB. We compared these implementations with the state of the art reasoners for RCC-8 and concluded that the relational model is not an appropriate model in checking the consistency of RCC-8 networks. In face of this discouraging result, we developed the reasoner PyRCC8 which outperforms the state of the art RCC-8 reasoners and also incorporated Renz's solver into the codebase of MonetDB. This way, we have prepared the ground for the development of query processing techniques for the stSPARQL and GeoSPARQL query languages targeting at querying incomplete spatial information.

For the rest of the project, we plan to continue our evaluation and benchmarking efforts. We plan to test the valid time features of Strabon with larger, real and synthetic, datasets. We also plan to compete with Allegrograph to test the temporal capabilities of the two systems with respect to the user-defined time, as Allegrograph is not valid time enabled. We also plan to..

In the forthcoming deliverable named: "The evaluation of the developed implementation" we will report on the results of the experimental evaluation of our implementations.

A. Instructions for installing and using the temporal component of Strabon

A.1 Instructions for installing and using the temporal component of Strabon

In this section we provide guidelines for installing and using the temporal component of Strabon.

Installation

In order to install the temporal component of Strabon the following steps should be performed:

- Create a spatially and temporally enabled database by performing the following steps:
  - Download and install PostgreSQL. A PostgreSQL database will be used as back-end.
  - Download and install the PostgreSQL Temporal extension of PostgreSQL as described here: `http://github.com/jeff-davis/PostgreSQL-Temporal`.
  - Download the spatiotemporal database template which is available here: `http://www.earthobservatory.eu/wiki/attach/WP4/spatiotemporal.sql`.
  - Create a new PostgreSQL database (without using any template), for example:
    ```
    $> sudo -u postgres createdb spatiotemporal
    ```
  - Import the spatiotemporal.sql that you downloaded into the newly created database. This action will load all PostGIS and PostgreSQL Temporal functions making your database spatially and temporally enabled.
    ```
    $> sudo -u postgres psql -d spatiotemporal -f spatiotemporal.sql
    ```
- Clone the source code of Strabon from the Strabon mercurial repository (Also available as a zip file here: `http://www.earthobservatory.eu/wiki/attach/WP4/StrabonTemporal.zip`).
  ```
  $> hg clone http://hg.strabon.di.uoa.gr/Strabon/
  ```
- Go to the directory where Strabon is located in
  ```
  $> cd Strabon
  ```
- Change to the "temporals" branch
  ```
  $> hg update temporals
  ```
- Build
  ```
  $> mvn clean package
  ```

Store

Currently, the temporal component of Strabon supports the N-QUADS format for storing stRDF graphs that contain triples annoated with their valid time. To store a file in N-QUADS format one should execute the following command:

```
scripts/strabon -h <hostname> -p <port> -db <database> -u <username> -pass <password> store
-f nquads <filepath>
```

The command provided above uses a strabon bash script that is located in Strabon/scripts and is used as an interface for executing store, query and update commands in Strabon.

For example, let us store the file: `runtime/src/test/resources/temporal-periods.nq`

using the following command:

```
scripts/strabon -h localhost -p 5432 -db spatiotemporal -u postgres -pass postgres store -f
nquads runtime/src/test/resources/temporal-periods.nq
```

Query

The command for executing queries against Strabon using the strabon script is the following:

```
scripts/strabon -h <hostname> -p <port> -db <database> -u <username> -pass <password> query
<query>.
```

The argument <query> is a string representation of an stSPARQL query. For example, let us pose the following query against the stRDF graph that we stored in the previous step:

```
SELECT distinct ?x1 ?x2
WHERE {
      ?x1 ?y1 ?z1 ?t1 .
      ?x2 ?y2 ?z2 ?t2 .
      FILTER(<http://strdf.di.uoa.gr/ontology#before>(?t1, ?t2) && ?x2!=?x1). }
```

The query provided above retrieves subjects of triples in pairs such that the valid time of the first is before the valid time of the second. We can pose this query against Strabon using the strabon script as follows:

```
$>scripts/strabon -h localhost -p 5432 -db spatiotemporal -u postgres -pass postgres query QUERY
```

where `QUERY` is the string representation of the query provided above.

B. Instructions for running the implementations of Chapter 3

In the following, we provide instructions on getting the source code corresponding to the implementations discussed in Section 3.5 and running a simple example to demonstrate its usage. In particular, those implementations that are based on MonetDB (i.e., those discussed in Sections 3.5.1, 3.5.2, and 3.5.3) have been based on MonetDB and the release of the October, 2012. The corresponding code is available at `http://earthobservatory.eu/wiki/attach/WP4/MonetDB-Oct2012_rcc8-renz.tar.bz2`. For the SQL implementation in PostgreSQL, the code is available at `http://www.earthobservatory.eu/wiki/attach/WP4/rcc8-postgres.zip`.

We assume that the directories that contain the source code of PostgreSQL and MonetDB are given by the environmental variables `$POSTGRES_SRC` and `$MONETDB_SRC` respectively. In order to run any implementation of path consistency in MonetDB, the user should specify the option `--enable-rcc8` in the configure phase.

```
$ mkdir -p /tmp/monet-build
$ cd /tmp/monet-build
$ ${MONETDB_SRC}/configure --prefix=path/to/<installation_directory> \
    --enable-rcc8 --enable-optimize --disable-assert --disable-geom \
    --disable-java --disable-jdbc --disable-odbc --disable-testing \
    --disable-strict --disable-merocontrol
```

Then, to compile MonetDB:

```
$ make -j
$ make install
```

B.1 Instructions to run Renz's code in MonetDB

The source code of the Renz's implementation in MonetDB can be found under the directory `${MONETDB_SRC}/monetdb5/extras/rcc8/renz/`.

You can execute Renz's code using the MonetDB client, mclient, and specifying either the MAL or SQL language. Then you can load RCC-8 relations either from a regular file or from a SQL table.

For using the MAL language, execute the following command:

```
$ <installation_directory>/bin/mclient -l mal -d <database_name>
Welcome to mclient, the MonetDB interactive terminal (unreleased)
Type \q to quit, \? for a list of available commands
mal>
```

Then the RCC-8 relations can be loaded from a regular file:

```
mal> renz.pcfile("<filename>",0);
```

or from a SQL table:

```
4 #wiki.csp (http://en.wikipedia.org/wiki/Region_Connection_Calculus)
 0 1 ( DC )
 0 2 ( TPP NTPP )
 0 3 ( EC )
 0 4 ( DC EC )
 1 2 ( DC EC )
 2 4 ( DC EC )
 1 4 ( NTPP )
 1 3 ( EC )
.
```

Figure B.1: File `wiki.csp`

```
CREATE SCHEMA RCC8;
CREATE TABLE RCC8.R(u INTEGER,v INTEGER,r INTEGER);
INSERT INTO RCC8.R VALUES(0,17,32);
INSERT INTO RCC8.R VALUES(0,18,32);
INSERT INTO RCC8.R VALUES(0,19,32);
INSERT INTO RCC8.R VALUES(0,20,32);
INSERT INTO RCC8.R VALUES(0,21,32);
INSERT INTO RCC8.R VALUES(0,22,32);
INSERT INTO RCC8.R VALUES(0,23,32);
INSERT INTO RCC8.R VALUES(0,24,32);
...
```

Figure B.2: Creation of schema `RCC8` and table `R`

```
mal> renz.pcsql("<schema_name>","<table_name>",0);
```

For example, using file `${MONETDB_SRC}/monetdb5/extras/rcc8/renz/input/wiki.csp`:

```
mal> renz.pcfile("${MONETDB_SRC}/monetdb5/extras/rcc8/renz/input/wiki.csp",0);
Running path consistency with renz Logic and renz structures (bat rows 8)
Network is consistent. Execution time [16 msecs] , [2 loops, 72 compositions]
```

In Figure B.1 you can see the structure that the regular file should have. The number in the first line correnspouds to the maximum identifier number of the nodes of the input constraint network. Then, each line specifies the constraints (RCC-8 relations) holding between two nodes. Finally, the file ends with a new line starting with a dot character, i.e., '.'.

If you prefer to use a SQL table, you have to create a schema and a table, the names of which are required in the command mentioned previously, and insert the relations (Figure B.2).

Finally, using the SQL language requires similar steps:

```
$ <installation_directory>/bin/mclient -l sql -d <database_name>
Welcome to mclient, the MonetDB/SQL interactive terminal (unreleased)
Database: MonetDB v11.13.6 (unreleased), 'mapi:monetdb://teleios4:50000/test'
Type \q to quit, \? for a list of available commands
auto commit mode: on
sql>
```

```
-- Input constraint network (bit encoding)
-- Each integer corresponds to a set of RCC-8 relations
CREATE TABLE R (
        u int NOT NULL,
        v int NOT NULL,
        r int NOT NULL
);

CREATE INDEX R_u_idx ON R (u);
CREATE INDEX R_v_idx ON R (v);
CREATE INDEX R_uv_idx ON R (u, v);
```

Figure B.3: Script `01_create_R.sql`

Load the relations from a regular file:

```
sql> select sys.pcfile('<filename>',0);
```

or from a SQL table:

```
sql> select sys.pcsql('<schema_name>','<table_name>',0);
```

B.2 Instructions to run the SQL implementation in MonetDB

The source code of the SQL implementation in MonetDB can be found under the directory `${MONETDB_SRC}/monetdb5/extras/rcc8/sql/`.

For the initialization of an RCC-8 enabled database, it is required to execute the following scripts upon creation of a database with name `rcc8`.

1. Create a relational table for keeping the constraint network as described in Section 3.5.2, by executing the script `01_create_R.sql` which is shown in Figure B.3.

   ```
   $ mclient -d rcc8 < ${MONETDB_SRC}/monetdb5/extras/rcc8/sql/01_create_R.sql
   ```

2. Create two tables for keeping the composition and the inverse relations between two sets of RCC-8 relations and a table for keeping some statistics. The above tables are created and filled with the appropriate data by the scripts `02_create_rcc8-env.sql` and `03_composition-table_copy-into.sql` depicted in Figures B.4 and B.5 respectively.

   ```
   $ mclient -d rcc8 <\
       ${MONETDB_SRC}/monetdb5/extras/rcc8/sql/02_create_rcc8-env.sql
   $ mclient -d rcc8 <\
       ${MONETDB_SRC}/monetdb5/extras/rcc8/sql/03_composition-table_copy-into.sql
   ```

3. Load a sample constraint network by executing the script `wiki.sql` which is shown in Figure B.6.

   ```
   $ mclient -d rcc8 < ${MONETDB_SRC}/monetdb5/extras/rcc8/sql/input/wiki.sql
   ```

```
 -- Composition table (bit/int version)
CREATE TABLE "rcc8_trans" (
        s int NOT NULL,
        t int NOT NULL,
        comp int NOT NULL               -- composition of (s, t): s o t
);

-- Table with inverse relations (bit/int version)
CREATE TABLE "rcc8_inv" (
        r int NOT NULL,
        ri int NOT NULL
);

-- Table with the names of a set of RCC-8 relations
CREATE TABLE "rcc8_names" (
        r int NOT NULL,                 -- set of RCC-8 relations (bit encoding)
        name varchar(37) NOT NULL       -- corresponding string names
);

-- Table for keeping statistics when running the path consistency algorithm
CREATE TABLE "rcc8_stats" (
        iterations INT NOT NULL,
        delta INT NOT NULL,
        inputsz INT NOT NULL
);

INSERT INTO "rcc8_inv" VALUES
   (1,1),
   (2,2),
   (3,3),
   (4,4),
   (5,5),
   (6,6),
   (7,7),
   (8,32),
   (9,33),
     ...
```

Figure B.4: Script `02_create_rcc8-env.sql`

```
COPY 65025 RECORDS INTO rcc8_trans FROM stdin USING DELIMITERS '\t';
1       1       255
1       2       31
1       3       255
1       4       31
1       5       255
1       6       31
1       7       255
        ...
```

Figure B.5: Script `verb03_composition-table_copy-into.sql`

```
-- The following constraint network has been taken from
-- http://en.wikipedia.org/wiki/Region_Connection_Calculus
INSERT INTO R (u, v, r) VALUES
(0, 1, 1),
(0, 2, 24),
(0, 3, 2),
(0, 4, 3),
(1, 2, 3),
(2, 4, 3),
(1, 4, 16),
(1, 3, 2),
(3, 4, 255),
(3, 2, 255);

-- insert inverse edges
 INSERT INTO R (u, v, r) (SELECT R.v, R.u, S.ri FROM R JOIN rcc8_inv as S ON R.r
    = S.r);
```

Figure B.6: Script `wiki.sql`

4. Load the SQL script named `pc_monet.sql`, which implements the path consistency algorithm as described in Section 3.5.2.

   ```
   $ mclient -d rcc8 < ${MONETDB_SRC}/monetdb5/extras/rcc8/sql/pc_monet.sql
   ```

5. Run the path consistency algorithm on the relational table R as follows:

   ```
   $ echo "SELECT * FROM pc();" | mclient -d rcc8

   +------------+
   | edge_count |
   +============+
   |         20 |
   +------------+
   1 tuple
   ```

B.3 Instructions to run the SQL implementation in PostgreSQL

All the appropriate scripts that are needed for the SQL implementation in PostgreSQL, are under `${POSTGRES_SRC}` directory. The instructions of the PostgreSQL implementation are relevant to these of MonetDB described in the previous section.
For the initialization of an RCC-8 enabled database, it is required to execute the following scripts upon creation of a database with name `rcc8`.

1. Create a relational table for keeping the constraint network, by executing the script `01_create_R.sql`.

   ```
   $ psql -d rcc8 < ${POSTGRES_SRC}/01-create_R.sql
   ```

2. Create two tables for keeping the composition and the inverse relations between two sets of RCC-8 relations. The above tables are created and filled with the appropriate data by the script `02_create_rcc8-env.sql`.

```
$ psql -d rcc8 < ${POSTGRES_SRC}/02-create_rcc8-env.sql
```

3. Load a sample constraint network by executing the script `wiki.sql`.

```
$ psql -d rcc8 < ${POSTGRES_SRC}/input/wiki.sql
```

4. Load the SQL script named `pc_postgres.sql`, which implements the path consistency algorithm as described in Section 3.5.2

```
$ psql -d rcc8 < ${POSTGRES_SRC}/pc_postgres.sql
```

5. Run the path consistency algorithm on the relational table R as follows:

```
$ echo "SELECT * FROM pc();" | psql -d rcc8

NOTICE:  table "rcc8_stats" does not exist, skipping
CONTEXT:  SQL statement "DROP TABLE IF EXISTS rcc8_stats"
PL/pgSQL function "pc" line 14 at SQL statement
NOTICE:  CREATE TABLE will create implicit sequence "rcc8_stats_iteration_seq" for serial col
CONTEXT:  SQL statement "CREATE TABLE rcc8_stats (iteration SERIAL, d INT NOT NULL, inputsz I
PL/pgSQL function "pc" line 15 at SQL statement
NOTICE:  iteration 0: delta = 20 size = 20
NOTICE:  iteration 1: delta = 6 size = 20
NOTICE:  iteration 2: delta = 2 size = 20
 pc
----
 20
(1 row)
```

B.4 Instructions to run the implementation of the RCC-8 module in MonetDB

The source code of the implementation of the RCC-8 module can be found under the directory `${MONETDB_SRC}/monetdb5/extras/rcc8/`.
In order to run the RCC-8 module which is implemented in the kernel of MonetDB, the user should firstly create a database with name `rcc8` and then follow the steps 1-3 from Section B.2. Afterwards, the following command is required:

```
$ mclient -d rcc8 -lmal ${MONETDB_SRC}/monetdb5/extras/rcc8/Tests/run_rcc8pc.mal
RCC8pc: iteration 0     input-size=20         delta=20
Elapsed time 1 ms
#---------------------------------#
# h     t       t       t         # name
# void  int     int     int       # type
#---------------------------------#
[ 0@0,    0,      1,      1       ]
[ 1@0,    0,      2,      24      ]
[ 2@0,    0,      3,      2       ]
[ 3@0,    0,      4,      3       ]
[ 4@0,    1,      0,      1       ]
[ 5@0,    1,      2,      1       ]
[ 6@0,    1,      3,      2       ]
[ 7@0,    1,      4,      16      ]
[ 8@0,    2,      0,      96      ]
```

```
## load RCC-8 module
include rcc8;

## open sql
sql.init();
_mvc := sql.mvc();

## bind columns
u:bat[:oid,:int] := sql.bind(_mvc, "sys", "r", "u", 0);
v:bat[:oid,:int] := sql.bind(_mvc, "sys", "r", "v", 0);
r:bat[:oid,:int] := sql.bind(_mvc, "sys", "r", "r", 0);

# run path consistency
(ru:bat[:oid, :int], rv:bat[:oid, :int], rr:bat[:oid, :int]) := rcc8.pc(u, v, r);
```

Figure B.7: Script `run_rcc8pc.mal`

```
[ 9@0,    2,      1,      1       ]
[ 10@0,   2,      3,      38      ]
[ 11@0,   2,      4,      3       ]
[ 12@0,   3,      0,      2       ]
[ 13@0,   3,      1,      2       ]
[ 14@0,   3,      2,      14      ]
[ 15@0,   3,      4,      12      ]
[ 16@0,   4,      0,      3       ]
[ 17@0,   4,      1,      64      ]
[ 18@0,   4,      2,      3       ]
[ 19@0,   4,      3,      36      ]
```

The MAL program `run_rcc8pc.mal`, which is depicted in Figure B.7 loads module `rcc8` and then executes the path consistency algorithm.

B.5 Extensions to MonetDB

For the SQL implementation in MonetDB, which is described in Section 3.5.2, a new bitwise AND aggregate function has been implemented for integer values. This was accomplished by extending the existing `aggr` module of MonetDB in the spirit of sum/product aggregates. Except for integers, the implementation takes care of other MonetDB primitive datatypes, such as byte, short, word, and long, but it only handles input BATs of same type. The implementation of this aggregate functions spans over the following files:

- `${MONETDB_SRC}/monetdb5/modules/kernel/aggr_be_bitand.mx`
- `${MONETDB_SRC}/monetdb5/modules/kernel/aggr_bge_bitand.mx`
- `${MONETDB_SRC}/sql/backends/monet5/bit_and/90_bitand.sql`

The first two files implement an aggregate which computes the bitwise AND. The SQL file performs the connection of the aggregate function in SQL level with its implementation in C. One can invoke the bitwise AND aggregate function directly from SQL as `bit_and(·)`.

C. Efficient Algorithms for Checking the Consistency of Chordal RCC-8 Networks

Efficient Algorithms for Checking the Consistency of Chordal RCC-8 Networks

Michael Sioutis[1]*, Manolis Koubarakis[1], and Jean-François Condotta[2]

[1] National and Kapodistrian University of Athens, Greece
{sioutis,koubarak}@di.uoa.gr
[2] Artois University, Arras, France
jfrancois.condotta@univ-artois.fr

We consider chordal RCC-8 networks and show that we can check their consistency by enforcing partial path consistency with weak composition. We prove this by using the fact that RCC-8 networks with relations from the maximal tractable subsets $\hat{\mathcal{H}}_8, \mathcal{C}_8$, and $\mathcal{Q}_8$ of RCC-8 have the patchwork property. The use of partial path consistency has important practical consequences that we demonstrate with the implementation of a reasoner which we call PyRCC8$\triangledown$. This reasoner is developed by extending the state of the art reasoner PyRCC8, which we also present and compare experimentally with other reasoners for RCC-8 (Renz's solver, GQR, and the reasoner utilized in PelletSpatial). Given an RCC-8 network with only tractable RCC-8 relations, we show that it can be solved very efficiently with PyRCC8$\triangledown$ by making its underlying constraint graph chordal and running path consistency on this sparse graph instead of the completion of the given network. In the same way, partial path consistency can be used as the consistency checking step in backtracking algorithms for networks with arbitrary RCC-8 relations resulting in very improved pruning for sparse networks while incurring a penalty for dense networks.

1 Introduction

Spatial reasoning is a major field of study in Artificial Intelligence; particularly in Knowledge Representation. This field has gained a lot of attention during the last few years as it extends to a plethora of areas and domains that include, but are not limited to, ambient intelligence, dynamic GIS, cognitive robotics, and spatiotemporal design [CR08,BGWH11,Haz12]. In this context, an emphasis has been made on qualitative spatial reasoning which relies on qualitative abstractions of spatial aspects of the common-sense background knowledge, on which our human perspective on the physical reality is based. The concise expressiveness of the qualitative approach provides a promising framework that further boosts research and applications in the aforementioned areas and domains.

The Region Connection Calculus (RCC) is the dominant Artificial Intelligence approach for representing and reasoning about topological relations [RCC92]. RCC can be used to describe spatial regions that are non-empty regular subsets of some topological space by stating their topological relations to each other.

* Currently with Pierre and Marie Curie University, Paris, France.

Table 1: Definition of the various relations of RCC. Relations in bold are included in RCC-8

Relation	**Description**	**Definition**
$C(x, y)$	connects with	primitive relation
$\mathbf{DC}(x, y)$	disconnected	$\neg C(x, y)$
$P(x, y)$	part	$\forall z[C(z, x) \rightarrow C(z, y)]$
$PP(x, y)$	proper part	$P(x, y) \wedge \neg P(y, x)$
$\mathbf{EQ}(x, y)$	equals	$P(x, y) \wedge P(y, x)$
$O(x, y)$	overlaps	$\exists z[P(z, x) \wedge P(z, y)]$
$\mathbf{PO}(x, y)$	partially overlaps	$O(x, y) \wedge \neg P(x, y) \wedge \neg P(y, x)$
$DR(x, y)$	discrete	$\neg O(x, y)$
$\mathbf{TPP}(x, y)$	tangential proper part	$PP(x, y) \wedge \exists z[EC(z, x) \wedge EC(z, y)]$
$\mathbf{EC}(x, y)$	externally connected	$C(x, y) \wedge \neg O(x, y)$
$\mathbf{NTPP}(x, y)$	non-tangential proper part	$PP(x, y) \wedge \neg\exists z[EC(z, x) \wedge EC(z, y)]$
$Pi(x, y)$	part inverse	$P(y, x)$
$PPi(x, y)$	proper part inverse	$PP(y, x)$
$\mathbf{TPPi}(x, y)$	tangential proper part inverse	$TPP(y, x)$
$\mathbf{NTPPi}(x, y)$	non-tangential proper part inverse	$NTPP(y, x)$

RCC is based on a single primitive dyadic relation between spatial regions: relation "connects with", usually denoted by C [Cla81]. The intended topological interpretation of $C(a, b)$, where a and b are regions, is that a connects with b, if and only if their topological closures share a common point. This primitive can be used to define many predicates and functions which capture interesting and useful topological distinctions [Got94, Got96]. Table 1 shows the definitions of the RCC relations in terms of the basic relation C. RCC-8 is the constraint language formed by the following 8 binary topological relations of RCC: disconnected (DC), externally connected (EC), equal (EQ), partially overlapping (PO), tangential proper part (TPP), tangential proper part inverse (TPPi), non-tangential proper part (NTPP), and non-tangential proper part inverse (NTPPi). The RCC-8 relations are depicted in Figure 1.

The satisfiability problem in the RCC-8 framework is the reasoning problem of deciding consistency of a set of disjunctive spatial formula Θ, i.e., whether there is a spatial configuration where the relations between the regions is described by Θ. The reasoning problem in RCC-8 is $\mathcal{NP}$-hard in general [RN98]. However, there exist large maximal tractable subsets of RCC-8 which can be used to make reasoning much more efficient even in the general $\mathcal{NP}$-hard case. These maximal tractable subsets of RCC-8 are the sets $\hat{\mathcal{H}}_8, \mathcal{C}_8$, and $\mathcal{Q}_8$ [Ren99].

The state of the art qualitative spatial reasoners (QSRs) for RCC-8 [RN01, GWW08, SS09] implement efficient algorithms to decide whether a given set of RCC-8 relations between regions are consistent and infer new information from them. For these two problems, all well-known reasoners consider complete

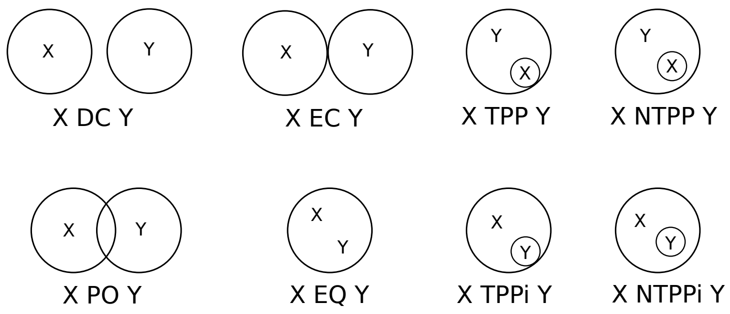


Fig. 1: Two dimensional examples for the eight base relations of RCC-8

graphs, and use some form of path consistency with weak composition[3] in the case that we have tractable RCC-8 networks, and backtracking-based algorithms for the general case. The task of inferring new information, in particular, can be viewed as solving the all-pairs shortest paths problem; the solution is then represented in a complete graph, which gives the inferred relations between all pairs of spatial regions. In this paper, we do not deal with this inference problem and concentrate only on the consistency checking problem of the input RCC-8 network by utilizing chordal graphs. We draw our motivation from the fact that chordal graphs generally contain far fewer edges than their complete counterparts. In addition, many RCC-8 networks are not densely connected[4] and, thus, we would like to attain decomposability that preserves their sparseness as much as possible instead of treating the network as a complete graph, which leads to a cubic complexity in all cases. This is made possible by enforcing partial path consistency which was originally introduced in [BSH99]. In summary, we make the following contributions:

– We consider RCC-8 networks with chordal constraint graphs and show that we can check their consistency by enforcing partial path consistency (PPC). PPC was originally introduced for finite domain CSPs in [BSH99] and it was most recently used in the case of Interval Algebra (IA) networks by [CC11]. PPC enforces path consistency on the constraint graph of a given constraint satisfaction problem (CSP). These two previous applications of chordality and partial path consistency consider convex finite-domain CSPs in [BSH99] and pre-convex Interval Algebra networks in [CC11]. We show that the same ideas can be applied to RCC-8 due to a recent result that shows that RCC-8 networks with relations from the maximal tractable subsets $\hat{\mathcal{H}}_8, \mathcal{C}_8$, and $\mathcal{Q}_8$ of RCC-8 [Ren99] have the patchwork property [Hua12].

[3] Some literature suggests the term *algebraic closure*, but this is equivalent to *path consistency* with *weak composition* (denoted by the symbol $\diamond$) [RL05], so we will use this more traditional term throughout the paper.

[4] The missing edges in an RCC-8 network implicitly represent universal relations.

– We demonstrate the practical applicability of our results with the implementation of a new reasoner, called PyRCC8$\bigtriangledown$. Given a network with only tractable RCC-8 relations, PyRCC8$\bigtriangledown$ can solve it very efficiently by making its underlying constraint graph chordal and running path consistency on this sparse graph instead of the completion of the given network. In the same way, it uses partial path consistency as a consistency checking step in backtracking algorithms for networks with arbitrary RCC-8 relations resulting in very improved pruning for sparse networks while incurring a penalty for dense networks.
– We make the case for a new generation of RCC-8 reasoners implemented in Python, a general-purpose, interpreted high-level programming language which enables rapid application development, and making use of advanced Python environments, such as PyPy[5], utilizing trace-based just-in-time (JIT) compilation techniques [BCFR09, BCF$^+$11]. We present such a reasoner, called PyRCC8, and compare it to other well-known reasoners from the literature [RN01, GWW08, SS09]. PyRCC8 serves as the basis for PyRCC8$\bigtriangledown$, and both reasoners are compared experimentally with each other.

The organization of this paper is as follows. Section 2 introduces the main notions and terminology about constraint networks, various notions of consistency, and discusses weak composition and algebraic closure. Section 3 introduces PPC and applies it to chordal RCC-8 networks. Section 4 introduces PyRCC8, our implementation of a state of the art reasoner for RCC-8. In Section 5 we evaluate the reasoner PyRCC8$\bigtriangledown$ experimentally. Finally, in Section 6 we give a brief overview of related work, and in Section 7 we conclude and give directions for future research.

2 Preliminaries

In this section we define the concepts of constraint networks and their corresponding constraint graphs, relation algebra, composition, weak composition, algebraic closure (a-closure), and various notions of local consistency. More details can be found in [RN07].

2.1 Constraint Networks and Relation Algebra

Knowledge about entities or about the relationships between entities is often given in the form of *constraints*. Given a set of m variables $\mathcal{V} = \{x_1, \ldots, x_m\}$ over a domain $\mathcal{D}$, an *n-ary constraint* consists of an n-ary relation $R_i \subseteq \mathcal{D}^n$ and an n-tuple of variables $\langle x_{i_1}, \ldots, x_{i_n} \rangle$, written $R_i(x_{i_1}, \ldots, x_{i_n})$. For binary constraints, we will also use the infix notation $x_1 R_i x_2$. A (partial) *instantiation* f of variables to values is a (partial) function from the set of variables $\mathcal{V}$ to the set of values $\mathcal{D}$. We say that an instantiation f *satisfies the constraint* $R_i(x_{i_1}, \ldots, x_{i_n})$ if and only if $< f(x_{i_1}), \ldots, f(x_{i_n}) > \in R_i$.

[5] `http://pypy.org/`

A CSP consists of a set of variables $\mathcal{V}$ over a domain $\mathcal{D}$ and a set of constraints Θ. A *solution* of a CSP is an instantiation which satisfies all the constraints in Θ. A CSP is *consistent* if it has a solution. In this paper we restrict ourselves to binary CSPs, i.e., CSPs where only binary constraints are used. A binary CSP can be represented by a *constraint network* which is a labelled digraph where each node is labelled by a variable x_i (or simply by the variable index i) and each directed edge is labelled by a binary relation. We will use the notation R_{ij} to denote the relation constraining the variable pair $\langle x_i, x_j \rangle$. A *constraint graph* for such a problem is an undirected graph where the nodes represent variables and two nodes are linked to represent the *existence* of a constraint which involves these variables. At most a single edge will join two nodes, even if they are related by more than one constraint. By overloading notation, we also use R_{ij} to denote the constraint $R_{ij}(x_i, x_j)$ itself. If the domain of the variables is finite, CSPs can be solved by *backtracking* over the ordered domains of the single variables. If the domain of the variables is infinite, as is the case with RCC-8, backtracking over the domain is not possible and other methods have to be applied. Infinite domains is a central difference of spatial or temporal CSPs to normal CSPs. For instance, there are infinitely many time points or temporal intervals on the time line and infinitely many regions in a two or three dimensional space.

One way of dealing with infinite domains is using constraints over a finite set of binary relations, by employing a *relation algebra* [LM94]. A *relation algebra* consists of a set of binary relations which is closed under several operations on relations and contains some particular relations. The operations are union ($\cup$), intersection ($\cap$), composition ($\circ$), complement ($\bar{\ }$), and conversion ($\breve{\ }$), where conversion is defined as $\breve{R} \overset{def}{=} \{\langle x, y \rangle \mid \langle y, x \rangle \in R\}$ and composition is defined as $R \circ S \overset{def}{=} \{\langle x, y \rangle \mid \exists z : \langle x, z \rangle \in R \wedge \langle z, y \rangle \in S\}$. The particular binary relations mentioned above are the empty relation $\emptyset$ which does not contain any pair, the universal relation $*$ which contains all possible pairs, and the identity relation Id which contains all pairs of identical elements in a set of constraints. We assume that a set of constraints Θ contains one constraint for each pair of variables involved in Θ, i.e., if no information is given about the relation holding between two variables x_i and x_j, then the universal relation $*$ constrains the pair, i.e., $R_{ij} = *$. Another assumption that we make is that whenever a constraint R_{ij} between x_i and x_j is in Θ, the converse relation constrains x_j and x_i , i.e., $\breve{R}_{ij} = R_{ji}$.

Of particular interest are relation algebras that are based on finite sets of *jointly exhaustive and pairwise disjoint* (JEPD) relations. JEPD relations are sometimes called atomic, basic, or base relations. We refer to them as base relations. RCC-8 is the set of base relations of RCC. Since any two entities are related by exactly one of the base relations, they can be used to represent definite knowledge with respect to the given level of granularity. Indefinite knowledge can be specified by unions of possible base relations. For these relation algebras, the universal relation is the union over all base relations. Converse, complement, intersection and union of relations can easily be obtained by performing the corresponding set theoretic operations. Composition of base relations has to be

Table 2: Composition of RCC-8 relations

$\diamond$	**DC**	**EC**	**PO**	**TPP**	**NTPP**	**TPPi**	**NTPPi**	**EQ**
DC	*	DC,EC PO,TPP NTPP	DC,EC PO,TPP NTPP	DC,EC PO,TPP NTPP	DC,EC PO,TPP NTPP	DC	DC	DC
EC	DC,EC PO,TPPi NTPPi	DC,EC PO,TPP TPPi,EQ	DC,EC PO,TPP NTPP	EC,PO TPP NTPP	PO,TPP NTPP	DC,EC	DC	EC
PO	DC,EC PO,TPPi NTPPi	DC,EC PO,TPPi NTPPi	*	PO,TPP NTPP	PO,TPP NTPP	DC,EC PO,TPPi NTPPi	DC,EC PO,TPPi NTPPi	PO
TPP	DC	DC,EC	DC,EC PO,TPP NTPP	TPP NTPP	NTPP	DC,EC PO,TPP TPPi,EQ	DC,EC PO,TPPi NTPPi	TPP
NTPP	DC	DC	DC,EC PO,TPP NTPP	NTPP	NTPP	DC,EC PO,TPP NTPP	*	NTPP
TPPi	DC,EC PO,TPPi NTPPi	EC,PO TPPi NTPPi	PO,TPPi NTPPi	PO,EQ TPP TPPi	PO,TPP NTPP	TPPi NTPPi	NTPPi	TPPi
NTPPi	DC,EC PO TPPi NTPPi	PO,TPPi NTPPi	PO,TPPi NTPPi	PO,TPPi NTPPi	PO,TPP NTPP NTPPi TPPi,EQ	NTPPi	NTPPi	NTPPi
EQ	DC	EC	PO	TPP	NTPP	TPPi	NTPPi	EQ

computed using the semantics of the relations. Composition of unions of base relations can be obtained by computing only the union of the composition of the base relations. Usually, compositions of the base relations are pre-computed and stored in a *composition table*. The composition table for RCC-8 is shown in Table 2.

2.2 Composition and Weak Composition

According to the definition of composition, we have to look at an infinite number of tuples in order to compute composition of base relations, which is clearly not feasible. Fortunately, many domains such as points or intervals on a time line are ordered or otherwise well-structured domains and composition can be computed using the semantics of the relations. However, for domains such as arbitrary spatial regions that are not well structured and where there is no common representation for the entities we consider, computing the true composition is not feasible and composition has to be approximated by using *weak composition* [RL05]. Further, there exist compositions that cannot be expressed by disjunctions of RCC-8 base relations [LY03, RL05]. Weak composition ($\diamond$) of two relations S and T for a set of base relations $\mathcal{B}$ is defined as the strongest relation $R \in 2^{\mathcal{B}}$ which contains $S \circ T$, or formally, $S \diamond T \overset{def}{=} \{R_i \in \mathcal{B} \mid R_i \cap (S \circ T) \neq \emptyset\}$. The advantage of weak composition is that we stay within the given set of relations $R = 2^{\mathcal{B}}$ while applying the algebraic operators, as R is by definition closed under weak composition, union, intersection, and converse. The composition operation for RCC-8 captured by Table 2 is actually weak composition [RL05].

2.3 Path Consistency and Algebraic Closure

Because of the high complexity of deciding consistency and computing relations entailed by a given CSP, different forms of local consistency and algorithms for achieving local consistency have been introduced in the literature. When solving a CSP, local consistency is used to prune the search space by eliminating local inconsistencies. In some cases local consistency is even enough for deciding consistency. The path consistency algorithm is one of the central methods to solve constraint networks in qualitative spatial and temporal calculi. Path consistency is implemented by introducing rules defining compositions and intersections of supported relations. Path consistency propagates constraints in a network by iteratively performing the following operation until a fixed point $\overline{R}$ is reached:

$$\forall i, j, k \; R_{ij} \leftarrow R_{ij} \cap (R_{ik} \circ R_{kj})$$

If $R_{ij} = \emptyset$ for a pair (i, j) then R is inconsistent, otherwise $\overline{R}$ is *path consistent.*

When weak composition differs from composition, we cannot apply the path consistency algorithm as it requires composition and not just weak composition. We can, however, replace the composition operator in the path consistency algorithm with the weak composition operator. The resulting algorithm is called the algebraic closure or *a-closure* algorithm [LR04]. In the rest of the paper, we prefer the equivalent term *path consistency with weak composition* which we shorten to $\triangledown$*-path consistency* when enforced on chordal constraint graphs. Similarly, the backtracking algorithm that uses $\triangledown$-path consistency as a subroutine to solve the consistency problem for RCC8 networks (Section 3.3) is called $\triangledown$-Consistency and the reasoner which implements all these algorithms is called PyRCC8$\triangledown$.

Similarly to the PPC algorithm defined in [BSH99] which utilizes classical path consistency [Mon74], $\triangledown$-path consistency considers only triangles of nodes in the chordal graph corresponding to a constraint network, and enforces partial path consistency on these triangles. We will use the terms partial path consistency and $\triangledown$-path consistency interchangeably throughout this paper, since in our case we enforce partial path consistency on the *chordal* constraint graph of a given RCC-8 network.[6]

This completes our presentation of preliminaries. We now proceed with the main results of the paper.

3 Solving Chordal RCC-8 Networks

We now start presenting the main contribution of this paper: how to solve chordal RCC-8 networks efficiently and what impact this special case can have to solving general RCC-8 networks.

[6] We remind the reader that PPC enforces path consistency on the constraint graph of a given CSP, which could be any graph in the general case.

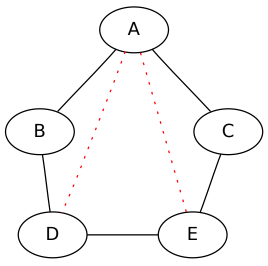


Fig. 2: Example of a chordal graph

We start by introducing chordal graphs and networks, and partial path consistency. The state of the art reasoners, including PyRCC8, consider complete graphs when checking the consistency of an input network. The techniques of this section will show how to make this task more efficient using $\triangledown$-path consistency if the underlying constraint graphs are chordal or by turning them into chordal if they are not.

3.1 Chordal Graphs and Graph Triangulation

In this section we list some definitions and theorems from graph theory that are essential in understanding the discussion to follow, and the new algorithms to be presented in Section 3.3. The interested reader may find more results regarding chordal graphs, and graph theory in general, in [Gol04].

Definition 1 *Let $G = (V, E)$ be an undirected graph. The following are well-known concepts from graph theory:*

- *The* neighborhood of a vertex $v \in V$ *is* $N(v) = \{v' \neq v \mid (v, v') \in E\}$*. The* neighborhood of a set of vertices S *is* $N(S) = \bigcup_{s \in S} N(s) \setminus S$*.*
- *If $S \subset V$ is a set of vertices of G, the* subgraph induced *by S, denoted by $G(S)$, is the subgraph that has exactly the edges that appear in G over the vertex set S.*
- *The* vertex set *of G is denoted by $V(G)$.*
- *A subset of vertices $S \subseteq V$ is a* minimal separator *iff $G(V \setminus S)$ has at least two connected components C_1 and C_2 such that $N(V(C_1)) = N(V(C_2)) = S$.*
- *If $(v_1, v_2, \ldots, v_k, v_{k+1} = v_1)$ with $k > 3$ is a cycle, then any edge on two nonadjacent vertices v_i, v_j with $1 < j - i < k - 1$ is a* chord *of this cycle.*
- *G is* chordal *or* triangulated *if every cycle of length greater than 3 has a chord.*
- *A* clique *is a set of vertices which are pairwise adjacent. A clique is* maximal *if it is not a subset of a larger clique.*

- *A vertex $v \in V$ is* simplicial *if the set of its neighbors $N(v)$ induces a clique, i.e., $\forall s, t \in V$, if $s, t \in N(v)$ then $(s, t) \in E$.*
- *Let $d = (v_n, \ldots, v_1)$ be an ordering of V. Also, let G_i denote the subgraph of G induced by $V_i = \{v_1, \ldots, v_i\}$; note that $G_n = G$. The ordering d is a* perfect elimination ordering *of G if every vertex v_i with $n \geq i \geq 1$ is a simplicial vertex of the graph G_i.*

The following theorem gives some well-known properties of chordal graphs [Gol04].

Theorem 1 *Let $G = (V, E)$ be an undirected chordal graph. The following statements are equivalent and characterize G :*

- *G admits a perfect elimination ordering.*
- *Every minimal separator of G is a clique.*
- *There exists a tree T, called a* clique tree *of G, whose vertex set is the set of maximal cliques of G and whose edge set is the set of minimal separators of G.*

The graph shown in Figure 2 consists of a cycle which is formed by the black solid edges and two chords that correspond to the dotted edges. As for this part, the graph is chordal. However, removing one dotted edge would result in a non-chordal graph. Indeed, the other dotted edge with three black edges would form a cycle of length four with no chords. Chordality checking can be done in linear time, since testing whether an ordering is a perfect elimination ordering can be performed in linear time with the lexicographic breadth-first search algorithm or the maximum cardinality search algorithm [BBH02]. Both algorithms have time complexity $O(|V| + |E|)$ for a given graph $G = (V, E)$.

If a graph is not chordal, it can be made so by the addition of a set of new edges, called *fill edges*. This process is usually called *triangulation* of the given graph. The fill edges can be found by eliminating the vertices one by one and connecting all vertices in the neighborhood of each eliminated vertex, thus making it simplicial in the elimination graph. This process constructs a perfect elimination ordering as a byproduct. If the graph was already chordal, following its perfect elimination ordering during triangulation has the result that no fill edges are added. A perfect elimination ordering for the chordal graph shown in Figure 2 would be the ordering (B, D, E, C, A) of its set of vertices.

In general, it is desirable to achieve chordality with as few fill edges as possible. However, obtaining an optimum graph triangulation is known to be NP-hard [BBH02]. For this purpose several advanced heuristic algorithms have been developed to aid the approximation of a good triangulation [CM94]. In our implementation of PyRCC8$\bigtriangledown$ discussed in Section 3.3 below, we use the simple *minimum degree* heuristic to obtain an elimination ordering of the set of vertices V. The minimum degree heuristic, whenever applied, chooses the vertex with the smallest number of neighbors which consequently produces a clique of minimal size.

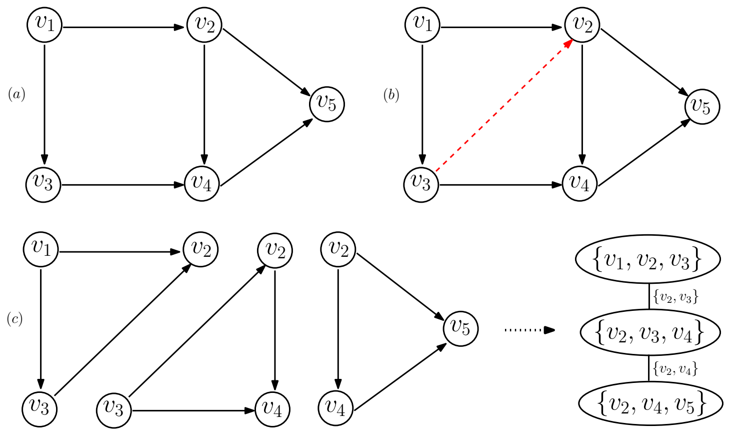


Fig. 3: Triangulation and decomposition of a network

In the rest of the paper, the concepts of chordal graphs introduced above will be applied to the constraint graph of a given RCC-8 network. If such a graph is not chordal, it will be made into one by introducing fill edges that correspond to the universal relation. In what follows, we will often refer to universal relations as trivial constraints, since they basically allow every RCC-8 configuration between two spatial regions.

3.2 $\triangledown$-Path Consistency and Patchwork

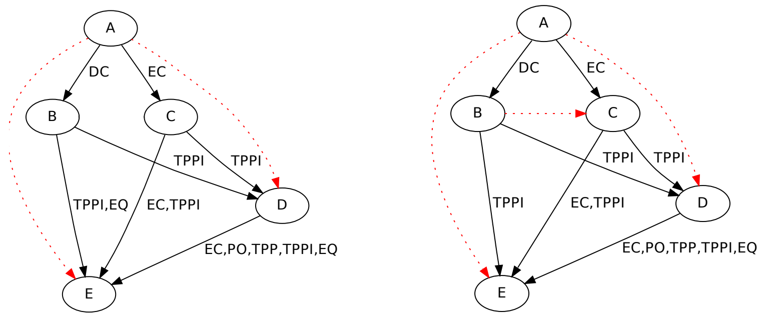
We now show that $\triangledown$-path consistency is sufficient to decide the consistency of a network with relations from the maximal tractables subsets $\hat{\mathcal{H}}_8, \mathcal{C}_8$, and $\mathcal{Q}_8$ of RCC-8. The proof of our result makes use of the patchwork property of RCC-8 networks originally defined in [LM07] and more recently used in [Hua12]. This property, which we define below, allows to "patch" a number of satisfiable RCC-8 networks into a bigger network (their union) which is also satisfiable, assuming that the networks "agree" on their common part.

Let C be a constraint network from a given CSP. We will use $\mathcal{V}_C$ to refer to the set of variables of C. If $\mathcal{V}$ is any set of variables, $C_\mathcal{V}$ will be the constraint network that results from C by keeping only the constraints which involve variables of $\mathcal{V}$.

Definition 2 *We will say that a CSP has the* patchwork property *if for any finite satisfiable constraint networks C and C' of the CSP such that $C_{\mathcal{V}_C \cap \mathcal{V}_{C'}} = C'_{\mathcal{V}_C \cap \mathcal{V}_{C'}}$, the constraint network $C \cup C'$ is satisfiable [Hua12].*

Proposition 1 *The three CSPs for path consistent $\hat{\mathcal{H}}_8, \mathcal{C}_8$, and $\mathcal{Q}_8$ networks, respectively, all have patchwork [Hua12].*

Proposition 2 *Let C be an RCC-8 constraint network with relations from $\hat{\mathcal{H}}_8, \mathcal{C}_8$, and $\mathcal{Q}_8$ on its edges. Let G be the chordal graph that results from triangulating*



(a) Chordal network (b) Complete network

Fig. 4: Pruning capacity of PC on a chordal constraint network and its completion

the associated constraint graph of C, and T a clique tree of G. Let C' denote the constraint network corresponding to G (C' is C plus some universal relations corresponding to fill edges). C is consistent if all the networks corresponding to the nodes of T are path consistent.

Proof. Let $T = (V, E)$, where $V = \{V_1, V_2, \ldots, V_n\}$ is the set of all maximal cliques of G. We enforce path consistency on C'. For every complete subgraph G_i of G induced by V_i, path consistency can decide the satisfiability of the corresponding subnetwork C'_i of C', because we have relations from the maximal tractable subsets $\hat{\mathcal{H}}_8, \mathcal{C}_8$, and $\mathcal{Q}_8$ of RCC-8. If all C'_i's are satisfiable then for any two networks C'_j and C'_k with $1 \leq j < k \leq n$ we have that $(C'_j)_{\mathcal{V}_{C'_j} \cap \mathcal{V}_{C'_k}} = (C'_k)_{\mathcal{V}_{C'_j} \cap \mathcal{V}_{C'_k}}$. Thus, it follows from Proposition 1 that $\bigcup_{i=1}^{n} C'_i = C'$ is satisfiable. Since C' is at least as restrictive as C, C is satisfiable.

An example of Proposition 2 is shown in Figure 3. The constraint graph of the initial network C is shown in (a). Then, we triangulate this graph by adding edge (v_2, v_3) that is depicted as a dotted line in (b). Finally, we obtain a decomposition of this graph that consists of maximal cliques $\{v_1, v_2, v_3\}$, $\{v_2, v_3, v_4\}$, and $\{v_2, v_4, v_5\}$, and is separated by minimal separators $\{v_2, v_3\}$ and $\{v_2, v_4\}$ (as viewed in the clique tree in (c)). Path consistency can be enforced upon the corresponding subnetworks, and given that they are satisfiable, they can be "patched" back together into a satisfiable network. This process can be viewed as an appliance of reverse amalgamation. In this example, by enforcing path consistency on the chordal constraint network we consider only 3 triangles of relations. If we had opted to complete the constraint network depicted in Figure 3 and obtain the corresponding complete constraint graph of order 5, enforcing path consistency would result in considering a total of 10 triangles of relations.

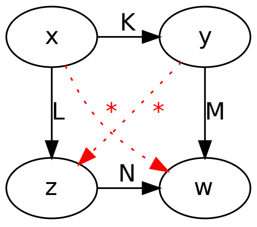
In general, the number of cycles of length 3, viz. triangles, in a complete graph G of order n, denoted by $c_3(G)$, is given by the following mathematical expression:

$$c_3(G) = n!/(6(n-3)!)$$

Given the results in this section the question arises whether more pruning can be obtained by completing a chordal graph. In Figure 4 we present an example of the pruning capacity of path consistency on a chordal constraint network and its completion. The chordal constraint network in Figure 4a is path consistent with respect to its corresponding chordal graph. However, the addition of edge (B, C) that completes the network, shown in Figure 4b, results in the pruning of base relation EQ on edge (B, E). Thus, in the case of RCC-8 we cannot have a result similar to the one by Bliek and Sam-Haroud that consider convex CSPs and show that the pruning capacity of path consistency on the chordal graph is equivalent to the pruning capacity of path consistency on the completed network when we consider the common edges [BSH99].

Let us now present a simple example that considers an RCC-8 network consisting of four RCC-8 relations that together form a cycle of length 4. We triangulate the constraint graph corresponding to this network in two different ways, one for each of the two diagonals of the graph serving as a chord. We enforce path consistency on the obtained constraint graph for each of the two cases. This gives rise to the following interesting result:

Corollary 1 *Given the RCC-8 network shown below, suppose that K, L, M, and N are RCC-8 relations from the maximal tractable subsets $\hat{\mathcal{H}}_8, \mathcal{C}_8$, and $\mathcal{Q}_8$ of RCC-8, and suppose that diagonals are the* universal *relation.*



Then, the following holds:

- *If the initial RCC-8 network is inconsistent, then using any of the two diagonals as a chord to triangulate the constraint graph and enforcing path consistency on the obtained chordal graph, results in the assignment of the empty relation $\emptyset$ to the corresponding diagonal edge.*

The above result is a direct consequence of Proposition 2 as it demonstrates how fill edges that correspond to universal relations contribute to the consistency checking and inferring process in RCC-8 reasoning with chordal graphs.

```
$\triangledown$-Path-Consistency(C, G)
Input: A constraint network C and its chordal graph G
Output: True if constraint network C is path consistent, and False otherwise

1:  Q ← {(i, j) | (i, j) ∈ E } // Initialize the queue
2:  while Q is not empty do
3:          select and delete an (i, j) from Q
4:          foreach k such that (i, k), (k, j) ∈ E do
5:                  t ← $C_{ik} \cap (C_{ij} \diamond C_{jk})$
6:                  if t ≠ $C_{ik}$ then
7:                     if t = ∅ then return False
8:                     $C_{ik}$ ← t
9:                     $C_{ki}$ ← $\breve{t}$
10:                    Q ← Q ∪ {(i, k)}
11:                 t ← $C_{kj} \cap (C_{ki} \diamond C_{ij})$
12:                 if t ≠ $C_{kj}$ then
13:                    if t = ∅ then return False
14:                    $C_{kj}$ ← t
15:                    $C_{jk}$ ← $\breve{t}$
16:                    Q ← Q ∪ {(k, j)}
17: return True
```

Fig. 5: $\triangledown$-Path-Consistency Algorithm

We now present the implementation of a reasoner that makes use of chordal graphs which we call PyRCC8$\triangledown$.

3.3 $\triangledown$-Path Consistency and $\triangledown$-Consistency

In this section, we give an algorithm to decide the consistency problem of a RCC-8 network by using maximal tractable subsets $\hat{\mathcal{H}}_8, \mathcal{C}_8$, and $\mathcal{Q}_8$ of RCC-8 and consistency checking restricted to triangles of the chordal network. This algorithm will be presented in the context of our PyRCC8$\triangledown$ reasoner.

First, we consider function $\triangledown$-path consistency, shown in Figure 5, which takes as parameters a network C, and its corresponding chordal constraint graph $G = (V, E)$ which is obtained by triangulating the constraint graph corresponding to C. The objective of $\triangledown$-path consistency is to enforce path consistency to all triangles of relations in G. Notice that this will result in path consistent complete networks that correspond to the nodes of a clique tree T of G. Thus, it follows from Proposition 2 that $\triangledown$-path consistency decides the consistency of network C, assuming that C contains relations from maximal tractable subsets $\hat{\mathcal{H}}_8, \mathcal{C}_8$, and $\mathcal{Q}_8$ of RCC-8. If δ denotes the maximum degree of a vertex of G, we have that for each arc (i, j) selected at line 3, there are at most δ vertices of G corresponding to index k such that v_i, v_j, v_k forms a triangle. Additionaly, there

```
$\triangledown$-**Consistency**(C, G)
Input: A constraint network C and its chordal graph G
Output: A refined constraint network C' if C is satisfiable, and None otherwise

 1: if not $\triangledown$-Path-Consistency(C, G) then
 2:     return None
 3: if no constraint can be split then
 4:     return C
 5: else
 6:     choose unprocessed constraint $x_i R x_j$ ;
        split $R$ into $S_1, ..., S_k \in S$: $S_1 \cup ... \cup S_k = R$
 7:     Values $\leftarrow \{S_l \mid 1 \leq l \leq k\}$
 8: foreach $V$ in Values do
 9:             replace $x_i R x_j$ with $x_i V x_j$ in C
10:             result = $\triangledown$-Consistency(C, G)
11:             if result $\neq$ None then
12:                 return result
13: return None
```

Fig. 6: Recursive $\triangledown$-Consistency Algorithm

exist $|E|$ arcs in the network and one can remove at most $|\mathcal{B}|$ values from any relation that corresponds to an arc, where $\mathcal{B}$ refers to the set of base relations of RCC-8. As a result, the time and space complexity of $\triangledown$-path consistency is $O(\delta \cdot |E| \cdot |\mathcal{B}|)$ and $O(|E|)$ respectively.

For the general case of RCC-8 networks, we have a backtracking algorithm, called $\triangledown$-Consistency, which is presented in Figure 6. The algorithm splits a relation R into relations that belong to some tractable set of relations S (line 6). Then, each of these relations is instantiated accordingly to the constraint network C (line 9) and the $\triangledown$-path consistency algorithm is reapplied. Notice, however that, except for the first step, the $\triangledown$-path consistency algorithm only has to be run for the paths that are possibly affected by each prior instantiation, which takes $\Theta(\delta \cdot |\mathcal{B}|)$ intersections and compositions. This detail is not included in Figure 6.

PyRCC8$\triangledown$ also offers an additional algorithm, which is the iterative counterpart of the recursive chronological backtracking algorithm. The iterative algorithm is shown in Figure 7. The recursive and iterative algorithms are functionally equivalent. However, the iterative algorithm can not suffer from a recursion depth or a recursion stack limit, which is the case for recursion in many languages, such as C, C++, Python, and Java.

```
**▽-Consistency**(C, G)
Input: A constraint network C and its chordal graph G
Output: A refined constraint network C' if C is satisfiable, and None otherwise

 1: Stack ← {} // Initialize stack
 2: if not ▽-Path-Consistency(C, G) then
 3:     return None
 4: while 1 do
 5:     if no constraint can be split then
 6:         return C
 7:     else
 8:         choose unprocessed constraint $x_i R x_j$ ;
            split $R$ into $S_1, ..., S_k \in S$: $S_1 \cup ... \cup S_k = R$
 9:         Values ← {$S_l$ | 1 ≤ l ≤ k}
10:     while 1 do
11:         if not Values then
12:             while Stack do
13:                 C, Values = Stack.pop()
14:                 if Values then
15:                     break
16:             else
17:                 return None
18:         $V$ = Values.pop()
19:         replace $x_i R x_j$ with $x_i V x_j$ in C
20:         if ▽-Path-Consistency(C, G) then
21:             break
22:     Stack.push(C, Values)
23: raise RuntimeError, Can't happen
```

Fig. 7: Iterative ▽-Consistency Algorithm

4 PyRCC8 - A state of the art reasoner for RCC-8

In this section we present PyRCC8[7], an open source, efficient reasoner for RCC-8 written in Python, that has served as the basis for the reasoner for chordal RCC-8 networks that we discussed in Section 3. PyRCC8 is implemented using PyPy, a fast, compliant implementation of the Python 2 language. To the best of our knowledge, PyRCC8 is the first implementation of a RCC-8 reasoner on top of a trace-based JIT compiler. Previous implementations have used either static compilers, e.g., Renz's solver [RN01] and GQR [GWW08], or method-based JIT compilers, e.g., the RCC-8 reasoning module of the description logic reasoner PelletSpatial [SS09]. The advantage of trace-based JIT compilers is that they can discover optimization opportunities in common dynamic execution paths,

[7] `http://pypi.python.org/pypi/PyRCC8/`

that are not apparent to a static compiler or a method-based JIT compiler [BCFR09, BCF^{+}11].

PyRCC8 offers a path consistency algorithm for solving tractable RCC-8 networks and a backtracking-based algorithm for general networks. Both algorithms draw on the original ideas of [RN01], but offer some more interesting features that we discuss below.

The path consistency algorithm processes arcs in a strict FIFO manner. This functionality is based on the implementation of a *hybrid queue* data structure that comprises a double-ended queue and a set. This allows *pushing*, *popping*, and *membership checking* of an arc to be achieved in $O(1)$ time.

The path consistency algorithm can also handle weighted arcs according to their restrictiveness. Most restrictive arcs are processed first because they restrict other arcs more and, thus, render them less prone to be processed again. Two weighting schemes are being used: (i) exact weighting of arcs [RN01], and (ii) approximate weighting of arcs, using the approach by Van Beek and Manchak [vBM96]. This functionality is based on the implementation of a *priority queue* data structure that comprises a heap and a hash map. This allows *pushing*, and *popping* of an arc to be achieved in $O(log(n))$ time, and *membership checking* of an arc to be achieved in $O(1)$ time.

Both our hybrid queue and priority queue data structure implementations are more advanced than the queue data structures found in Renz's solver, GQR, and PelletSpatial (e.g., Renz's solver uses a $n \times n$ matrix as a queue). Furthermore, our path consistency algorithm processes only meaningful arcs, i.e., arcs that do not correspond to the universal relation[8], thus, doing also fewer consistency checks.

Regarding concistency of general RCC-8 networks, PyRCC8 is the only reasoner we know that offers an iterative counterpart of the recursive backtracing algorithm. Additionaly, PyRCC8 precomputes the converse of relations to avoid time consuming and exhaustive repetition of converse computations for large datasets.

Finally, regarding heuristics, the classic weight and cardinality heuristics [vBM96] are implemented and used in the selection of sub-relations and the ordering of variables. PyRCC8 offers static and dynamic reasoning as in [RN01] and also considers heuristic criteria based on the scope (local or global) of constraints.

4.1 Comparing PyRCC8 to other reasoners

Since PyRCC8 is a new reasoner, we compared its performance with that of Renz's solver [RN01], GQR [GWW08] release 1418, and PelletSpatial [SS09] version 0.1, with their best performing heuristics enabled. At this point we should mention that PelletSpatial has no backtracking algorithm for general RCC-8 networks, and offers only a path consistency algorithm. Additionaly, it receives

[8] The result of the composition of any relation with the universal relation is the universal relation.

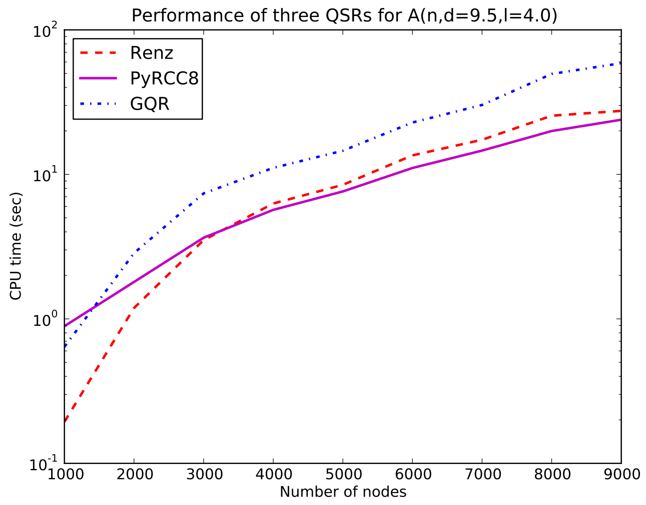


Fig. 8: Comparison of different PC algorithms

as input spatial relations expressed in RDF/XML syntax of OWL2. Since PelletSpatial proved to be highly inefficient and could not meet our expectations, we left it out in most of our experimental comparisons.

The experiments were carried out on a computer with an Intel Xeon 4 Core X3220 processor with a CPU frequency of 2.40 GHz, 8 GB RAM, and the Debian Lenny x86_64 OS. Renz's solver and GQR were compiled with gcc/g++ 4.4.3. PelletSpatial was run with OpenJDK 6 build 19, which implements Java SE 6. PyRCC8 was run with PyPy 1.8, which implements Python 2.7.2. Only one of the CPU cores was used for the experiments.

Path Consistency. The performance of the path consistency algorithm is crucial for the overall performance of a qualitative reasoner, since path consistency can be used to solve tractable networks, can be run as a preprocessing step, and as the consistency checking step of any backtracking algorithm. For our first experiment, we compared PyRCC8's path consistency implementation to that of Renz's solver and GQR. We considered network sizes between 1000 and 9000 nodes. For each size, 30 networks were generated using all RCC-8 relations with Renz's random instance generator [RN01]. Additionaly, networks were generated with an average degree (the number of non-universal constraints involving a node in average) of 9.5, because this degree belongs to the phase transition of RCC-8 relations for randomly generated RCC-8 networks [RN01], and, hence, guarantees hard and more time consuming, in terms of solubility, instances for the path consistency algorithm. The results of this experiment are shown in Figure 8. The corresponding graph shows that PyRCC8 outperforms GQR and Renz's solver for path consistency checking. In fact, as constraint networks grow

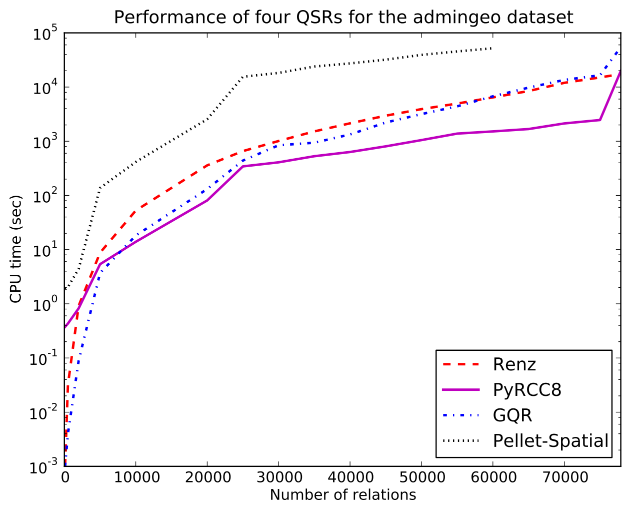


Fig. 9: Comparison of different PC algorithms using the admingeo dataset

larger, PyRCC8 becomes about 3 times faster than GQR and steadily opens the gap with Renz's solver.

For our second experiment we compared PyRCC8's path consistency implementation to that of Renz's solver, GQR, and PelletSpatial, using a dataset which encodes the administrative geography of Great Britain using RCC-8 [GDH08]. We call this dataset *admingeo* in what follows. The admingeo dataset is a real dataset published by Ordnance Survey and consists of RCC-8 base relations between geographic entities in Great Britain.[9] Since the data is encoded in RDF/XML syntax of OWL2, we had to translate it to match the input format of PyRCC8, Renz's solver, and GQR. The admingeo dataset is very large and very sparse, posing a challenge of scalability to any path consistency algorithm implementation. A nice visualisation of the admingeo dataset where its sparseness becomes apparent is depicted in Figure 10.[10] The admingeo dataset comprises a consistent constraint network of over 10000 nodes and nearly 80000 relations. For our experiment we created constraint networks of different size, by taking into account a minimum number of relations from the initial dataset and increasing it at every next step. Of course, the whole dataset was used as a final step. The results of the path consistency experiment using the admingeo dataset

[9] `http://www.ordnancesurvey.co.uk/ontology/AdministrativeGeography/v2.0/AdministrativeGeography.rdf`

[10] We used Gephi [BHJ09] to create this visualisation. The visualisation can be viewed in full color in the following link: `https://www.dropbox.com/s/879iaghq4qglth4/OS.pdf`. Colors outline the community structure of the network, i.e., colors signify groups of nodes with dense connections within them.

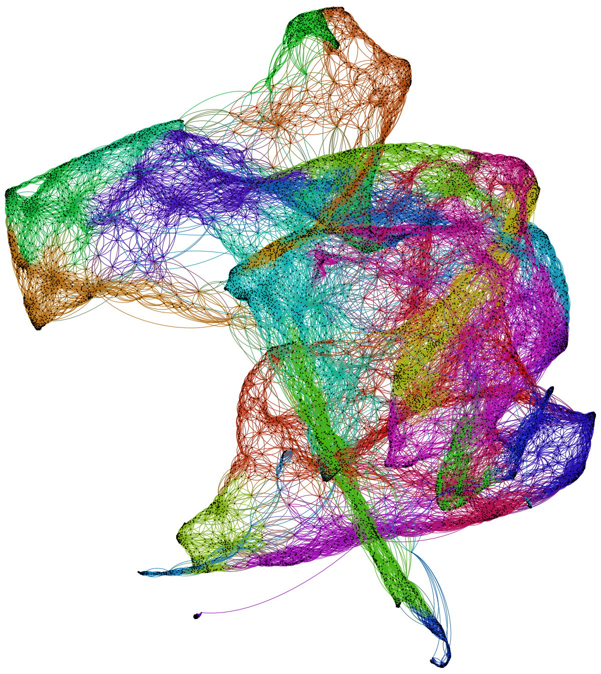


Fig. 10: A visualisation of the admingeo dataset

are shown in Figure 9. Again, PyRCC8 outperforms significantly all other reasoners, with the exception of Renz's solver when the whole dataset is considered at the final step, where both reasoners are very close to each other. At the final step, the existence of some identity relations in the network, cause the queue used by PyRCC8 to expand dramatically to retain candidate arcs for revision. Therefore, the advantage of starting with a compact queue of meaningful arcs disappears. We believe this is also the case with GQR. Notice that there is no experimental results for PelletSpatial after the step where 60000 relations were considered. At that point, PelletSpatial went into swap after having run for over 30 hours.

Since the admingeo dataset comprises a consistent constraint network, we also tried to impose an inconsistency in the dataset and perform our experiment again for all the aforementioned reasoners. Due to the sparseness of the constraint

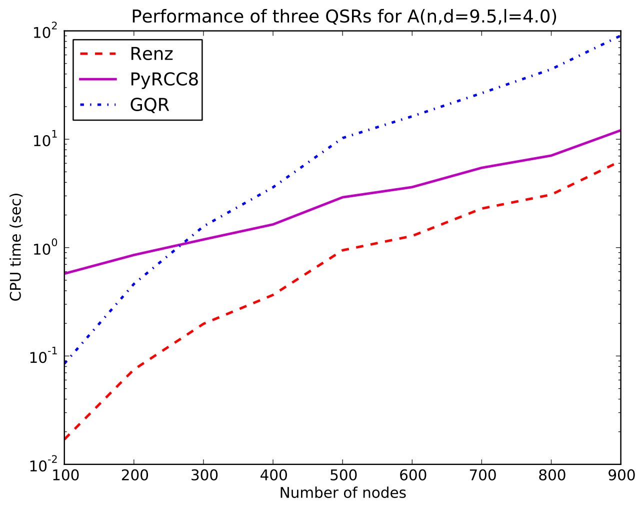


Fig. 11: Comparison of different consistency algorithms

network obtained from the dataset and the fact that it consists solely of RCC-8 base relations, the imposed inconsistencies could not propagate sufficiently and thinned out into trivial local inconsistencies almost immediately. In fact, every reasoner was able to detect the inconsistency very fast, during a first iteration of the queue of arcs which represent relations. Additionally, the fact that every reasoner was able to detect the inconsistency during a first iteration of the queue of arcs, deems the performance of every reasoner strongly dependent on the position of the arc in the queue that exposes the inconcistency, and, thus, renders any obtained results misleading and uninteresting.

To the best of our knowledge, this is the first set of experiments with RCC-8 reasoners where a real and big dataset has been used. We hope that using real and big datasets to evaluate the performance qualitative spatial reasoners will continue given the interest of the community[11] and the use of qualitative spatial relations in publicly available datasets today, e.g., in linked data [KKK+11, Ope12]. In Section 5.1 later, we try to tackle a dataset which is even bigger than admingeo.

Consistency checking. To assess the speed of the backtracking search of PyRCC8, we considered network sizes between 100 and 900 nodes. For each size, 30 networks were generated using all RCC-8 relations with Renz's random instance generator [RN01]. Again, networks were generated with an average degree of

[11] See the focus of the Workshop "Benchmarks and Applications of Spatial Reasoning" at IJCAI 2011 (`http://reasoner.informatik.uni-freiburg.de/ijcai11-bench/`).

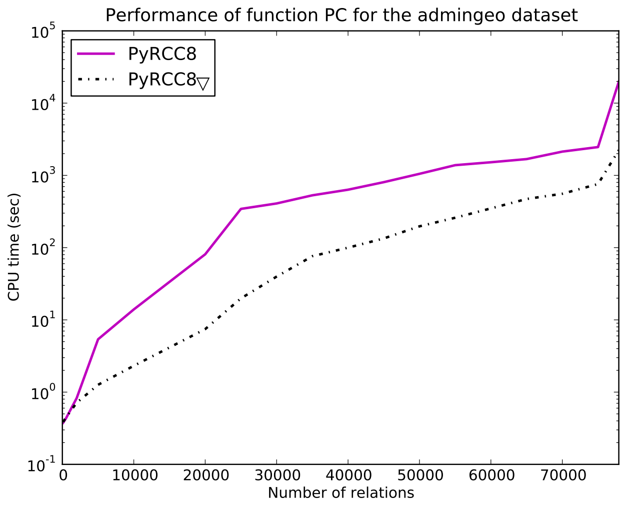


Fig. 12: Comparison diagram of PC algorithms on CPU time

9.5, to ensure the hardness of our instances. The results of the consistency experiment are shown in Figure 11. PyRCC8 outperforms GQR and comes close to the performance of the backtracking algorithm in Renz's solver. The latter result is due to more abstract coding of heuristics in PyRCC8 as opposed to Renz's solver, which affects the execution speed.

The excellent experimental results of PyRCC8 presented above demonstrate the advantages of this particular implementation, but also the potential benefits of trace-based JITs over static compilers. Figure 11 shows that as constraint networks grow larger, the trace-based JIT kicks in and makes PyRCC8 behave in a more scalable and robust manner as opposed to the statically compiled reasoners.

5 Experimental Results

In this section, we compare the performance of PyRCC8▽ with that of PyRCC8 performing experiments that target both path concistency and consistency checking implementations. We do not use other reasoners in our experiments, because our main point is to show how partial path consistency is compared to path consistency in the context of RCC-8. For this purpose, having two implementations that are very similar in their core components, is not only sufficient, but necessary for avoiding confusion and clearly comparing an aproach using triangulations of constraint graphs with an approach using complete graphs. Both reasoners were configured for best performance. The experiments were carried out on the same machine as described in Section 4.1, and both Python imple-

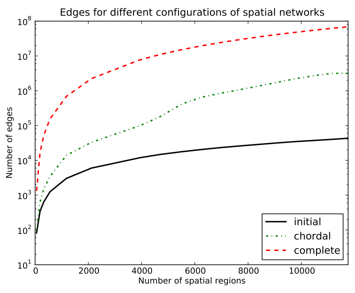
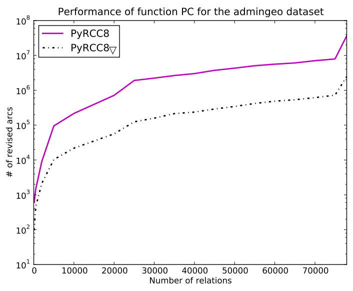
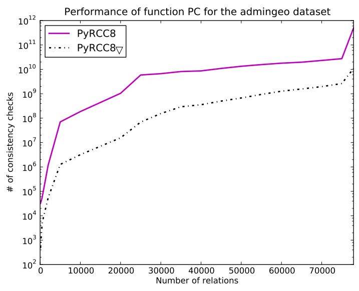


Fig. 13: Comparison diagram on # of edges for different graph configurations of instances of admingeo



(a) # of revised arcs



(b) # of consistency checks

Fig. 14: Comparison diagrams of PC algorithms

mentations were run with PyPy 1.8. Only one of the CPU cores was used for the experiments.

Path Consistency. In this case, we used the admingeo dataset presented earlier [GDH08]. We performed the experiment in the same way as described in Section 4.1. The results of the path consistency experiment using the (consistent) admingeo dataset are shown in Figure 12.

The path consistency implementation of PyRCC8▽, viz. ▽-path consistency, outperforms the path consistency implementation of PyRCC8 by a very large margin. When the whole dataset is considered at the final step, PyRCC8 decides the consistency of the constraint network in about 5 hours on the Lenny machine, whereas PyRCC8▽ requires less than 40 minutes for the same task. In fact, PyRCC8▽ runs significantly faster than PyRCC8 for all different network sizes. This comes as no surprise. Completing a network of more than 10000 nodes results in about 50 million edges that PyRCC8 has to consider. On the other hand, triangulating the network with an initial count of nearly 80000 edges, resulted in a total of only 4 million edges for PyRCC8$\triangledown$ to consider. A detailed diagrammatic comparison on the number of edges for initial, chordal, and complete graph configurations of admingeo dataset instances as a function of the considered number of nodes is shown in Figure 13. The results shown in Figure 12 are also comparable to the ones shown in Figure 9 of Section 4.1, since the same dataset is used and, thus, the performance of PyRCC8 remains unchanged. It follows that for this particular dataset, PyRCC8$\triangledown$ not only significantly outperforms PyRCC8, but the state of the art reasoners in total too.

The number of edges in a network inevitably affects the number of revisions of arcs that different path consistency algorithm implementations have to perform. A diagrammatic comparison on the number of arcs that each algorithm processes is shown in Figure 14a. The result is again overwhelmingly in favor of PyRCC8$\triangledown$. When the whole dataset is considered at the final step, PyRCC8 revises about 40 million arcs, whereas PyRCC8$\triangledown$ revises only 2.5 million arcs. Every revision of an arc results in several composition and intersection operations that we will refer to as consistency checks. A diagrammatic comparison on the number of consistency checks that each algorithm performs is shown in Figure 14b. At the final step, PyRCC8 performs around 500 billion consistency checks, whereas PyRCC8$\triangledown$ performs only 10 billion consistency checks.

A summary of the results that is based on the average of the different parameters used for comparing our PC algorithms for all steps follows. The percentage decrease is shown in the last column.

	PyRCC8	PyRCC8$\triangledown$	%
CPU time	1825.129s	289.203s	84.15%
revised arcs	4834133.78	373080.28	92.28%
consistency checks	$3.606e+10$	$1.181e+09$	96.72%

It is clear that for large sparse spatial networks, graph triangulation offers a great advantage over graph completion and, thus, $\triangledown$-path consistency is the better choice. Similar results are obtained for randomly generated sparse networks. However, path consistency implementations of the state of the art reasoners deal very easily with randomly generated instances, even if they are in the phase transition region, as the ones used in Section 4.1. We considered the admingeo dataset to be a much more interesting case, since it proved to really strain the different implementations.

Consistency checking. To assess the speed of the backtracking search algorithms we used for consistency checking, we considered network sizes of 100 nodes for an average degree d varying from 3 to 15 with a step of 0.5. For each series, 300 networks were generated using all RCC-8 relations with Renz's random instance generator [RN01]. The hardest instances are located in an interval where the average degree ranges from 8 to 11 and the phase transition occurs. The main objective of our experimentation is to compare the efficiency of the different consistency algorithms. For this purpose we use the same parameters as with the path consistency experiment, that is, CPU time, number of revised arcs,

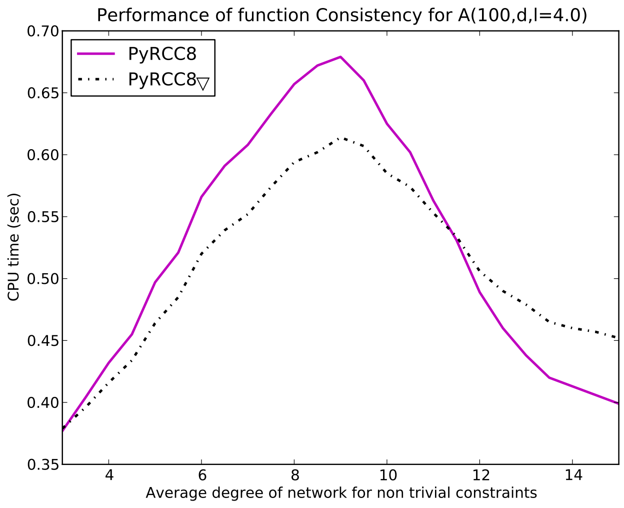


Fig. 15: Comparison diagram of consistency algorithms on CPU time

and number of consistency checks. The results of the consistency experiment based on the CPU time of execution are shown in Figure 15.

PyRCC8$\triangledown$ outperforms PyRCC8 as long as relatively sparse constraint networks are considered. Significant gains in the performance of PyRCC8$\triangledown$ are noted in the phase transition region where the hardest and more time consuming, in terms of solubility, instances appear. However, the performance of PyRCC8$\triangledown$ deteriorates when constraint networks of average degree $d > 12$ are considered. RCC-8 networks with an average degree $d > 12$ are over-constrained, dense, and soluble with a very low probability, thus, they are often easy to decide [RN01]. As networks become more dense, PyRCC8$\triangledown$ tries to simulate the behavior of PyRCC8, by considering more and more initial edges that result to a much denser chordal graph as shown in Figure 16. This has a direct impact on PyRCC8$\triangledown$'s computational complexity. Let us explain why.

A chordal constraint graph in PyRCC8$\triangledown$ is represented as a *dictionary*, an unordered set of *key:value* pairs where every key is a vertex of the graph and its value is the set of its neighbors. When an arc (i, j) is processed we compute the intersection between vertex's i and vertex's j sets of neighbors to obtain all triangles of relations that arc (i, j) is part of. In the average case, the time complexity for this operation is $O(min(|N(i)|, |N(j)|)$. It is clear that a big average degree of the initial network results in a bigger average degree of its chordal constraint graph after triangulation and, thus, time complexity rises. Of course, one could precompute and store all triangles of relations, but for large datasets this would lead to large space requirements. Though it may be argued that the time performance of the $\triangledown$-path consistency algorithm is related to

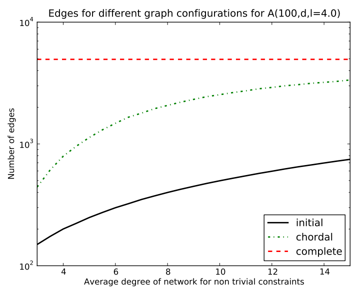
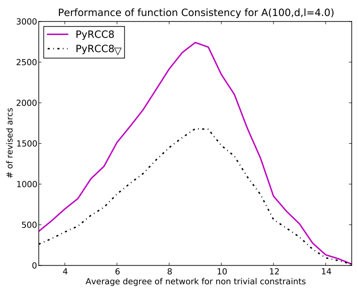
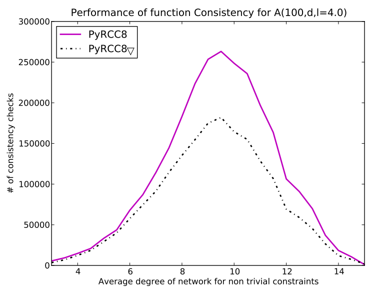


Fig. 16: Comparison diagram on # of edges for different graph configurations of A(100,d,l=4.0) instances



(a) # of revised arcs



(b) # of consistency checks

Fig. 17: Comparison diagrams of PC algorithms

the details of our implementation, intuitively, a cost in restricting consistency checking to the corresponding chordal graph of an RCC-8 network is inevitable, and must be paid at some point regardless of implementation design.

We continue our analysis with a diagrammatic comparison on the number of arcs that each consistency algorithm processes, shown in Figure 17a. Again, the difference in the performance of PyRCC8▽ and PyRCC8 is bigger in the phase transition region. Finally, since every revision of an arc results in several consistency checks, similarly with the path consistency experiment, we provide a diagrammatic comparison on the number of consistency checks that each algorithm performs, shown in Figure 17b. A summary of the results that is based on the average of the different parameters used for our comparisons for all steps follows. The percentage decrease is shown in the last column.

	PyRCC8	PyRCC8▽	%
CPU time	0.524s	0.509s	2.80%
revised arcs	1300.681	801.204	38.40%
consistency checks	105751.173	74864.985	29.21%

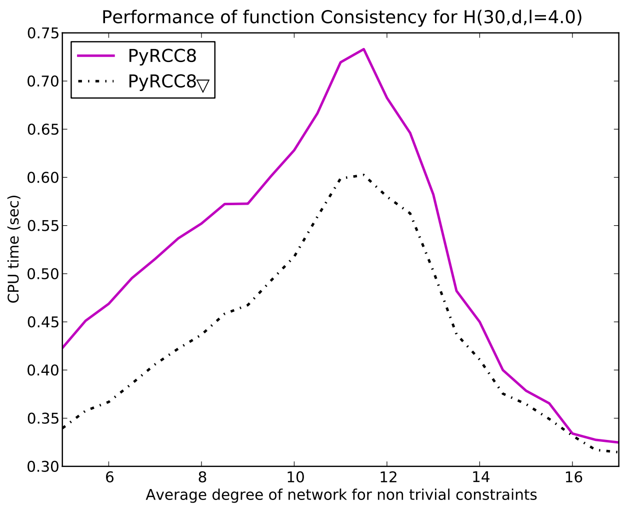


Fig. 18: Comparison diagram of consistency algorithms on CPU time

One would expect that since our consistency algorithm implementations are strongly dependent on the underlying path consistency algorithms, performance results of our consistency experiment would reflect those of our path consistency experiment. This is true up to the point when dense graphs of relative big size are considered. Our experimental results show that the performance of PyRCC8$\triangledown$ is deteriorating, with respect to the performance of PyRCC8, when dealing with dense graphs. However, overall the performance of PyRCC8$\triangledown$ is much better, especially for network instances in the phase transition region that are the most difficult to solve. We used the maximal tractable subset $\hat{\mathcal{H}}_8$ of RCC-8 as a split set (line 6 in Figure 6), since it best decomposes RCC-8 relations [RN01]. The performance gain in the phase transition region would be more apparent if we had opted for a tractable set of relations with a bigger average branching factor (e.g., the set of RCC-8 base relations $\mathcal{B}$) [RN99], because this would make input RCC-8 networks even more difficult to solve in the phase transition region.

As a second experiment for assessing the speed of the two backtracking algorithms, we considered hard instances with network sizes of 30 nodes for an average degree d varying from 5 to 17 with a step of 0.5. For each series, 300 networks were generated using only RCC-8 relations from the $\mathcal{NP}_8$ class of relations with Renz's random instance generator [RN01]. In the case of $\mathcal{NP}_8$ relations the hardest instances are located in an interval where the average degree ranges from 10 to 13 and the phase transition occurs. The results of this experiment based on the CPU time of execution are shown in Figure 18. An interesting immediate observation is that in this experiment PyRCC8$\triangledown$ performs better than PyRCC8 for all series of varying degrees, and the performance of PyRCC8$\triangledown$ does not

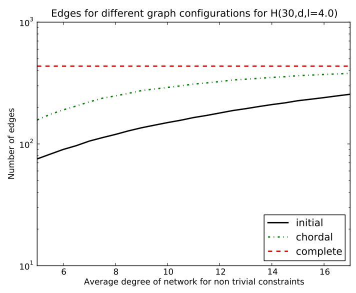
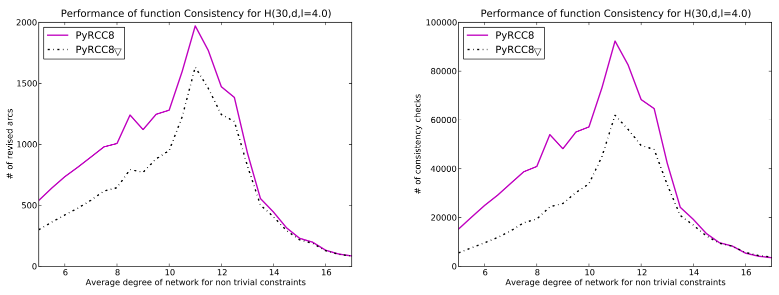


Fig. 19: Comparison diagram on # of edges for different graph configurations of H(30,d,l=4.0) instances



(a) # of revised arcs (b) # of consistency checks

Fig. 20: Comparison diagrams of PC algorithms

deteriorate when dense networks are considered. We provide the following two explanations for this behavior: (*i*) the network sizes considered are small and don't allow for a big average degree[12] of the chordal constraint graphs which are obtained after triangulating the initial networks, and (*ii*) network instances with only $\mathcal{NP}_8$ relations become over-constrained at a mush faster rate than normal RCC-8 network instances and, thus, pose nearly no challenge at all for either reasoner when they become dense. Figure 13 provides a detailed diagrammatic comparison on the number of edges for initial, chordal, and complete graph configurations of hard instances as a function of the average degree of initial networks.

A diagrammatic comparison on the number of arcs that each consistency algorithm processes is shown in Figure 20a. Finally, since every revision of an arc results in several consistency checks, similarly with the path consistency

[12] We remind the reader that the bigger the average degree of a graph, the more complexity is imposed on PyRCC8$\triangledown$ as explained in the previous experiment where network instances from all RCC-8 relations where considered.

experiment, we provide a diagrammatic comparison on the number of consistency checks that each algorithm performs, shown in Figure 20b.

A summary of the results that is based on the average of the different parameters used for our comparisons for all steps follows. The percentage decrease is shown in the last column.

	PyRCC8	PyRCC8$\triangledown$	%
CPU time	0.516s	0.438s	15.11%
revised arcs	867.480	650.020	25.07%
consistency checks	37165.678	23061.819	37.95%

This experiment demonstrates that the use of $\triangledown$-path consistency can be also extremely beneficial for hard instaces of RCC-8 networks, i.e., networks that consist of the $\mathcal{NP}_8$ class of relations of RCC-8. Although it is safe to assume that hard instances do not correspond to real case scenarios, they are often used to test the efficiency of algorithm implementations. Again, the maximal tractable subset $\hat{\mathcal{H}}_8$ of RCC-8 was our split set of choice.

5.1 Pushing the Envelope in RCC-8 Reasoning

We employed the GADM[13] dataset, a spatial database of the location of the world's administrative areas, to perform some experiments. The GADM dataset consists of 276728 regions and 590865 RCC-8 relations. Completing such a network results in about 40 billion edges. As expected, none of the state of the art RCC-8 reasoners [RN01, GWW08, SS09], including PyRCC8 and PyRCC8$\triangledown$, could tackle this dataset. Specifically, GQR and Renz's solver cannot reason with more than 36000 and 32000 spatial regions respectively, as they produce memory allocation errors. We did not test the dataset against PelletSpatial, but given the poor results with the far smaller admingeo dataset described in Section 4.1, it was deemed unnecessary. Then, we created a small subset of the GADM dataset by considering only the 5% of its relations. This resulted in a dataset of 38000 spatial regions and 31000 relations. We managed to test both PyRCC8 and PyRCC8$\triangledown$ against this dataset. Given the sparseness of the network, PyRCC8 was able to decide its satisfiability in approximately 106 seconds, whereas PyRCC8$\triangledown$ required only around 16 seconds.

In light of the above experiment, the question remains whether it is possible to reason with such large datasets using the known approaches in spatial reasoning and the state of the art reasoning tools. We believe that in order to make progress, we have to go beyond current implementations and consider alternative techniques, e.g., based on multicore CPUs or clusters of computers. In our work, we are currently exploring the use of systems like Pregel [MAB$^+$10] for scalable and fault-tolerant graph computations over clusters of commodity machines. We consider chordal graphs as the graph represention of choice for constructing network decompositions that could be useful for the aforementioned techniques.

[13] `http://gadm.geovocab.org/`

6 Related Work

The previous sections have already discussed a lot of related work in the area of RCC-8 and compared our work with existing RCC-8 reasoners. In this section, we discuss some more published papers that relate to other aspects of our work.

Bliek and Sam-Haroud were the first to study chordal graphs in the context of finite domain CSPs in [BSH99]. They show that for a convex CSP with graph G, strong path consistency is equivalent to strong path consistency computed only on the completion of G with the arcs that make it chordal. Similarly, Chmeiss and Condotta use partial path consistency to decide the consistency of pre-convex Interval Algebra networks [CC11]. Finally, chordal graphs have also been used for solving networks of temporal difference constraints [XC03, PdWvdK08, PdWvdK12]. All of the above works exhibit many similarities in terms of the data structures and triangulation algorithms used for chordal graphs.

The proof techniques that we used in Section 3.2 are related to a number of papers that have appeared in the literature. In [LW06], Li et al. show that RCC-8 binary constraint networks can be consistently *one-shot* extended. In [LKRL08], various ways of combining binary constraint networks in qualitative reasoning are discussed. In [LHR09], Li et al. speed up consistency checking in sparse atomic Interval Algebra networks, by recursively decomposing the networks in a divide-and-conquer manner and eliminating the need for examining triangles across subnetworks when enforcing path consistency. In [BW11], Bodirsky et al. introduce the notion of tree decomposition for constraint networks, and define the amalgamation property for atomic RCC-8 networks that allows satisfiable subnetworks to be glued together in a tree-like manner resulting in a satisfiable network. In [Hua12], Huang shows that the patchwork property, in the presence of compactness, holds for Interval Algebra and RCC-8 networks, for all maximal tractable subsets of the respective calculi, thus, significantly stregthening all previous results regarding the Interval Algebra, RCC-8, and their fragments and extensions.

7 Conclusion and Future Work

In this paper we introduced $\triangledown$-path consistency for RCC-8 networks. Based on the patchwork property defined in [Hua12], we showed that $\triangledown$-path consistency is sufficient to decide the consistency problem for the maximal tractable subsets $\hat{\mathcal{H}}_8, \mathcal{C}_8$, and $\mathcal{Q}_8$ of RCC-8. Further, we gave algorithms to solve networks of RCC-8 and presented extensive experimental results on both real and random network instances of RCC-8 accompanied by a detailed summary. Finally, we showed that in the case of RCC-8 we cannot have a result similar to the one by Bliek and Sam-Haroud that consider convex CSPs and show that the pruning capacity of path consistency on the chordal graph is equivalent to the pruning capacity of path consistency on the completed network when we consider the common edges [BSH99].

Future work consists of using other methods of triangulation and comparing the behavior of our algorithm for these different methods. We consider chordal graphs as the graph represention of choice for constructing network decompositions, and we will explore applications of this representation with distributed systems such as Pregel [MAB+10]. We would also like to study solving spatial networks incrementally in the way of [PdWYS10]. Further, we will try to find subsets of RCC-8 where $\triangledown$-path consistency can be used to derive minimal RCC-8 networks.

As a final note, it was stated in Section 2 that RCC-8 networks are encoded as CSPs. Thus, we would like to bring RCC-8 reasoning closer to traditional constraint programming notions. For example, since path consistency is one of the central methods to solve constraint networks in RCC-8, we could explore the use of singleton local consistencies [PSW00] in the context of qualitative spatial reasoning. Additionally, it would be interesting to introduce *soft constraints* [Bar02] in qualitative constraint networks. Finally, we will research on how our techniques can benefit from the existence of implicit constraints as recently introduced in [Ren12].

Acknowledgment

This work has been funded by the FP7 project TELEIOS (257662). A preliminary version of this paper appeared in the *Proceedings of the* 24th *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)* [SK12].

References

Bar02. Roman Bartak. Modelling Soft Constraints: A Survey. *Neural Network World*, 12:421–431, 2002.

BBH02. Anne Berry, Jean R. S. Blair, and Pinar Heggernes. Maximum Cardinality Search for Computing Minimal Triangulations. In *WG*, 2002.

BCF+11. Carl Friedrich Bolz, Antonio Cuni, Maciej Fijalkowski, Michael Leuschel, Samuele Pedroni, and Armin Rigo. Runtime feedback in a meta-tracing JIT for efficient dynamic languages. In *ICOOOLPS*, 2011.

BCFR09. Carl Friedrich Bolz, Antonio Cuni, Maciej Fijalkowski, and Armin Rigo. Tracing the meta-level: PyPy's tracing JIT compiler. In *ICOOOLPS*, 2009.

BGWH11. Mehul Bhatt, Hans Guesgen, Stefan Wölfl, and Shyamanta Hazarika. Qualitative Spatial and Temporal Reasoning: Emerging Applications, Trends, and Directions. *Spatial Cognition & Computation*, 11:1–14, 2011.

BHJ09. Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An Open Source Software for Exploring and Manipulating Networks. In *ICWSM*, 2009.

BSH99. Christian Bliek and Djamila Sam-Haroud. Path consistency on triangulated constraint graphs. In *IJCAI*, 1999.

BW11. Manuel Bodirsky and Stefan Wölfl. RCC8 is polynomial on networks of bounded treewidth. In *IJCAI*, 2011.

CC11. Assef Chmeiss and Jean-Francois Condotta. Consistency of Triangulated Temporal Qualitative Constraint Networks. In *ICTAI*, 2011.

Cla81. Bowman L Clarke. A calculus of individuals based on “connection”. *NDJFL*, 22:204–218, 1981.

CM94. Andrés Cano and Serafín Moral. Heuristic Algorithms for the Triangulation of Graphs. In *IPMU*, 1994.

CR08. A. G. Cohn and J. Renz. Qualitative Spatial Representation and Reasoning. *Handbook of Knowledge Representation*, pages 551–596, 2008.

GDH08. John Goodwin, Catherine Dolbear, and Glen Hart. Geographical Linked Data: The Administrative Geography of Great Britain on the Semantic Web. *Transactions in GIS*, 12:19–30, 2008.

Gol04. M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Elsevier Science, 2nd edition, 2004.

Got94. Nicholas Mark Gotts. How Far Can We ‘C’? Defining a ‘Doughnut’ Using Connection Alone. In *KR*, 1994.

Got96. Nicholas Mark Gotts. Topology From A Single Primitive Relation: Defining Topological Properties and Relations In Terms Of Connection. Technical report, School of Computer Studies, University of Leeds, 1996.

GWW08. Zeno Gantner, Matthias Westphal, and Stefan Woelfl. GQR - A Fast Reasoner for Binary Qualitative Constraint Calculi. In *AAAI Workshop on Spatial and Temporal Reasoning*, 2008.

Haz12. S.M. Hazarika. *Qualitative Spatio-Temporal Representation and Reasoning: Trends and Future Directions*. Igi Global, 2012.

Hua12. Jinbo Huang. Compactness and Its Implications for Qualitative Spatial and Temporal Reasoning. In *KR*, 2012.

KKK$^+$11. Manolis Koubarakis, Kostis Kyzirakos, Manos Karpathiotakis, Charalampos Nikolaou, Michael Sioutis, Stavros Vassos, Dimitrios Michail, Themistoklis Herekakis, Charalampos Kontoes, and Ioannis Papoutsis. Challenges for Qualitative Spatial Reasoning in Linked Geospatial Data. In *IJCAI Workshop on Benchmarks and Applications of Spatial Reasoning*, 2011.

LHR09. Jason Jingshi Li, Jinbo Huang, and Jochen Renz. A divide-and-conquer approach for solving interval algebra networks. In *IJCAI*, 2009.

LKRL08. Jason Jingshi Li, Tomasz Kowalski, Jochen Renz, and Sanjiang Li. Combining binary constraint networks in qualitative reasoning. In *ECAI*, 2008.

LM94. Peter B. Ladkin and Roger D. Maddux. On binary constraint problems. *JACM*, 41:435–469, 1994.

LM07. C. Lutz and M. Milicic. A Tableau Algorithm for DLs with Concrete Domains and GCIs. *JAR*, 38:227–259, 2007.

LR04. Gérard Ligozat and Jochen Renz. What Is a Qualitative Calculus? A General Framework. In *PRICAI*, 2004.

LW06. Sanjiang Li and Huaiqing Wang. RCC8 binary constraint network can be consistently extended. *AI*, 2006.

LY03. Sanjiang Li and Mingsheng Ying. Region connection calculus: Its models and composition table. *Artif. Intell.*, 145(1-2):121–146, 2003.

MAB$^+$10. Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD*, 2010.

Mon74. Ugo Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Inf. Sci.*, 7:95–132, 1974.

Ope12. Open Geospatial Consortium. OGC GeoSPARQL - A geographic query language for RDF data. OGC® Implementation Standard, 2012.

PdWvdK08. Léon Planken, Mathijs de Weerdt, and Roman van der Krogt. P3C: A New Algorithm for the Simple Temporal Problem. In *ICAPS*, 2008.

PdWvdK12. Léon Planken, Mathijs de Weerdt, and Roman van der Krogt. Computing All-Pairs Shortest Paths by Leveraging Low Treewidth. *JAIR*, 43:353–388, 2012.

PdWYS10. Léon Planken, Mathijs de Weerdt, and Neil Yorke-Smith. Incrementally Solving STNs by Enforcing Partial Path Consistency. In *ICAPS*, 2010.

PSW00. Patrick Prosser, Kostas Stergiou, and Toby Walsh. Singleton Consistencies. In *CP*, 2000.

RCC92. David A. Randell, Zhan Cui, and Anthony Cohn. A Spatial Logic Based on Regions and Connection. In *KR*. 1992.

Ren99. Jochen Renz. Maximal Tractable Fragments of the Region Connection Calculus: A Complete Analysis. In *IJCAI*, 1999.

Ren12. Jochen Renz. Implicit Constraints for Qualitative Spatial and Temporal Reasoning. In *KR*, 2012.

RL05. Jochen Renz and Gérard Ligozat. Weak Composition for Qualitative Spatial and Temporal Reasoning. In *CP*, 2005.

RN98. Jochen Renz and Bernhard Nebel. Spatial Reasoning with Topological Information. In *Spatial Cognition*, 1998.

RN99. Jochen Renz and Bernhard Nebel. On the Complexity of Qualitative Spatial Reasoning: A Maximal Tractable Fragment of the Region Connection Calculus. *AI*, 108:69–123, 1999.

RN01. Jochen Renz and Bernhard Nebel. Efficient Methods for Qualitative Spatial Reasoning. *JAIR*, 15:289–318, 2001.

RN07. Jochen Renz and Bernhard Nebel. Qualitative Spatial Reasoning Using Constraint Calculi. In *Handbook of Spatial Logics*, pages 161–215. 2007.

SK12. Michael Sioutis and Manolis Koubarakis. Consistency of Chordal RCC-8 Networks. In *ICTAI*, 2012. Athens, Greece, November 07–09, 2012.

SS09. Markus Stocker and Evren Sirin. PelletSpatial: A Hybrid RCC-8 and RDF/OWL Reasoning and Query Engine. In *OWLED*, 2009.

vBM96. Peter van Beek and Dennis W. Manchak. The design and experimental analysis of algorithms for temporal reasoning. *JAIR*, 4:1–18, 1996.

XC03. Lin Xu and Berthe Y. Choueiry. A New Efficient Algorithm for Solving the Simple Temporal Problem. In *TIME*, 2003.

D. Incomplete Information in RDF

Incomplete Information in RDF

Charalampos Nikolaou
University of Athens
University Campus, Ilisia
15784 Athens, Greece
charnik@di.uoa.gr

Manolis Koubarakis
University of Athens
University Campus, Ilisia
15784 Athens, Greece
koubarak@di.uoa.gr

ABSTRACT

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

1. INTRODUCTION

Incomplete information has been studied in-depth in relational databases [12, 6] and knowledge representation. It is also an important issue in Semantic Web frameworks such as RDF, description logics, and OWL 2 especially given that all these systems rely on the Open World Assumption (OWA). Making the OWA means that we cannot capture negative information implicitly, i.e., if a formula ϕ is not entailed by our knowledge base, we cannot assume its negation as in the Closed World Assumption (CWA).

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. Incomplete information can also arise even if we start from complete databases, e.g., in relational view updates, data integration, data exchange, etc., 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 [2, 5]. 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.

[9] introduces the concept of *anonymous timestamps* in general temporal RDF graphs, i.e., graphs containing quads of the form $(s, p, o)[t]$ where t is a timestamp (a natural number) or an anonymous timestamp x stating that the triple (s, p, o) is valid in some unknown time point x. [11] subsequently extends the concept of general temporal RDF graphs of [9] so that one is allowed to express temporal constraints involving anonymous timestamps using a formula ϕ which is a conjunction of order constraints $x_1 \; OP \; x_2$ where OP is an arithmetic comparison operator such as $<$, $\leq$, etc. [11] calls c-temporal graphs the resulting pairs (G, ϕ) where G is a general temporal RDF graph and ϕ is a conjunction of constraints. [11] defines a semantics for c-temporal graphs and studies the relevant problem of entailment.

More recently, [4] examines the question of whether SPARQL is an appropriate language for RDF given the OWA typically associated with the framework. It defines a *certain answer* semantics for SPARQL query evaluation based on well-known ideas from incomplete information research. According to this semantics, if G is an RDF graph then evaluating a SPARQL query q over G is defined as evaluating q over all graphs $H \supseteq G$ that are possible extensions of G according to the OWA, and then taking the intersection of all answers. [4] shows that if we evaluate a monotone graph pattern (e.g., one using only the operators AND, UNION, and FILTER) using the well-known W3C semantics, we get the same result we would get if we used the certain answer semantics. The converse also holds, thus monotone SPARQL graph patterns are exactly the ones that have this nice property. However, OPTIONAL (OPT) is not a monotone operator and the two semantics do not coincide for it. [4] defines the notion of weak monotonicity that appears to capture the intuition behind OPT, and shows 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, [4] shows that the fragment of SPARQL consisting of the well-designed graph patterns defined originally in [26] is weakly monotone.

1.1 Contributions

In this paper we continue the line of research started by [9, 11, 4] and study in a general way an important kind of incomplete information that has so far been ignored in the context of RDF. Our contributions are the following.

First, we extend RDF with the ability to define a new kind of literals for each datatype. These literals will be called *e-literals* ("e" comes from the word "existential") and can be used to represent values of properties that *exist but are unknown or partially known.* Such information is abundant in recent applications where RDF is being used (e.g., sensor networks, the modeling of geospatial information, etc.). In the proposed extension of RDF, called RDF$^{\text{i}}$ (where "i" stands for "incomplete"), e-literals are allowed to appear only in the object position of triples.

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 [6, 15]. In the spirit of this tradition, RDF$^{\text{i}}$ allows partial information regarding property values represented by e-literals to be expressed by a quantifier-free formula of a first-order *constraint language* $\mathcal{L}$. Thus, RDF$^{\text{i}}$ extends the concept of an RDF graph to the concept of an RDF$^{\text{i}}$ *database* which is a pair (G, ϕ) where G is an RDF graph possibly containing triples with e-literals in their object positions, and ϕ is a quantifier-free formula of $\mathcal{L}$. Our recent workshop paper [22] motivates the need for introducing RDF$^{\text{i}}$ by concentrating on the representation of incomplete spatial knowledge.

Following ideas from the incomplete information literature [12, 6], we develop a semantics for RDF$^{\text{i}}$ databases and SPARQL query evaluation. The semantics defines the set of possible RDF graphs corresponding to an RDF$^{\text{i}}$ database and the fundamental concept of certain answer for SPARQL query evaluation over an RDF$^{\text{i}}$ database. We transfer the well-known concept of *representation system* from [12] to the case of RDF$^{\text{i}}$, and show that CONSTRUCT queries without blank nodes in their templates and using only operators AND, UNION, and FILTER or the restricted fragment of graph patterns corresponding to the well-designed patterns of [4] can be used to define a representation system for RDF$^{\text{i}}$. Our results for 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 answer to SPARQL queries over RDF$^{\text{i}}$ databases and present an algorithm for its computation. Finally, we present preliminary complexity results for computing certain answers by considering equality, temporal, and spatial constraint languages $\mathcal{L}$ and the class of CONSTRUCT queries of our representation system. Our results show that the data complexity of evaluating a query of this class over RDF$^{\text{i}}$ databases increases from LOGSPACE (the upper bound for evaluating queries from this class over RDF graphs [26]) to coNP-complete for the case of equality and temporal constraints. This result is in line with similar complexity results for querying incomplete information in relational databases [6, 14]. The same coNP-completeness bound is shown for the case of spatial constraints on rectangles in $\mathbb{Q}^2$ [14]. For topological constraints over more general spatial regions (regular, closed subsets of $\mathbb{Q}^2$), the best upper bound that we can show is EXPTIME. To the best of our knowledge, it is an open problem how to achieve better complexity results in this case. The complexity of the closely related problem of SPARQL query evaluation over RDF graphs (e.g., as manifested in geospatial extensions stSPARQL [18] and GeoSPARQL [23]) has not been investigated so far in any detail, and it remains an open problem as well.

The organization of the paper is as follows. Section 2 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 will be used in the paper. Section 3 introduces RDF$^{\text{i}}$ and then Section 4 defines its semantics. Section 5 defines the evaluation of SPARQL queries over RDF$^{\text{i}}$ databases. Section 6 presents fragments of SPARQL that can be used to define a representation system for RDF$^{\text{i}}$. Section 7 gives an algorithm for computing the certain answer for SPARQL queries over RDF$^{\text{i}}$ databases and presents our complexity results. Sections 8 and 9 discuss related and future work respectively. Last, the Appendix contains the complete proofs of the results established in this paper.

2. CONSTRAINT LANGUAGES

We will consider many-sorted first-order languages, structures, and theories. 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 $\mathcal{L}$, we will distinguish a class of quantifier free formulae called *$\mathcal{L}$-constraints*. The atomic formulae of $\mathcal{L}$ will be included in the class of $\mathcal{L}$-constraints. There will also be two distinguished $\mathcal{L}$-constraints *true* and *false* with obvious semantics.

Every first-order language $\mathcal{L}$ we consider has a distinguished equality predicate, denoted by EQ, with the standard semantics. The class of $\mathcal{L}$-constraints is assumed to: *a*) contain all formulae t_1 EQ t_2 where t_1, t_2 are terms of $\mathcal{L}$, and *b*) be *weakly closed under negation*, i.e., the negation of every $\mathcal{L}$-constraint is equivalent to a disjunction of $\mathcal{L}$-constraints. This property is needed in Section 7 where the certain answer to a SPARQL query over an RDF$^{\text{i}}$ database is computed.

Section A of the Appendix defines formally various constraint languages that allow us to explore the scope of modeling possibilities that RDF$^{\text{i}}$ offers. These languages are ECL, diPCL, dePCL, PCL, TCL and RCL. ECL is the first order language of equality constraints of the form x EQ y and x EQ c (where x, y are variables and c is a constant) interpreted over an infinite domain [6]. 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 [12]. The languages diPCL and dePCL are the first order languages of temporal difference constraints of the form $x - y \leq c$ interpreted over the integers (diPCL) or the rationals (dePCL) [14]. These are constraint languages that allow RDF$^{\text{i}}$ to represent incomplete temporal information as in [9, 11] and older works such as [16]. PCL, TCL and RCL are spatial constraint languages and are defined in detail below. PCL is the language that we have used in our introductory paper [22] and we will also use it in the examples of this paper. PCL, TCL and RCL will be referred to in Section 8 where the power of RDF$^{\text{i}}$ for geospatial modeling is compared with other modeling frameworks.

2.1 The Languages PCL, TCL, and RCL

The language *PCL* (*P*olygon *C*onstraint *L*anguage) allows us to represent topological properties of non-empty, regular closed subsets of $\mathbb{Q}^2$ (we call them *regions*). PCL is a first-order language with the following 6 binary predicate symbols corresponding to the topological relations of RCC-8 calculus [28]: $\mathrm{DC}, \mathrm{EC}, \mathrm{PO}, \mathrm{EQ}, \mathrm{TPP}$, and NTPP. The constant symbols of PCL represent polygons in $\mathbb{Q}^2$. We will write these constants as conjunctions of linear constraints in quotes (half-space representation of the convex polygon). The terms and atomic formulae of PCL are defined as follows. Constants and variables are *terms*. An *atomic formula* of PCL (PCL-constraint) is a formula of the form $t_1 \ R \ t_2$ where t_1, t_2 are terms and R is one of the above predicates.

The intended structure for PCL, denoted by $\mathbf{M}_{PCL}$, has the set of non-empty, regular closed subsets of $\mathbb{Q}^2$ as its domain. $\mathbf{M}_{PCL}$ interprets each constant symbol by the corresponding polygon in $\mathbb{Q}^2$ and each of the predicate symbols by the corresponding topological relation of RCC-8 [28].

Language TCL (*T*opological *C*onstraint *L*anguage) is defined like PCL, but now terms can only be variables (no topological reasoning with constants, i.e., landmarks [20], is allowed). Language RCL (*R*ectangle *C*onstraint *L*anguage) is a simpler first-order constraint language that represents information about *rectangles* in $\mathbb{Q}^2$ using rational constants and order or difference constraints ($x - y \leq c$) on the vertices of rectangles. RCL has essentially the same expressive power with dePCL, but it's been carefully crafted for rectangles.

3. THE RDF$^{\text{i}}$ FRAMEWORK

As in theoretical treatments of RDF [26], we assume the existence of pairwise-disjoint, countably infinite sets I, B, and L that contain IRIs, blank nodes, and literals respectively. We also assume the existence of a datatype map M [10] and distinguish a set of datatypes A from M for which e-literals are allowed. Finally, we assume the existence of a many-sorted first order constraint language $\mathcal{L}$ with the properties discussed in Section 2. $\mathcal{L}$ is related to the datatype map M in the following way:

- 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. We also assume the existence of a countably infinite set of e-literals for each datatype in A and use U to denote the union of these sets. By convention, the identifiers of e-literals will start with an underscore, e.g., `_R5`. C and U are assumed to be disjoint from each other and from I, B, and L. 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 2.

We now define the basic concepts of RDF$^{\text{i}}$: e-triples, conditional triples, conditional graphs, global constraints, and databases. Triples in RDF$^{\text{i}}$ (called e-triples) are as in RDF but now e-literals are also allowed in the object position. Combining an e-triple with a conjunction of $\mathcal{L}$-constraints, we get a conditional triple. Graphs in RDF$^{\text{i}}$ are conditional and consist of sets of conditional triples. Global constraints are simply Boolean combinations of $\mathcal{L}$-constraints. The combination of a conditional graph and a global constraint is called a database.

Definition 3.1. *An* e-triple *is an element of the set* $(I \cup B) \times I \times T$. *If* (s, p, o) *is an e-triple, s will be called the subject, p the predicate, and o the object of the triple. A* conditional triple *is a pair* (t, θ) *where t is an e-triple and θ is a conjunction of $\mathcal{L}$-constraints. If* (t, θ) *is a conditional triple, θ will be called the condition of the triple.*

Definition 3.2. *A* global constraint *is a Boolean combination of $\mathcal{L}$-constraints.*

Definition 3.3. *A* conditional graph *is a set of conditional triples. An* RDF$^{\text{i}}$ 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$^{\text{i}}$ is possible.

Example 3.4. *The following pair is an* RDF$^{\text{i}}$ *database.*

```
( { ((hotspot1, type, Hotspot), true),
    ((fire1, type, Fire), true),
    ((hotspot1, correspondsTo, fire1), true),
    ((fire1, occuredIn, _R1), true) },

    _R1 NTPP "x ≥ 6 ∧ x ≤ 23 ∧ y ≥ 8 ∧ y ≤ 19" )
```

Example 3.5. *The following pair is an* RDF$^{\text{i}}$ *database with a disjunctive global constraint.*

```
( { ((hotspot1, type, Hotspot), true),
    ((fire1, type, Fire), true),
    ((hotspot1, correspondsTo, fire1), true),
    ((fire1, occuredIn, _R1), true),
    ((fire2, occuredIn,
                "x ≥ 6 ∧ x ≤ 23 ∧ y ≥ 8 ∧ y ≤ 19"), true) },

(_R1 NTPP "x ≥ 6 ∧ x ≤ 23 ∧ y ≥ 8 ∧ y ≤ 19"   ∧
 _R1 NTPP "x ≥ 10 ∧ x ≤ 21 ∧ y ≥ 12 ∧ y ≤ 17") ∨
                _R1 PO "x ≥ 2 ∧ x ≤ 6 ∧ y ≥ 4 ∧ y ≤ 8" )
```

4. SEMANTICS OF RDF$^{\text{i}}$

The semantics of RDF$^{\text{i}}$ are inspired by [12]. An RDF$^{\text{i}}$ database $D = (G, \phi)$ corresponds to a set of possible RDF graphs each one representing a possible state of the real world. This set of possible graphs captures completely the semantics of an RDF$^{\text{i}}$ database. The global constraint ϕ determines the number of possible RDF graphs corresponding to D; there is one RDF graph for each solution of ϕ obtained by considering the e-literals of ϕ as variables and solving the constraint ϕ.

Example 4.1. *Let* $D = (G, \phi)$ *be the* RDF$^{\text{i}}$ *database given in Example 3.4. Database D mentions a hotspot, which is located in a region that is inside but does not intersect with the boundary of rectangle defined by the points* $(6, 8)$ *and* $(23, 19)$. *The same knowledge can be represented by an (infinite) set of possible RDF graphs, one for each rectangle inside P. Two of these graphs are:*

```
G1 = { (hotspot1, type, Hotspot), (fire1, type, Fire),
     (hotspot1, correspondsTo, fire1),
     (fire1, occuredIn, "x ≥ 11 ∧ x ≤ 15 ∧ y ≥ 13 ∧ y ≤ 15")}

G2 = { (hotspot1, type, Hotspot), (fire1, type, Fire),
     (hotspot1, correspondsTo, fire1),
     (fire1, occuredIn, "x ≥ 10 ∧ x ≤ 21 ∧ y ≥ 12 ∧ y ≤ 17")}
```

In order to be able to go from RDF$^{\mathrm{i}}$ databases to the equivalent set of possible RDF graphs, the notion of *valuation* is needed. Informally, a valuation maps an e-literal to a specific constant from C.

Definition 4.2. *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 valuation v to an e-triple t. $v(t)$ is obtained from t by replacing any e-literal $\_l$ appearing in t by $v(\_l)$ and leaving all other terms the same. If θ is a formula of $\mathcal{L}$ (e.g., the condition of a conditional triple or the global constraint of a database) then $v(\theta)$ denotes the application of v to formula θ. The expression $v(\theta)$ is obtained from θ by replacing all e-literals $\_l$ of θ by $v(\_l)$.

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

Definition 4.3. *Let G be a conditional graph and v a valuation. Then $v(G)$ denotes the RDF graph*

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

The set of valuations that satisfy the global constraint of an RDF$^{\mathrm{i}}$ 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 4.4. *Let $D = (G, \phi)$ be an* RDF$^{\mathrm{i}}$ *database. The set of RDF graphs corresponding to D is the following:*

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

In incomplete relational databases [12], *Rep* is a *semantic* function: it is used to map a table (a syntactic construct) to a set of relational instances (i.e., a set of possible words, a semantic construct). According to the well-known distinction between model theoretic and proof theoretic approaches to relational databases, *Rep* and the approaches based on it [12, 6] belong to the model theoretic camp. However, the use of function *Rep* in the above definition is different. *Rep* takes an RDF$^{\mathrm{i}}$ 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 [10]. This is a deliberate choice in our work since we want to explore which well-known tools from incomplete relational databases carry over to the RDF framework.

Notice that the definition of *Rep* above uses the containment relation instead of equality. The reason for this is to capture the OWA that the RDF model makes. By using the containment relation, $Rep(D)$ includes all graphs H containing at least the triples of $v(G)$. In this respect, we follow the approach of [4, 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 [4] 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 has not been studied in [4]; only the issue of OWA has been explored there.

The following notation will be useful below.

Notation 1. *Let $\mathcal{G}$ be a set of RDF graphs and q a SPARQL query. The expression $\bigcap \mathcal{G}$ will denote the set $\bigcap_{G \in \mathcal{G}} G$. The expression $[\![q]\!]_{\mathcal{G}}$, which extends the notation of [26] to the case of sets of RDF graphs, will denote the element-wise evaluation of q over $\mathcal{G}$, that is,*

$$[\![q]\!]_{\mathcal{G}} = \{[\![q]\!]_G \mid G \in \mathcal{G}\}.$$

Given the semantics of an RDF$^{\mathrm{i}}$ database as a set of possible RDF graphs, what is an appropriate definition for the answer to a certainty query? This is captured by the following definition of *certain answer* which extends the corresponding definition of Section 3.1 of [4] by applying it to a more general incomplete information setting.

Definition 4.5. *Let q be a query and $\mathcal{G}$ a set of RDF graphs. The* certain answer *to q over $\mathcal{G}$ is the set $\bigcap [\![q]\!]_{\mathcal{G}}$.*

Example 4.6. *Let us consider the following query over the database of Example 3.4: "Find all fires that have occurred in a region which is a non-tangential proper part of the rectangle defined by the points $(2, 4)$ and $(28, 22)$". The certain answer to this query is the set of mappings* $\{\{?\mathrm{F} \to \mathrm{fire1}\}\}$.

5. EVALUATING SPARQL ON RDF$^{\mathrm{i}}$ DATABASES

Let us now discuss how to evaluate SPARQL queries on RDF$^{\mathrm{i}}$ databases. We will use the algebraic syntax of SPARQL presented in [17]. We will consider only the monotone graph pattern fragment of SPARQL which uses only the AND, UNION, and FILTER operators [4]. 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 [12, 6]. The exact details will be given later in this section.

We use set semantics for query evaluation by extending the SPARQL query evaluation approach of [26]. 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 [10] 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 assumed to be disjoint from the set of terms T we defined in Section 3.

We first define the concept of *e*-mapping ("*e*" from the word "existential") which extends the concept of mapping of [17] with the ability to have an e-literal as value of a special query variable.

Definition 5.1. *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 $x \in V_s$.*

Example 5.2. *The following are e-mappings.*

$\mu_1 = \{\ ?\mathrm{F} \to \mathrm{fire1}, ?\mathrm{S} \to$ "$\mathrm{x} \geq 1 \wedge \mathrm{x} \leq 2 \wedge \mathrm{y} \geq 1 \wedge \mathrm{y} \leq 2$" $\}$
$\mu_2 = \{\ ?\mathrm{F} \to \mathrm{fire1}, ?\mathrm{S} \to \_\mathrm{R1}\ \}$
$\mu_3 = \{\ ?\mathrm{F} \to \mathrm{fire1}, ?\mathrm{S} \to \_\mathrm{R2}\ \}$
$\mu_4 = \{\ ?\mathrm{F} \to \mathrm{fire1}\ \}$

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

We now extend the concept of e-mapping and define conditional mappings, i.e., mappings that are equipped with a condition which constrains e-literals that appear in the e-mapping.

Definition 5.3. *A* conditional mapping *μ is a pair (ν, θ) where ν is an e-mapping and θ is a conjunction of $\mathcal{L}$-constraints.*

Example 5.4. *The following are conditional mappings.*

$\mu_1 = (\{?\mathrm{F} \to \mathrm{fire1}, ?\mathrm{S} \to$ "$\mathrm{x} \geq 1 \wedge \mathrm{x} \leq 2 \wedge \mathrm{y} \geq 1 \wedge \mathrm{y} \leq 2$"$\}, \mathrm{true})$
$\mu_2 = (\{?\mathrm{F} \to \mathrm{fire1}, ?\mathrm{S} \to \_\mathrm{R1}\},$
_R1 NTPP "$\mathrm{x} \geq 0 \wedge \mathrm{x} \leq 10 \wedge \mathrm{y} \geq 0 \wedge \mathrm{y} \leq 10$")
$\mu_3 = (\{?\mathrm{F} \to \mathrm{fire1}, ?\mathrm{S} \to \_\mathrm{R1}\},$
(_R1 NTPP _R2) $\wedge$
(_R2 DC "$\mathrm{x} \geq 0 \wedge \mathrm{x} \leq 1 \wedge \mathrm{y} \geq 0 \wedge \mathrm{y} \leq 1$"))
$\mu_4 = (\{?\mathrm{F} \to \mathrm{fire1}, ?\mathrm{S} \to \_\mathrm{R1}\}, \mathrm{true})$

Notice that conditional mappings with constraint *true*, such as μ_4 above, are logically equivalent to e-mappings.

The notions of domain and restriction for a conditional mapping are now defined as follows.

Definition 5.5. *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.*

Definition 5.6. *Let $\mu = (\nu, \theta)$ be a conditional mapping with domain S and $W \subseteq S$. The* restriction *of the mapping μ to W, denoted by $\mu_{|W}$, is the mapping $(\nu_{|W}, \theta)$ where $\nu_{|W}$ is the restriction of mapping ν to W.*

We now define the basic notion of triple pattern.

Definition 5.7. *A* triple pattern *is an element of the set $(I \cup V) \times (I \cup V) \times (I \cup L \cup C \cup U \cup V)$.*

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 new query variables.

If p is a triple pattern, $var(p)$ denotes the variables appearing in p. A conditional mapping can be applied to a triple pattern. 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 triple obtained from p by replacing each variable $x \in var(p)$ by $\nu(x)$.

We now introduce the notion of compatible conditional mappings as in [26].

Definition 5.8. *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 $\nu_1(x) = \nu_2(x)$.*

Example 5.9. *Mappings μ_1 and μ_2 from Example 5.4 are not compatible, while mappings μ_2 and μ_3 are.*

To take into account e-literals, we also need to define another notion of compatibility of two conditional mappings.

Definition 5.10. *Two conditional mappings $\mu_1 = (\nu_1, \theta_1)$ and $\mu_2 = (\nu_2, \theta_2)$ are* possibly compatible *if for all $x \in dom(\mu_1) \cap dom(\mu_2)$, we have $\nu_1(x) = \nu_2(x)$ or at least one of $\nu_1(x), \nu_2(x)$ where $x \in V_s$ is an e-literal from U.*

Example 5.11. *Conditional mappings μ_1, μ_2, and μ_3 from Example 5.4 are pairwise possibly compatible.*

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

Definition 5.12. *Let $\mu_1 = (\nu_1, \theta_1)$ and $\mu_2 = (\nu_2, \theta_2)$ be possibly compatible conditional mappings. The* join *$\mu_1 \bowtie \mu_2$ is a new conditional mapping (ν_3, θ_3) where:*

i. $\nu_3(x) = \nu_1(x) = \nu_2(x)$ *for each* $x \in dom(\mu_1) \cap dom(\mu_2)$ *such that* $\nu_1(x) = \nu_2(x)$.

ii. $\nu_3(x) = \nu_1(x)$ *for each* $x \in dom(\mu_1) \cap dom(\mu_2)$ *such that* $\nu_1(x)$ *is an e-literal and* $\nu_2(x)$ *is a literal from* C.

iii. $\nu_3(x) = \nu_2(x)$ *for each* $x \in dom(\mu_1) \cap dom(\mu_2)$ *such that* $\nu_2(x)$ *is an e-literal and* $\nu_1(x)$ *is a literal from* C.

iv. $\nu_3(x) = \nu_1(x)$ *for* $x \in dom(\mu_1) \cap dom(\mu_2)$ *such that both* $\nu_1(x)$ *and* $\nu_2(x)$ *are e-literals.*

v. $\nu_3(x) = \nu_1(x)$ *for* $x \in dom(\mu_1) \setminus dom(\mu_2)$.

vi. $\nu_3(x) = \nu_2(x)$ *for* $x \in dom(\mu_2) \setminus dom(\mu_1)$.

vii. θ_3 *is* $\theta_1 \wedge \theta_2 \wedge \xi_1 \wedge \xi_2 \wedge \xi_3$ *where:*

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

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

Example 5.13. *If μ_1 and μ_2 are the conditional mappings of Example 5.4, then:*

$\mu_1 \bowtie \mu_2 = (\{?\mathrm{F} \to \mathrm{fire1}, ?\mathrm{S} \to \_\mathrm{R1}\},$ true $\wedge$
_R1 EQ "$\mathrm{x} \geq 1 \wedge \mathrm{x} \leq 2 \wedge \mathrm{y} \geq 1 \wedge \mathrm{y} \leq 2$" $\wedge$
_R1 NTPP "$\mathrm{x} \geq 0 \wedge \mathrm{x} \leq 10 \wedge \mathrm{y} \geq 0 \wedge \mathrm{y} \leq 10$")

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 RDF [26]. The new thing 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 substituted 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:

$\Omega_1 \setminus \Omega_2 =$

$\{\mu_1 \in \Omega_1 \mid$ for all $\mu_2 \in \Omega_2, \mu_1$ and μ_2 are not compatible$\} \cup$
$\{(\nu, \theta') \mid \mu = (\nu, \theta) \in \Omega_1$ and $\mu_1 = (\nu_1, \theta_1), \ldots, \mu_n = (\nu_n, \theta_n)$
$\in \Omega_2$ such that $\mu_i,\ \mu$ are possibly compatible for all
$1 \leq i \leq n,$ and for every other $\mu_j \in \Omega_2$ different than
the μ_i's, μ_j and μ are not compatible. In this case, θ' is

$$\theta \bigwedge_i \left(\theta_i \supset \bigvee_x \left(\neg(\mu(x) \text{ EQ } \mu_i(x))\right)\right)$$

for every $x \in dom(\mu) \cap dom(\mu_i) \cap V_s$ and $1 \leq i \leq n\}$

The reader is invited to compare this definition with the definition of difference in [26]. The new thing in RDF^i is that we have to anticipate the possibility that a mapping μ from Ω_1 is not compatible with all the mappings of Ω_2 (i.e., it should be included in the difference) due to the presence of e-literals in it given some constraints. These constraints are added to the condition of μ.

EXAMPLE 5.14. *Let $\Omega_1 = \{\mu_{11}, \mu_{12}\}$, $\Omega_2 = \{\mu_{21}, \mu_{22}\}$ be sets of conditional mappings such that*

$\mu_{11} = (\{?\text{F} \to \text{fire1}, ?\text{S} \to \_\text{R1}\},$
$\quad \_\text{R1 NTPP "x} \geq 0 \wedge \text{x} \leq 10 \wedge \text{y} \geq 0 \wedge \text{y} \leq 10\text{"})$

$\mu_{12} = (\{?\text{F} \to \text{fire1}, ?\text{S} \to \text{"x} \geq 1 \wedge \text{x} \leq 2 \wedge \text{y} \geq 1 \wedge \text{y} \leq 2\text{"}\},$
$\quad \_\text{R1 PO "x} \geq 0 \wedge \text{x} \leq 10 \wedge \text{y} \geq 0 \wedge \text{y} \leq 10\text{"})$

$\mu_{21} = (\{?\text{F} \to \text{fire2}\},\ \text{true})$

$\mu_{22} = (\{?\text{F} \to \text{fire1}, ?\text{S} \to \text{"x} \geq 1 \wedge \text{x} \leq 2 \wedge \text{y} \geq 1 \wedge \text{y} \leq 2\text{"}\}, \text{true})$

Then, $\Omega_1 \setminus \Omega_2 = \{\mu\}$ where μ has been constructed from $\mu_{11} \in \Omega_1$ and is the following mapping:

$\mu = (\{?\text{F} \to \text{fire1}, ?\text{S} \to \_\text{R1}\},$
$\quad (\_\text{R1 NTPP "x} \geq 0 \wedge \text{x} \leq 10 \wedge \text{y} \geq 0 \wedge \text{y} \leq 10\text{"}) \wedge$
$\quad \neg(\_\text{R1 EQ "x} \geq 1 \wedge \text{x} \leq 2 \wedge \text{y} \geq 1 \wedge \text{y} \leq 2\text{"})\ )$

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

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

It has been noted in [26] that the OPT operator of SPARQL (the counterpart of the left-outer join algebraic operator) can be used to express difference in SPARQL. For data models that make the OWA, such an operator is unnatural since negative information cannot be expressed. However, we deliberately include operator OPT because if it is combined with operators AND and FILTER under certain syntactic restrictions, it turns out that the resulting graph patterns cannot express a difference operator anymore [4]. In particular, the class of graph patterns produced by this syntactic restriction are known as *well-designed graph patterns*. Well-designed graph patterns are discussed in more depth in Section 6 where representation systems for RDF^i are investigated.

We can now define the result of evaluating a graph pattern over an RDF^i database (the definition of graph patterns is omitted). Given the previous operations on sets of mappings, graph pattern evaluation in RDF^i can now be defined exactly as in standard SPARQL for RDF graphs [26] except for the case of evaluating a triple pattern.

DEFINITION 5.15. *Let $D = (G, \phi)$ be an RDF^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 we have two cases. If o is a literal from the set C then*

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

else

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

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 the first item of the above definition the "else" part is to accommodate the case in which evaluation can be done as in standard SPARQL. This is the case in which the object part of the triple pattern is not a literal from C. The "if" part accommodates the case in which the triple pattern involves a literal o from the set C. Here, there are two alternatives: the graph contains a conditional triple matching with every component of the triple pattern (i.e., a triple which has o in the object position) or it contains a conditional triple with an e-literal $\_l$ from U in the object position. We catch a possible match for the second case by adding in the condition of the mapping the constraint that restricts the value of e-literal $\_l$ to be equal to the literal o of the triple pattern (i.e., the constraint $\_l$ EQ o). In all cases of the first item of the above definition, since the triples in the database are conditional, their conditions become parts of the conditions of the mappings in the answer.

EXAMPLE 5.16. *Let us first give two examples for the evaluation of triple patterns over the database D of Example 3.5.*

$[\![(?\text{F, occuredIn, "x} \geq 1 \wedge \text{x} \leq 2 \wedge \text{y} \geq 1 \wedge \text{y} \leq 2\text{"})]\!]_\text{D} = \{\ \} \cup$
$\{(\{?\text{F} \to \text{fire1}\}, \_\text{R1 EQ "x} \geq 1 \wedge \text{x} \leq 2 \wedge \text{y} \geq 1 \wedge \text{y} \leq 2\text{"})\}$

$[\![(?\text{F, occuredIn, "x} \geq 6 \wedge \text{x} \leq 23 \wedge \text{y} \geq 8 \wedge \text{y} \leq 19\text{"})]\!]_\text{D} =$
$\{(\{?\text{F} \to \text{fire2}\},\ \text{true})\} \cup \{(\{?\text{F} \to \text{fire1}\},$
$\quad \_\text{R1 EQ "x} \geq 6 \wedge \text{x} \leq 23 \wedge \text{y} \geq 8 \wedge \text{y} \leq 19\text{"})\}$

These examples correspond to the "if" part of the first item of Definition 5.15 in which the triple pattern involves a literal from the set C.

EXAMPLE 5.17. *Let us now give an example of an evaluation of graph pattern P_1 AND P_2 over the database D of Example 3.4, where P_1, P_2 are the triple patterns $(?\text{F}, \text{type}, \text{Fire})$ and $(?\text{F}, \text{occuredIn}, ?\text{R})$ respectively. According to the above definition, we have:*

$[\![(\text{P}_1 \text{ AND P}_2)]\!]_\text{D} = [\![\text{P}_1]\!]_\text{D} \bowtie [\![\text{P}_2]\!]_\text{D} =$
$[\![(?\text{F, type, Fire})]\!]_\text{D} \bowtie [\![(?\text{F, occuredIn, ?R})]\!]_\text{D} =$
$\{(\{?\text{F} \to \text{fire1}\}, \text{true})\} \bowtie \{(\{?\text{F} \to \text{fire1}, ?\text{R} \to \_\text{R1}\}, \text{true})\} =$
$\quad \{(\{?\text{F} \to \text{fire1}, ?\text{R} \to \_\text{R1}\}, \text{true})\}$

The evaluation of both triple patterns P_1, P_2 corresponds to the "else" part of the first item of Definition 5.15. In this case evaluation 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 conjunctions of $\mathcal{L}$-constraints as expressions that constrain query variables, e.g., constraints like ?X NTPP ?Y or ?X EQ "$\mathrm{x} \geq 1 \wedge \mathrm{x} \leq 2 \wedge \mathrm{y} \geq 1 \wedge \mathrm{y} \leq 2$" when $\mathcal{L}$ is PCL as in our examples.

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 [26], but simply imposes these constraints on the mappings that are in the answer of the graph pattern involved.

Definition 5.18. *Given an* RDF$^{\mathrm{i}}$ *database* $D = (G, \phi)$*, a graph pattern* P *and a conjunction of* $\mathcal{L}$*-constraints* R*, we have:*

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

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

The extension of FILTER to the case that R is a Boolean combination of $\mathcal{L}$-constraints is now easy to define and is omitted. Similarly, the extension of FILTER to the case that R contains also other built-in conditions of standard SPARQL [26] is easy to define and is omitted as well.

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. Thus, evaluation of FILTERs is "lazy". In an implementation, one can also simplify constraints at this stage; such issues are beyond the scope of this paper.

Example 5.19. *Based on the evaluation of the graph pattern of Example 5.17, the evaluation of the graph pattern* $((P_1\ AND\ P_2)\ FILTER\ R)$*, where* R *is the PCL-constraint* (?R NTPP "$\mathrm{x} \geq 10 \wedge \mathrm{x} \leq 21 \wedge \mathrm{y} \geq 12 \wedge \mathrm{y} \leq 17$")*, is the following:*

$$\begin{aligned}&[\![(\mathrm{P}_1\ \ \mathrm{AND}\ \ \mathrm{P}_2)\ \ \mathrm{FILTER}\ \ \mathrm{R}]\!]_\mathrm{D} = \\ &[\![((?\mathrm{F}, \mathrm{type}, \mathrm{Fire})\ \ \mathrm{AND}\ \ (?\mathrm{F}, \mathrm{occuredIn}, ?\mathrm{R}))\ \ \mathrm{FILTER} \\ &(?\mathrm{R}\ \mathrm{NTPP}\ \ "\mathrm{x} \geq 10 \wedge \mathrm{x} \leq 21 \wedge \mathrm{y} \geq 12 \wedge \mathrm{y} \leq 17")]\!]_\mathrm{D} = \\ &\{(\{?\mathrm{F} \rightarrow \mathrm{fire1}, ?\mathrm{R} \rightarrow \_\mathrm{R1}\}, \\ &\qquad \_\mathrm{R1}\ \mathrm{NTPP}\ \ "\mathrm{x} \geq 10 \wedge \mathrm{x} \leq 21 \wedge \mathrm{y} \geq 12 \wedge \mathrm{y} \leq 17")\}\end{aligned}$$

The next definition defines the concept of a SELECT query [26].

Definition 5.20. *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 5.21. *Let us consider the following query over the database of Example 3.4: "Find all fires that have occurred in a region which is a non-tangential proper part of rectangle defined by the points* $(10, 12)$ *and* $(21, 17)$*". This query can be expressed as follows:*

$$\begin{aligned}&(\{?\mathrm{F}\}, (?\mathrm{F}, \mathrm{type}, \mathrm{Fire})\ \ \mathrm{AND}\ \ (?\mathrm{F}, \mathrm{occuredIn}, ?\mathrm{R}) \\ &\qquad \mathrm{FILTER}\ (?\mathrm{R}\ \mathrm{NTPP}\ \ "\mathrm{x} \geq 10 \wedge \mathrm{x} \leq 21 \wedge \mathrm{y} \geq 12 \wedge \mathrm{y} \leq 17"))\end{aligned}$$

The next definition defines the notion of answer to a SELECT query. In contrast to SELECT queries over RDF graphs, SELECT queries over RDF$^{\mathrm{i}}$ databases have answers that consist of conditional mappings so they might be harder to understand.

Definition 5.22. *Let* $q = (W, P)$ *be a* SELECT *query. The* answer *to* q *over an* RDF$^{\mathrm{i}}$ *database* $D = (G, \phi)$ *(in symbols* $[\![q]\!]_D$*) is the set of conditional mappings* $\{\mu_{|W} \mid \mu \in [\![P]\!]_D\}$*.*

The conditional mappings of the answer to a query might contain e-literals. These literals are constrained by the global constraint ϕ, therefore ϕ can be understood to be implicitly included in the answer (this can also be done formally by considering answers to be pairs).

Example 5.23. *The answer to the query from Example 5.21 can be obtained from the evaluation of the respective graph pattern from Example 5.19. The answer is a set that contains only the following mapping:*

$$(\{?\mathrm{F} \rightarrow \mathrm{fire1}\}, \_\mathrm{R1}\ \mathrm{NTPP}\ \ "\mathrm{x} \geq 10 \wedge \mathrm{x} \leq 21 \wedge \mathrm{y} \geq 12 \wedge \mathrm{y} \leq 17"\ )$$

This answer is conditional. Because the information in the database of Example 3.4 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.*

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

Definition 5.24. *A template* E *is a finite subset of the set* $(T \cup V) \times (I \cup V) \times (T \cup V)$*.*

Thus, the elements of a template are like triple patterns but blank nodes 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.

Definition 5.25. *A* CONSTRUCT *query is a pair* (E, P) *where* E *is a template and* P *a graph pattern.*

Example 5.26. *Let us consider the query of Example 5.21. A new version of this query using the* CONSTRUCT *query form is:*

$$\begin{aligned}&(\{(?\mathrm{F}, \mathrm{type}, \mathrm{Fire})\}, (?\mathrm{F}, \mathrm{type}, \mathrm{Fire})\ \ \mathrm{AND}\ \ (?\mathrm{F}, \mathrm{occuredIn}, ?\mathrm{R}) \\ &\qquad \mathrm{FILTER}\ (?\mathrm{R}\ \mathrm{NTPP}\ \ "\mathrm{x} \geq 10 \wedge \mathrm{x} \leq 21 \wedge \mathrm{y} \geq 12 \wedge \mathrm{y} \leq 17"))\end{aligned}$$

Next we define what it means for a conditional mapping to be applied to a template.

Definition 5.27. *Let* $\mu = (\nu, \theta)$ *be a conditional mapping and* E *a template. We denote by* $\mu(E)$ *the application of conditional mapping* μ *to template* E*.* $\mu(E)$ *is obtained from* E *by replacing in* E *every variable* x *of* $var(E) \cap dom(\mu)$ *by* $\nu(x)$*.*

Templates are used to specify the graph that results from the evaluation of a CONSTRUCT query.

Example 5.28. *Let us consider the template* $E = \{(?F, type, ?Z), (?F, occuredIn, ?S)\}$ *and mapping* μ_4 *from Example 5.4. The result of applying* μ_4 *to* E *is the following set:*

$$\{(\mathrm{fire1}, \mathrm{type}, ?\mathrm{Z}), (\mathrm{fire1}, \mathrm{occuredIn}, \_\mathrm{R1})\}$$

Notice that Definition 5.27 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 e-triple, i.e., it is not an element of the set $(I \cup B) \times I \times T$. Such a triple is dropped from the answer to a CONSTRUCT *query (see Definition 5.30 below).*

Next we define the concept of answer to a CONSTRUCT query. The definition extends the specification of standard SPARQL [27] to account for the RDF^i framework and follows the formal approach of [25]. Before we give the definition, we need to introduce the notion of *renaming function*.

DEFINITION 5.29. *Let E be a template, P a graph pattern, and $D = (G, \phi)$ an* RDF^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.*

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 5.30. *Let $q = (E, P)$ be a* CONSTRUCT *query, $D = (G, \phi)$ an* RDF^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^i *database $D' = (G', \phi)$ where*

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

In the above definition, renaming functions are used to ensure that brand new blank nodes are created for each conditional mapping μ. The intersection with $(I \cup B) \times I \times T$ makes sure that no illegal triples are returned as answers (see Example 5.28 above).

EXAMPLE 5.31. *The answer to the* CONSTRUCT *query from Example 5.26 can be obtained from the evaluation of the respective graph pattern from Example 5.19. The answer is the following* RDF^i *database:*

```
( { ((fire1, type, Fire),
          _R1 NTPP  "x ≥ 10 ∧ x ≤ 21 ∧ y ≥ 12 ∧ y ≤ 17") },

  _R1 NTPP  "x ≥ 6 ∧ x ≤ 23 ∧ y ≥ 8 ∧ y ≤ 19" )
```

6. REPRESENTATION SYSTEMS FOR RDF^i

Let us now recall the semantics of RDF^i as given by *Rep*. $Rep(D)$ is the set of possible RDF graphs corresponding to an RDF^i database D. Clearly, if we were to evaluate a query q over D, we could use the semantics of RDF^i and evaluate q over any RDF graph of $Rep(D)$ as follows:

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

However, this is not the best answer we wish to have in terms of representation; we queried an RDF^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^i database as the output. Thus, we are interested in finding an RDF^i database $[\![q]\!]_D$ representing the answer $[\![q]\!]_{Rep(D)}$. This requirement is translated to the following formula:

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

Formula (1) allows us to compute the answer to any query over an RDF^i database in a consistent way with respect to the semantics of RDF^i without having the need to apply the query on all possible RDF graphs. $[\![q]\!]_D$ can be computed using the algebra of Section 5 above. But can the algebra of Section 5 compute always such a database $[\![q]\!]_D$ representing $[\![q]\!]_{Rep(D)}$? In other words, can we prove (1) for all SPARQL queries considered in Section 5? The answer is *no* in general. The following example modelled after [7] illustrates this negative fact.

EXAMPLE 6.1. *Consider the* RDF^i *database $D = (G, \phi)$, where $G = \{((s, p, o), true)\}$ and $\phi = true$, i.e., D contains the single triple (s, p, o) where $s, p, o \in I$. Consider now a* CONSTRUCT *query q over D that selects all triples having s as the subject. The algebraic version of query q would be $(\{(s, ?p, ?o)\}, (s, ?p, ?o))$ and evaluated as $[\![q]\!]_D$ using Definition 5.30. Then, the triple (s, p, o) and nothing else is in the resulting database $[\![q]\!]_D$. However, equation (1) is not satisfied, since for instance (c, d, e) occurs in some $g \in Rep([\![q]\!]_D)$ according to the definition of Rep, whereas $(c, d, e) \notin g$ for all $g \in [\![q]\!]_{Rep(D)}$.*

Note that the above counterexample to (1) exploits only the fact that RDF makes the OWA. In other words, the counterexample would hold for any approach to incomplete information in RDF which respects the OWA. Thus, unless the CWA is adopted, which we do not want to do since we are in the realm of RDF, condition (1) has to be relaxed[1].

In the rest of this section we follow the literature of incomplete information [12, 6] and show how (1) can be weakened. The key concept for achieving this is the concept of certain answer we defined earlier. Given a fixed fragment of SPARQL $\mathcal{Q}$, two RDF^i databases cannot be distinguished by $\mathcal{Q}$ if they give the same certain answer to every query in $\mathcal{Q}$. The next definition formalizes this fact using the concept of $\mathcal{Q}$-equivalence. Originally this concept was defined for incomplete relational databases in [12].

DEFINITION 6.2. *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 answer 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^i database instead of using the set of possible graphs given by *Rep*. The definition of representation system (originally defined in [12] for incomplete relational databases) corresponds to the notion of *weak query system* defined in the same context by [6].

DEFINITION 6.3. *Let $\mathcal{D}$ be the set of all* RDF^i *databases, $\mathcal{G}$ the set of all RDF graphs, $Rep : \mathcal{D} \to \mathcal{G}$ a function determining the set of possible RDF graphs corresponding to an*

[1]If the CWA is adopted, we can prove (1) using similar techniques to the ones that enable us to prove Theorem 6.14 below.

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 the following condition is satisfied:*

$$Rep([\![q]\!]_D) \equiv_{\mathcal{Q}} [\![q]\!]_{Rep(D)}$$

The next step towards the development of a representation system for RDF$^{\text{i}}$ and SPARQL is to introduce various fragments of SPARQL that we will consider and define the notions of *monotonicity* and *coinitiality* as is done in [12]. As in Section 5, our only addition to standard SPARQL is the extension of FILTERs with another kind of conditions that are constraints of $\mathcal{L}$. We also consider the fragment of SPARQL graph patterns known as *well-designed*. Well-designed graph patterns form a practical fragment of SPARQL graph patterns that include the OPT operator and it has been showed in [26, 4] that that they have nice properties, such as lower combined complexity than in the general case, a normal form which is useful for optimization, and they are also weakly monotone. Thus, it is worth studying them in the context of RDF$^{\text{i}}$. Section B of the Appendix contains formal definitions and relevant background results for well-designed graph patterns.

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

The following definition introduces the concept of monotone fragments of SPARQL applied to RDF graphs. Then, Proposition 6.5 give us some fragments of SPARQL that are monotone.

Definition 6.4. *A fragment* $\mathcal{Q}$ *of SPARQL is* monotone *if for every* $q \in \mathcal{Q}$ *and RDF graphs* G *and* H *such that* $G \subseteq H$*, it is* $[\![q]\!]_G \subseteq [\![q]\!]_H$*.*

Proposition 6.5. *The following results hold with respect to the monotonicity of SPARQL: a) Language* $\mathcal{Q}^S_{AUF}$ *is monotone. b) The presence of OPT or* CONSTRUCT *makes a fragment of SPARQL not monotone. c) Language* $\mathcal{Q}^{C'}_{AUF}$ *is monotone. d) Language* $\mathcal{Q}^{C'}_{WD}$ *is monotone.*

Parts $a) - c)$ of the above proposition are trivial extensions of relevant results in [4]. However, part $d)$ is an interesting result showing that the weak monotonicity property of well-designed graph patterns suffices to get a monotone fragment of SPARQL containing the OPT operator, i.e., the class of CONSTRUCT queries without blank nodes in their templates. This is a result that cannot be established for the case of SELECT queries and with this respect CONSTRUCT queries deserve closer attention.

Monotonicity is a sufficient property for establishing our results about representation systems. Thus, in the following, we focus on the monotone query languages $\mathcal{Q}^{C'}_{AUF}$ and $\mathcal{Q}^{C'}_{WD}$.

Definition 6.6. *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 any* $G \in \mathcal{G}$ *there exists* $H \in \mathcal{H}$ *such that* $H \subseteq G$*, and for any* $H \in \mathcal{H}$ *there exists* $G \in \mathcal{G}$ *such that* $G \subseteq H$*.*

Example 6.7. *The following sets are coinitial.*

$$\mathcal{G} = \{\{(a,b,c),(a,e,d),(a,f,g)\},\{(a,b,c),(a,e,d)\},\{(a,b,c)\}\}$$
$$\mathcal{H} = \{\{(a,b,c),(a,e,d)\},\{(a,b,c)\}\}$$

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 $\bigcap \mathcal{G} = \bigcap \mathcal{H} = \{(a,b,c)\}$.

Proposition 6.8. *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 any* $q \in \mathcal{Q}$*, it holds that* $[\![q]\!]_{\mathcal{G}} \approx [\![q]\!]_{\mathcal{H}}$*.*

Lemma 6.9. *Let* $\mathcal{G}$ *and* $\mathcal{H}$ *be sets of RDF graphs. If* $\mathcal{G}$ *and* $\mathcal{H}$ *are coinitial then* $\mathcal{G} \equiv_{\mathcal{Q}^{C'}_{AUF}} \mathcal{H}$*.*

We will now present our main theorem which characterizes the evaluation of monotone $\mathcal{Q}^{C'}_{AUF}$ and $\mathcal{Q}^{C'}_{WD}$ queries (Theorem 6.14). 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. By applying a valuation to a conditional mapping, we get an *ordinary mapping* like in the case of RDF simply by disregarding the constraint that results since it is equivalent to *true*.

Definition 6.10. *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 results from substituting in e-mapping* ν *the constant* $v(\_l)$ *for each e-literal* $\_l$*.*

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

Definition 6.11. *Let* Ω *be a set of conditional mappings and* $v : U \to C$ *a valuation. Then*

$$v(\Omega) = \{v(\mu) \mid \mu = (\nu,\theta) \in \Omega \textit{ and } \mathbf{M}_{\mathcal{L}} \models v(\theta)\}.$$

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

Definition 6.12. *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 6.13. *Let* $D = (G,\phi)$ *be an* RDF$^{\text{i}}$ *database,* q *a query from a monotone fragment* $\mathcal{Q}$ *of SPARQL, and* v *a valuation such that* $\mathbf{M}_{\mathcal{L}} \models v(\phi)$*. Then,* $v([\![q]\!]_D) = [\![q]\!]_{v(D)}$ *implies* $Rep([\![q]\!]_D) \approx [\![q]\!]_{Rep(D)}$*.*

We are now ready to prove our main result.

Theorem 6.14. *The triples* $\langle \mathcal{D}, Rep, \mathcal{Q}^{C'}_{AUF}\rangle$ *and* $\langle \mathcal{D}, Rep, \mathcal{Q}^{C'}_{WD}\rangle$ *are representation systems.*

Since SELECT queries in SPARQL take as input an RDF graph but return a set of mappings (i.e., we do not have closure), it is not clear how to include them in the developed concept of a representation system (but see the discussion about SELECT in Section 7 below).

7. CERTAIN ANSWER COMPUTATION

This section studies how the certain answer to a SPARQL query q over an RDF$^{\text{i}}$ database D can be computed, i.e., how to compute $\bigcap [\![q]\!]_{Rep(D)}$. Having Theorem 6.14, it is easy to compute the certain answer to a query in the fragment of SPARQL $\mathcal{Q}^{C'}_{AUF}$ or $\mathcal{Q}^{C'}_{WD}$. Since $\langle \mathcal{D}, Rep, \mathcal{Q}^{C'}_{AUF} \rangle$ and $\langle \mathcal{D}, Rep, \mathcal{Q}^{C'}_{WD} \rangle$ are representation systems, we can apply Definition 6.2 for the identity query to get $\bigcap [\![q]\!]_{Rep(D)} = \bigcap Rep([\![q]\!]_D)$ for all q and D. Thus, we can equivalently compute $\bigcap Rep([\![q]\!]_D)$ where $[\![q]\!]_D$ can be computed using the algebra of Section 5.

Before presenting the algorithm for certain answer computation, we need to introduce some auxiliary constructs similar to the ones defined in [12, 6] in the case of incomplete relational databases.

Definition 7.1. *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 $\phi \models \_l$ EQ c (if such a constant exists).*

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

Definition 7.2. *Let $D = (G, \phi)$ be an* RDF$^{\text{i}}$ *database. The* normalized *form of D is the* RDF$^{\text{i}}$ *database $D^* = (G^*, \phi)$ where G^* is the set*

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

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. Notice that these new conditional triples do not follow Definition 3.1 which assumes conditions to be conjunctions of $\mathcal{L}$-constraints. We will allow this deviation from Definition 3.1 in this section.

Lemma 7.3. *Let $D = (G, \phi)$ be an* RDF$^{\text{i}}$ *database. Then:*

$$\bigcap Rep(D) = \bigcap Rep((D^{\text{EQ}})^*)$$

Having Lemma 7.3, it is easy to give an algorithm that computes the certain answer to a query.

Theorem 7.4. *Let $D = (G, \phi)$ be an* RDF$^{\text{i}}$ *database and q a query from $\mathcal{Q}^{C'}_{AUF}$ or $\mathcal{Q}^{C'}_{WD}$. The certain answer of q over D can be computed as follows: i) compute $[\![q]\!]_D$ according to Section 5 and let $D_q = (G_q, \phi)$ be the resulting* RDF$^{\text{i}}$ *database, ii) compute the* RDF$^{\text{i}}$ *database $(H_q, \phi) = ((D_q)^{\text{EQ}})^*$, and iii) return the following set of RDF triples:*

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

Let us now present a preliminary analysis of the data complexity of computing the certain answer to a CONSTRUCT query over an RDF$^{\text{i}}$ database when $\mathcal{L}$ is a constraint language. Following [6], we first define the corresponding decision problem.

Definition 7.5. *Let q be a* CONSTRUCT *query. The* certainty problem *for query q, RDF graph H, and* RDF$^{\text{i}}$ *database D, is to decide whether $H \subseteq \bigcap [\![q]\!]_{Rep(D)}$. We denote this problem by $CERT_C(q, H, D)$.*

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

Theorem 7.6. *Let $D = (G, \phi)$ be an* RDF$^{\text{i}}$ *database, q a query from $\mathcal{Q}^{C'}_{AUF}$ or $\mathcal{Q}^{C'}_{WD}$, and H an RDF graph. Then, $CERT_C(q, H, D)$ is equivalent to deciding whether the following formula is true in $\mathbf{M}_{\mathcal{L}}$:*

$$\bigwedge_{t \in H} (\forall \_\mathbf{l})(\phi(\_\mathbf{l}) \supset \Theta(t, q, D, \_\mathbf{l})) \qquad (2)$$

In the above formula:

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

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

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 6 that are needed. This involves first modifying Definition 6.2 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 6.3, 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 C.1.

7.1 Data Complexity Results

The data complexity of the certainty problem, $CERT_C(q, H, D)$, for q in the $\mathcal{Q}^{C'}_{AUF}$ fragment of SPARQL and D in the set of RDF$^{\text{i}}$ databases with constraints from ECL, diPCL, dePCL, and RCL is coNP-complete. This follows easily from known results of [6] for ECL and [14, 16, 35] for diPCL, dePCL, and RCL. Thus, we have the expected increase in data complexity given that the complexity of evaluating AND, UNION, and FILTER graph patterns over RDF graphs can be done in LOGSPACE [26].

Theorem 7.6 gives us immediately some easy upper bounds on the data complexity of the certainty problem in the case of RDF$^{\text{i}}$ with $\mathcal{L}$ equal to TCL or PCL. The satisfiability problem for conjunctions of TCL-constraints is known to be in PTIME [29]. Thus, the entailment problems arising in Theorem 7.6 can be trivially solved in EXPTIME. Therefore, the certainty problem is also in EXPTIME. To the best of our knowledge, no better bounds are known in the literature of TCL that we could have used to achieve a tighter bound for the certainty problem as we have done with the languages of the previous paragraph.

[20] shows that conjunctions of atomic RCC-5 constraints involving constants that are polygons in V-representation (called landmarks in [20]) can be decided in PTIME. Therefore, by restricting PCL so that only RCC-5 constraints are allowed and constants are given in V-representation, the certainty problem in this case is also in EXPTIME.

8. RELATED WORK

Incomplete information has been studied in-depth in relational databases starting with the paper of [12]. More recently, papers on uncertain [30, 3] and probabilistic [33] database models have reignited interest in this area.

In the context of the Web, incomplete information has been studied in detail for XML [2, 5]. Related work for incomplete information in RDF [9, 11, 4] has been discussed in the introduction, so we do not repeat the details here. The study of incomplete information in RDF undertaken in this paper goes beyond [4] where only the issue of OWA for RDF is investigated. Other cases of incomplete information in RDF (e.g., blank nodes according to the W3C RDF semantics which is different than the SPARQL semantics as we pointed out in Section 5) can also be investigated using an approach similar to ours. Comparing our work with [9, 11], 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 [11] is similar to ours since it is based on constraints, but, whereas we concentrate on query answering for RDF^{i}, [11] concentrates more on semantic issues such as temporal graph entailment. It is easy to see that RDF^{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 [15]. 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 in [9, 11].

Recently, some papers have started studying the problem of representing probabilistic information in RDF and querying it using SPARQL [34, 19]. It would be interesting to investigate how these approaches can be combined with the work presented in this paper as [8] has done in the model of probabilistic c-tables.

It is interesting to compare the expressive power that RDF^{i} gives us to other recent works that use Semantic Web data models and languages for geospatial applications. When equipped with a constraint language like TCL, PCL, or RCL, RDF^{i} goes beyond the proposals of the geospatial extensions of SPARQL, stSPARQL [18] and GeoSPARQL [23] that cannot query incomplete geospatial information. While GeoSPARQL provides a vocabulary for asserting topological relations (the topology vocabulary extension), the 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 geospatial information as it is studied in this paper can also be expressed in spatial description logics [24, 21]. For efficiency reasons, spatial DL reasoners such as RacerPro[2] and PelletSpatial [32] 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.

9. FUTURE WORK

Our future work focuses on the following: 1) explore other fragments of SPARQL that can be used to define a representation system for RDF^{i}, 2) study in more depth the complexity of certain answer computation for the various spatial and temporal constraint languages $\mathcal{L}$ we considered or the one used in [11] and identify tractable classes, and 3) study the complexity of evaluating various fragments of SPARQL over RDF^{i} databases like it has been done in [26, 31] for the case of SPARQL and RDF.

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

10. REFERENCES

[1] S. Abiteboul, P. Kanellakis, and G. Grahne. On the Representation and Querying of Sets of Possible Worlds. In *SIGMOD*, pages 34–48, 1987.

[2] S. Abiteboul, L. Segoufin, and V. Vianu. Representing and querying XML with incomplete information. *ACM TODS*, 31(1):208–254, 2006.

[3] L. Antova, C. Koch, and D. Olteanu. $10^{(10^6)}$ worlds and beyond: efficient representation and processing of incomplete information. *VLDBJ*, 18(5):1021–1040, 2009.

[4] M. Arenas and J. Pérez. Querying semantic web data with SPARQL. In *PODS*, pages 305–316, 2011.

[5] P. Barceló, L. Libkin, A. Poggi, and C. Sirangelo. XML with incomplete information. *JACM*, 58(1):4, 2010.

[6] G. Grahne. *The Problem of Incomplete Information in Relational Databases*. LNCS. Springer Verlag, 1991.

[7] G. Grahne. Incomplete information. In *Encyclopedia of Database Systems*, pages 1405–1410. Springer, 2009.

[8] T. J. Green and V. Tannen. Models for incomplete and probabilistic information. In *EDBT Workshops*, 2006.

[9] C. Gutierrez, C. A. Hurtado, and A. A. Vaisman. Introducing Time into RDF. *IEEE TKDE*, 19(2), 2007.

[10] P. Hayes. RDF Semantics, W3C Recommendation, 2004.

[11] C. A. Hurtado and A. A. Vaisman. Reasoning with Temporal Constraints in RDF. In *PPSWR*. Springer, 2006.

[12] T. Imielinski and W. Lipski. Incomplete Information in Relational Databases. *JACM*, 31(4):761–791, 1984.

[13] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint Query Languages. In *PODS*, 1990.

[14] M. Koubarakis. Complexity results for first-order theories of temporal constraints. In *KR*, pages 379–390, 1994.

[15] M. Koubarakis. Database models for infinite and indefinite temporal information. *Inf. Syst.*, 19(2):141–173, 1994.

[16] M. Koubarakis. The complexity of query evaluation in indefinite temporal constraint databases. *Theor. Comput. Sci.*, 171(1-2):25–60, 1997.

[17] M. Koubarakis and K. Kyzirakos. Modeling and querying metadata in the semantic sensor web: The model stRDF and the query language stSPARQL. In *ESWC*, 2010.

[18] K. Kyzirakos, M. Karpathiotakis, and M. Koubarakis. Strabon: A Semantic Geospatial DBMS. In *ISWC'12*, pages 295–311, 2012.

[19] X. Lian and L. Chen. Efficient query answering in probabilistic rdf graphs. In *SIGMOD*, pages 157–168,

2011.

[20] W. Liu, S. Wang, S. Li, and D. Liu. Solving qualitative constraints involving landmarks. In *CP*, 2011.

[21] C. Lutz and M. Miličić. A tableau algorithm for description logics with concrete domains and general tboxes. *J. Autom. Reason.*, 38:227–259, April 2007.

[22] C. Nikolaou and M. Koubarakis. Querying Linked Geospatial Data with Incomplete Information. In *5th International Terra Cognita Workshop - Foundations, Technologies and Applications of the Geospatial Web*, Boston, USA, 2012.

[23] Open Geospatial Consortium Inc. GeoSPARQL - A geographic query language for RDF data. OGC, 2010.

[24] Ö. Özcep and R. Möller. Computationally feasible query answering over spatio-thematic ontologies. In *GEOProcessing*, 2012.

[25] J. Pérez, M. Arenas, and C. Gutierrez. Semantics of SPARQL. Technical report, Univ. de Chile, 2006.

[26] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM TODS*, 34(3):1–45, 2009.

[27] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Recommendation 15 Jan. 2008.

[28] D. A. Randell, Z. Cui, and A. G. Cohn. A spatial logic based on regions and connection. In *KR*, 1992.

[29] J. Renz and B. Nebel. On the complexity of qualitative spatial reasoning: A maximal tractable fragment of the region connection calculus. *AIJ*, 108(1-2):69–123, 1999.

[30] A. D. Sarma, O. Benjelloun, A. Y. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, 2006.

[31] M. Schmidt, M. Meier, and G. Lausen. Foundations of sparql query optimization. In *ICDT*, pages 4–33, 2010.

[32] M. Stocker and E. Sirin. PelletSpatial: A Hybrid RCC-8 and RDF/OWL Reasoning and Query Engine. In *OWLED*, 2009.

[33] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Morgan & Claypool Publishers, 2011.

[34] O. Udrea, V. S. Subrahmanian, and Z. Majkic. Probabilistic RDF. In *IRI*, pages 172–177, 2006.

[35] R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. In *PODS*, pages 331–345, 1992.

APPENDIX

The Appendix is structured as follows. In Section A we formally define a number of constraint languages for modeling information in geospatial and temporal domains. These languages have been already defined informally in Section 2. Then, Section B gives formal definitions for the concept of *well-designed* graph patterns and relevant concepts, such as *subsumption for mappings* and *weak monotonicity*, while it presents known results for well-designed patterns. These results are useful for establishing the monotonicity results for the fragment of SPARQL corresponding to CONSTRUCT queries with well-designed graph patterns and without blank nodes in their templates. Section C provides the proofs for the section of Representation Systems (Section 6), while Section D provides the proofs for the section of Certain Answer Computation (Section 7). Last, Section E is devoted to additional propositions that are useful to establish some propositions and/or theorems of Section 6.

A. CONSTRAINT LANGUAGES

In this section we define formally all the constraint languages used in the paper.

A.1 The Language ECL

The language *ECL* (*E*quality *C*onstraint *L*anguage) with predicate symbol = and an infinite number of constants has been defined in [13]. The intended structure for this language interprets symbol = as equality and constants as "themselves". An ECL-constraint is an ECL formula of the form $x_1 = x_2$ where x_1, x_2 are variables or constants.

ECL has been used by [13] for the development of an extended relational model based on ECL-constraints and by [12, 1, 6] for querying and updating incomplete information in relational databases.

The following two languages are from [16].

A.2 The Language diPCL

The language *diPCL* (*di*screte *P*oint *C*onstraint *L*anguage) allows us to make statements about points in discrete time. diPCL is a first-order language with constants from the set of integers $\mathbb{Z}$, a 2-ary 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 (diPCL-constraint) is a formula of the form $t \sim c$ or $c \sim t$ where $\sim$ is either $<$ or $=$, t is a term, and c is a constant. For example, the following are diPCL-constraints:

$$x_1 - x_2 < 2,\ x_1 = 5,\ x_1 < 6$$

The intended structure for diPCL, denoted by $\mathbf{M}_{diPCL}$, has the set of integer constants as its domain. $\mathbf{M}_{diPCL}$ interprets each constant symbol by the corresponding integer number in $\mathbb{N}$, 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.

A.3 The Language dePCL

The language *dePCL* (*de*nse *P*oint *C*onstraint *L*anguage) allows us to make statements about points in dense time.

dePCL is a first-order language with constants from the set of rational numbers $\mathbb{Q}$, a 2-ary function symbol $-$, and a binary predicate symbol $<$. The terms and atomic formulae of dePCL 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 dePCL (dePCL-constraint) is a formula of the form $t \sim c$ or $c \sim t$ where $\sim$ is either $<$ or $=$, t is a term, and c is a constant. For example, the following are dePCL-constraints:

$$x_1 - x_2 < 1/2,\ x_1 = 5/2,\ x_1 < 6/1$$

The intended structure for dePCL, denoted by $\mathbf{M}_{dePCL}$, has the set of rational constants as its domain. $\mathbf{M}_{dePCL}$ interprets each constant symbol by the corresponding rational number in $\mathbb{Q}$, function symbol $-$ by the subtraction operation over the rationals, and predicate sybmol $<$ by the relation "less than". Then, theory $Th(\mathbf{M}_{dePCL})$ is a sub-theory of $Th(\mathbb{R}, +, <)$, the theory of real numbers with addition and order.

A.4 The Language TCL

The language *TCL* (*T*opological *C*onstraint *L*anguage) allows us to represent topological properties of non-empty, regular closed subsets of $\mathbb{Q}^2$ (we will call these subsets *regions* for brevity). TCL is a first-order language with the following 6 binary predicate symbols: DC, EC, PO, EQ, TPP, and NTPP. An *atomic formula* of TCL is a formula of the form $r_1\ R\ r_2$ where r_1, r_2 are variables and R is one of the above predicates. We will often use the terminology *$\mathcal{L}$-constraints* to refer to atomic formulae of an arbitrary constraint language $\mathcal{L}$. For example, the following are TCL-constraints:

$$\mathrm{r}_1 \text{ NTPP } \mathrm{r}_2,\quad \mathrm{r}_2 \text{ PO } \mathrm{r}_3,\quad \mathrm{r}_2 \text{ EQ } \mathrm{r}_3$$

The intended structure for TCL, denoted by $\mathbf{M}_{TCL}$, has the set of regions as its domain, and interprets each of the predicate symbols given above by the corresponding topological relation of RCC-8 [28]. Note that relations NTPPi and TPPi of RCC-8 are not included in the vocabulary of TCL since they can be expressed by interchanging the arguments of NTPP and TPP.

The 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 language PCL considered below deals with polygonal shapes.

A.5 The Language RCL

The 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 2 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 2-ary 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 $UR_y(r)$ is a term 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 $t \sim c$ where $\sim$ is $<$ or $=$, t is a term of sort $\mathcal{Q}$, and c a rational constant. Symbol $=$ is the equality predicate for sort $\mathcal{Q}$; we will not use the equality predicate for sort $\mathcal{R}$ in our formulae.

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 lower-right vertex respectively. Predicate $<$ is interpreted by the relation "less than" over $\mathbb{Q}$.

A *RCL-constraint* is a RCL formula of the form $t \sim c$ where $\sim$ is $=$, $<$, $>$, $\leq$ or $\geq$, t is a term of sort $\mathcal{Q}$, and c a rational constant (the predicates $<$, $\leq$, and $\geq$ are defined as usual). For example, the following are RCL-constraints:

$$LL_x(r_2) - LL_x(r_1) < 0,\ UR_y(r_1) - LL_y(r_2) = 5/2$$

B. WELL-DESIGNED GRAPH PATTERNS

In this section we present relevant material and known results for the fragment of SPARQL corresponding to the notion of well-designed graph patterns. These come from [26, 4].

The next definition introduces the notion of well-designed graph patterns.

DEFINITION B.1 (WELL-DESIGNED PATTERNS [4]). *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.*

In [26, 4], the authors identified in well-designed graph patterns unique and interesting properties that make query evaluation more efficient in contrast to what you get without the syntactic restrictions imposed on the graph patterns by Definition B.1 above. One of these properties is that the fragment of SPARQL graph patterns corresponding to well-designed graph patterns is *weakly monotone*. In the following we introduce the notion of *weak monotonicity*, but first we define the notion of *subsumption* for mappings which is needed for *weak monotonicity*.

DEFINITION B.2 (SUBSUMPTION OF MAPPINGS). *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 set 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 mapping $\mu_2 \in \Omega_2$ such that $\mu_1 \preceq \mu_2$.*

EXAMPLE B.3. *Let us consider Example 5.2 again. Mapping μ_4 is subsumed by mapping μ_1, i.e., $\mu_4 \preceq \mu_1$.*

Informally, when a mapping μ subsumes a mapping μ', then μ contains additional information to μ', i.e., it maps additional variables to RDF terms.

Definition B.4 (Weak Monotonicity). *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 $[\![P]\!]_G \sqsubseteq [\![P]\!]_H$.*

From [4] we know that every well-designed graph pattern is weakly monotone.

Theorem B.5 (Theorem 4.3 of [4]). *Every well-designed graph pattern is weakly monotone.*

C. PROOFS FOR SECTION 6

C.1 Proof of Proposition 6.5

Proof for part *a)*
The monotonicity property for $\mathcal{Q}^S_{AUF}$ follows easily from the monotonicity property of graph patterns containing only the AND, UNION, and FILTER operators as presented in [4, Lemma 3.2].

Proof for part *b)*
From the same paper, it trivially follows that $\mathcal{Q}^S_{OPT}$ and $\mathcal{Q}^C_{OPT}$ are not monotone.

Proof for part *c)*
Now 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 in [25] we have

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

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

From the monotonicity property of AUF graph patterns [4] 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). Therefore, if the sets in formulae (1) and (2) are the same, then we shall get the required relation for monotonicity, that is, $[\![q]\!]_G \subseteq [\![q]\!]_H$.

Notice, however, that this is not the case because of the renaming functions. According to Definition 4.5 of [25], a renaming function not only depends on the mapping that has been constructed from the evaluation of a graph pattern, but also on the underlying RDF graph over which the graph pattern is evaluated. Thus, a renaming function besides renaming a specific blank node to another one per each mapping solution, that renaming has to correspond to a fresh blank node not appearing in the underlying RDF graph. 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.

Hence, in order to have the monotonicity property for $\mathcal{Q}^C_{AUF}$, we have to restrict ourselves in CONSTRUCT queries without blank nodes in their template. In such a case, the renaming functions do not have any effect on the templates of CONSTRUCT queries. Hence, the sets in formulae (1) and (2) are the same for same mappings, and thus,

$$[\![q]\!]_G \subseteq [\![q]\!]_H.$$

Proof for part *d)*
Consider a query $q = (E, P) \in \mathcal{Q}^{C'}_{WD}$ 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 in [25] we have

$$[\![q]\!]_G = \bigcup_{\mu \in [\![P]\!]_G} \{\mu(f_\mu(E)) \cap ((I \cup B) \times I \times T)\} \qquad (3)$$

$$[\![q]\!]_H = \bigcup_{\mu' \in [\![P]\!]_H} \{\mu'(f_{\mu'}(E)) \cap ((I \cup B) \times I \times T)\} \qquad (4)$$

Since the template E does not contain blank nodes, we can omit the renaming functions from these expressions and get

$$[\![q]\!]_G = \bigcup_{\mu \in [\![P]\!]_G} \{\mu(E) \cap ((I \cup B) \times I \times T)\} \qquad (5)$$

$$[\![q]\!]_H = \bigcup_{\mu' \in [\![P]\!]_H} \{\mu'(E) \cap ((I \cup B) \times I \times T)\} \qquad (6)$$

Since P is well-designed, it follows from Theorem B.5 that 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, 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, $[\![q]\!]_G \subseteq [\![q]\!]_H$.

C.2 Proof of Proposition 6.8

The proof of Proposition 6.8 is straightforward from the monotonicity property. Since $\mathcal{G} \approx \mathcal{H}$ we have the following:

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

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

$$[\![q]\!]_{\mathcal{G}} \approx [\![q]\!]_{\mathcal{H}}.$$

C.3 Proof of Lemma 6.9

The proof is similar to the one given in [12, Lemma 4.2]. We have to prove that $\bigcap [\![q]\!]_{\mathcal{G}} = \bigcap [\![q]\!]_{\mathcal{H}}$ for every $q \in \mathcal{Q}^{C'}_{AUF}$. Let $\mathcal{G} \approx \mathcal{H}$. Then, from Proposition 6.8 and because of the monotonicity property of $\mathcal{Q}^{C'}_{AUF}$, we have that $[\![q]\!]_{\mathcal{G}} \approx [\![q]\!]_{\mathcal{H}}$ for every $q \in \mathcal{Q}^{C'}_{AUF}$. Thus, for every $[\![q]\!]_G \in [\![q]\!]_{\mathcal{G}}$ there exists an $[\![q]\!]_{H_G} \in [\![q]\!]_{\mathcal{H}}$ 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 $\bigcap_{G \in \mathcal{G}} [\![q]\!]_G$ 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

$$[\![q]\!]_{H_{G_i}} \subseteq [\![q]\!]_{G_i}.$$

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, we have that $\bigcap \mathcal{H}_\mathcal{G} \supseteq \bigcap \mathcal{H}$. Similarly if q is a monotone query, we have $[\![q]\!]_{\mathcal{H}_\mathcal{G}} \subseteq [\![q]\!]_{\mathcal{H}}$ 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 get

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

C.4 Proof of Proposition 6.13

Let $[\![q]\!]_D$ be the pair $D_1 = (G_1, \phi)$ and G' an RDF graph such that $G' \in \mathit{Rep}(D_1)$. From the definition of *Rep*, there exists a valuation v' such that $\mathbf{M}_\mathcal{L} \models v'(\phi)$ and $G' \supseteq v'(G_1)$. From the assumption that $v([\![q]\!]_D) = [\![q]\!]_{v(D)}$ and since $\mathbf{M}_\mathcal{L} \models v'(\phi)$ we get

$$G' \supseteq v'(G_1) = v'(D_1) = v'([\![q]\!]_D) = [\![q]\!]_{v'(D)} = H$$

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

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

it turns out that $H \in [\![q]\!]_{\mathit{Rep}(D)}$. To see this, notice that

$$H = [\![q]\!]_{v'(D)}$$

and that

$$v'(D) \in \mathit{Rep}(D).$$

This proves that for each $G' \in \mathit{Rep}([\![q]\!]_D)$ there exists an $H \in [\![q]\!]_{\mathit{Rep}(D)}$ such that $H \subseteq G'$. To prove that $\mathit{Rep}([\![q]\!]_D) \approx [\![q]\!]_{\mathit{Rep}(D)}$ we need to show the same for the other direction.

Let H' be an RDF graph such that $H' \in [\![q]\!]_{\mathit{Rep}(D)}$. Then $H' = [\![q]\!]_H$ for some $H \in \mathit{Rep}(D)$. From 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 $H \supseteq v'(D)$. From our assumption that $v([\![q]\!]_D) = [\![q]\!]_{v(D)}$ and $\mathbf{M}_\mathcal{L} \models v'(\phi)$, we have $[\![q]\!]_{v'(D)} = v'([\![q]\!]_D)$. Since q belongs to a monotone fragment of SPARQL and $H \supseteq v'(D)$, we have

$$[\![q]\!]_H \supseteq [\![q]\!]_{v'(D)}$$

which is equivalent to

$$H' \supseteq v'([\![q]\!]_D).$$

Now observe that since $\mathbf{M}_\mathcal{L} \models v'(\phi)$, $v'([\![q]\!]_D)$ is an RDF graph G' and that $G' \in \mathit{Rep}([\![q]\!]_D)$. Therefore, we showed that for every $H' \in [\![q]\!]_{\mathit{Rep}(D)}$ there exists an $G' \in \mathit{Rep}([\![q]\!]_D)$ such that $G' \subseteq H'$.

Hence

$$\mathit{Rep}([\![q]\!]_D) \approx [\![q]\!]_{\mathit{Rep}(D)}.$$

C.5 Proof of Theorem 6.14

The proof for item *a*) can be found in the proof for Theorem C.1, while the proof for item *b*) can be found in the proof for Theorem C.2 below.

THEOREM C.1. *The triple $\langle \mathcal{D}, \mathit{Rep}, \mathcal{Q}^{C'}_{AUF}\rangle$ is a representation system.*

PROOF. To prove Theorem C.1, it is sufficient to show that for any $D = (G, \phi) \in \mathcal{D}$ and any query $q = (E, P) \in \mathcal{Q}^{C'}_{AUF}$ it is possible to define $[\![q]\!]_D$ in such a way that

$$\mathit{Rep}([\![q]\!]_D) \equiv_{\mathcal{Q}^{C'}_{AUF}} [\![q]\!]_{\mathit{Rep}(D)}.$$

By Lemma 6.9 it is sufficient to prove that

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

From Proposition 6.13 it now suffices to prove that for any valuation v such that $\mathbf{M}_\mathcal{L} \models v(\phi)$ it is

$$v([\![q]\!]_D) = [\![q]\!]_{v(D)}.$$

From Proposition E.2, the above holds if for any valuation v such that $\mathbf{M}_\mathcal{L} \models v(\phi)$, the following holds

$$v([\![P]\!]_D) = [\![P]\!]_{v(D)}.$$

This is done by 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 the two cases of Definition 5.15 (1):
  (i) In this case $o \in C$. In this case $\mathit{dom}(\nu')$ 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

  $$(\mu'(P), \theta') \in G.$$

  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 $\theta \wedge (\_l\ EQ\ o)$. Since $\mathbf{M}_\mathcal{L} \models v(\theta')$, we have $\mathbf{M}_\mathcal{L} \models v(\theta)$ and $\mathbf{M}_\mathcal{L} \models v(\_l\ EQ\ o)$. From the second set of the first item of Definition 5.15 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 $\mathbf{M}_\mathcal{L} \models v(\_l\ EQ\ o)$ 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 $o \in I \cup B \cup L \cup V$. Therefore $\nu'(o) \in I \cup B \cup L \cup U \cup C$ and

$$(\mu'(P), \theta') \in G.$$

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 the two cases of Definition 5.15 (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 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 5.15 (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 $\mu' = (\mu, \theta \wedge \_l\ EQ\ o)$ is an element of the second set, i.e., $\mu' \in [\![P]\!]_D$. Since $v(\_l) = v(x) = o$, we have $\mathbf{M}_\mathcal{L} \models v(\_l\ EQ\ o)$. Because also $\mathbf{M}_\mathcal{L} \models v(\theta)$, it holds that $\mathbf{M}_\mathcal{L} \models (\theta \wedge \_l\ EQ\ o)$, 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 $o \in I \cup B \cup L \cup V$. We have two cases to consider.
If $o \in I \cup B \cup L \cup V_n$, then $dom(\mu)$ does not contain any special query variable. Since $\mu(P) \in v(D)$, there exists conditional triple $(\mu(P), \theta) \in G$ such that $\mathbf{M}_\mathcal{L} \models v(\theta)$. By the "else" part of Definition 5.15 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 relation and get

$$v(\mu') \in v([\![P]\!]_D)$$

which 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

$$(\mu'(P), \theta) \in G$$

or equivalently

$$\mu' \in [\![P]\!]_D.$$

Applying v to the last relation 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 $v([\![P_1\ AND\ P_2]\!]_D) = [\![P_1\ AND\ P_2]\!]_{v(D)}$.

    Let $\mu \in v([\![P_1\ AND\ P_2]\!]_D)$. Therefore there exists a conditional mapping $\mu' = (\nu', \theta') \in [\![P_1\ AND\ P_2]\!]_D$ such that $\mu = v(\mu')$ and $\mathbf{M}_\mathcal{L} \models v(\theta')$. Because $[\![P_1\ AND\ P_2]\!]_D = [\![P_1]\!]_D \bowtie [\![P_2]\!]_D$, 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 E.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

    $$\mu \in [\![P_1\ AND\ P_2]\!]_{v(D)}.$$

    This proof establishes that

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

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

    Let μ be a mapping such that $\mu \in [\![P_1\ AND\ P_2]\!]_{v(D)}$. 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 E.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

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

  - 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

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

    A mapping μ is in $[\![P_1\ UNION\ P_2]\!]_{v(D)}$ 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

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

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

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

    Without loss of generality, we give the proof only for the case of filters that are atomic $\mathcal{L}$-constraints (Definition 5.18). Let μ be in $[\![P_1\ FILTER\ R]\!]_{v(D)}$. 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

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

    We can apply valuation v to

    $$\mu_1 \in [\![P_1\ FILTER\ R]\!]_D$$

    and get

    $$v(\mu_1) \in v([\![P_1\ FILTER\ R]\!]_D)$$

which is equivalent to $\mu \in v([\![P_1 \ FILTER \ R]\!]_D)$.

This proof establishes that

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

The other direction of the proof is similar and goes as follows.
Let μ be a mapping in $v([\![P_1 \ FILTER \ R]\!]_D)$. Then there exists a conditional mapping $\mu_1 = (\nu_1, \theta_1) \in [\![P_1 \ FILTER \ R]\!]_D$ 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 $\mathbf{M}_{\mathcal{L}} \models v(\theta_1)$, then 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

$$\mu \in [\![P_1 \ FILTER \ R]\!]_{v(D)}.$$

□

THEOREM C.2. *The triple $\langle \mathcal{D}, Rep, \mathcal{Q}^{C'}_{WD} \rangle$ is a representation system.*

PROOF. The proof for Theorem C.2 is the same with the proof for Theorem C.1 and differs only in the inductive step for the OPTIONAL operator. Thus, in this case,

$$P \text{ is } P_1 \ OPT \ P_2.$$

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 $v([\![P_1 \ OPT \ P_2]\!]_D) = [\![P_1 \ OPT \ P_2]\!]_{v(D)}$ or equivalently

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

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

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

For the first case, i.e., $\mu' \in ([\![P_1]\!]_D \bowtie [\![P_2]\!]_D)$ the proof is the same as in the proof of Theorem C.1, hence we finally get that

$$\mu \in ([\![P_1]\!]_{v(D)} \bowtie [\![P_2]\!]_{v(D)})$$

and thus from Formula (1) we have

$$\mu \in [\![P_1 \ OPT \ P_2]\!]_{v(D)}.$$

For the second 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 cases:

1. $\mu' \in [\![P_1]\!]_D$ and for all $\mu_2 \in [\![P_2]\!]_D$, μ' and μ_2 are not 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' — and hence not 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 (1) we have

$$\mu \in [\![P_1 \ OPT \ P_2]\!]_{v(D)}.$$

2. μ' is the conditional mapping (ν', θ') and there exists 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 mappings $\mu_i = (\nu_i, \theta_i) \in [\![P_2]\!]_D$, μ'' and μ_i are possibly compatible and

   $$\theta' \text{ is } \theta \wedge \left(\theta_i \supset \bigvee_x \neg(\mu''(x) \text{ EQ } \mu_i(x)) \right)$$

   for $x \in dom(\mu'') \cap dom(\mu_i) \cap V_s$.

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 possible 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, 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 (1) we get

$$\mu \in [\![P_1 \ OPT \ P_2]\!]_{v(D)}.$$

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

D. PROOFS FOR SECTION 7

D.1 Proof of Lemma 7.3

To show that $\bigcap Rep(D) = \bigcap Rep((D^{\mathrm{EQ}})^*)$ we will first prove that

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

Let t be an RDF triple such that $t \notin \bigcap Rep((D^{\mathrm{EQ}})^*)$. Then, by definition of *Rep* we get

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

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

- either there is no conditional triple $(t', \theta') \in (G^{\mathrm{EQ}})^*$ such that $\mathbf{M}_{\mathcal{L}} \models v(\theta')$, that is, $\mathbf{M}_{\mathcal{L}} \not\models v(\theta')$,
- 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 will be 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 the first case above, and 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 the second case above.

Consider now the second case above and a triple $(t', \theta') \in (G^{\mathrm{EQ}})^*$. Since $\mathbf{M}_{\mathcal{L}} \models v(\theta')$ and $(t', \theta'_i) \in G^{\mathrm{EQ}}$, 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 $(t', \theta'_i) \in G^{\mathrm{EQ}}$, then there exists 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'' would be the same with t', or it would have in its object position an e-literal $\_l$ such that $\phi \models \_l$ EQ o.
2. If $o \notin C$, then t' and t'' would be 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 the second item and since $v(t') \neq t$ (see second case above), we have that $v(t'') \neq t$ as well. As for the first item, 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 $\phi \models \_l$ 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 (this is because we are considering the second case for which $v(t') \neq t$). Since also $\phi \models \_l$ EQ o, then every valuation v' that makes $v'(\phi)$ *true* it should make $v'(\_l$ 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 $v(G)$ cannot contain triple t or equivalently that $t \notin v(G)$. Hence, from the definition of *Rep* we have

$$t \notin \bigcap Rep(D)$$

which proves that

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

The other direction of the proof for showing

$$\bigcap Rep((D^{\mathrm{EQ}})^*) \subseteq \bigcap Rep(D)$$

is similar.

D.2 Proof of Theorem 7.4

Notice that the certain answer for q over D is the set

$$\bigcap [\![q]\!]_{Rep(D)}.$$

From the Representation Theorem and since $q \in \mathcal{Q}^{C'}_{AUF}$ 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 6.3 for the identity query.

Having Lemma 7.3 it now suffices to prove that the given algorithm computes the set

$$\bigcap Rep((([\![q]\!]_D)^{\mathrm{EQ}})^*)$$

or, using the notation of Theorem 7.4, set

$$\bigcap Rep(((D_q)^{\mathrm{EQ}})^*).$$

The first step of the algorithm evaluates q over D, that is, it computes $D_q = (G_q, \phi)$, while the second step 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 three 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

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

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 $\mathbf{M}_{\mathcal{L}} \models v(\phi)$.

The first condition, that is, requiring that $\mathbf{M}_{\mathcal{L}} \models v(\theta')$ holds for all valuations v such that $\mathbf{M}_{\mathcal{L}} \models v(\phi)$, is equivalent to requiring that $\phi \models \theta'$ holds, a requirement that step three imposes.

As for the second condition, 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 three 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 three computes exactly the set

$$\bigcap Rep(((D_q)^{\text{EQ}})^*).$$

D.3 Proof of Theorem 7.6

Working similar to Proof D.2, it suffices to show that an RDF triple t is in the certain answer of $q \in \mathcal{Q}^{C'}_{AUF}$ over D, that is, $t \in \bigcap Rep(\llbracket q \rrbracket_D)$, if and only if the following formula is valid:

$$(\forall \_\mathbf{l})(\phi(\_\mathbf{l}) \supset \Theta(t, q, D, \_\mathbf{l})) \qquad (1)$$

Let us now construct formula $\Theta(t, q, D, \_\mathbf{l})$ given the evaluation of q over D, i.e., $\llbracket q \rrbracket_D = (G', \phi)$. Recall that formula $\Theta(t, q, D, \_\mathbf{l})$ is a disjunction of constraints θ_i for each conditional triple $(t'_i, \theta'_i) \in G'$ such that if t and t'_i have the same subject and predicate, θ_i is

- θ'_i if they agree in the object position as well,
- $\theta'_i \wedge (\_l \text{ EQ } o)$ if t has the constant $o \in C$ at the object position and t'_i has the e-literal $\_l \in U$ at the object position.

In every other case (i.e., t and t'_i do not agree in the subject and predicate or agree but the object of t is not a constant from C or the object of t'_i is not an e-literal from U) no constraint θ_i is generated for those conditional triples and yet $\Theta(t, q, D, \_\mathbf{l})$ is taken to be $false$. Therefore, formula (1) is either unsatisfiable (we assume that the global constraint is always satisfiable) or of the form

$$(\forall \_\mathbf{l})(\phi(\_\mathbf{l}) \supset \theta_1 \vee \ldots \vee \theta_k) \qquad (2)$$

Consider now an RDF triple $t \notin \bigcap Rep(\llbracket q \rrbracket_D)$. 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 $t \neq v(t')$.

Considering the first case above and since $\mathbf{M}_\mathcal{L} \not\models v(\theta')$, formula (2) 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 $\theta_i \vee \ldots \vee \theta_k$ in formula (2) is always $false$, and hence the whole formula is unsatisfiable.

To prove our result (i.e., that formula (1) is unsatisfiable for the specific RDF triple we considered), we have to show that formula (2) is unsatisfiable as well for the case in which $\mathbf{M}_\mathcal{L} \models v(\theta')$ and $t \neq v(t')$ (the second case above). 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 to that constant.

From the first case, no constraint θ_i is included in formula (2). 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 (2) is unsatisfiable as well.

The other direction of the proof is similar.

D.4 Proofs for Section 7.1

Proposition D.1. *Let $D = (G, \phi)$ be an* RDF$^{\text{i}}$ *database, q a query from $\mathcal{Q}^{C'}_{AUF}$ and H an RDF graph. The certainty problem, $CERT_C(q, H, D)$, when the language of $\mathcal{L}$-constraints is* ECL *is* coNP*-complete.*

Proof. To decide $CERT_C(q, H, D)$, we have to check that $H \subseteq \bigcap \llbracket q \rrbracket_{Rep(D)}$ which from Definition 4.4 is equivalent to checking that $H \subseteq \llbracket q \rrbracket_{v(D)}$ 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 $H \not\subseteq \llbracket q \rrbracket_{v(D)}$. 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 $t \notin \llbracket q \rrbracket_{v(D)}$. 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 $t \notin \llbracket q \rrbracket_{v(D)}$. The steps for accomplishing the latter check, using Definition 4.6 for evaluating CONSTRUCT query forms of standard SPARQL [25], are the following:

1. Choose the next triple $t \in H$.
2. Loop over all candidate mappings μ for set $\llbracket P \rrbracket_{v(D)}$ generating a mapping per iteration, where P is the graph patter of query q.
3. Check that $\mu \in \llbracket P \rrbracket_{v(D)}$.
4. Construct the renaming function f_μ based on the mapping μ.
5. Generate set $S_\mu = \{\mu(f_\mu(E)) \cap (I \cup B) \times I \times T\}$.
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 $\llbracket P \rrbracket_{v(D)}$ 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 in [26]. 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 in [6, Theorem 5.11, p. 118]. $\square$

Proposition D.2. *Let $D = (G, \phi)$ be an* RDF$^{\text{i}}$ *database, q a query from $\mathcal{Q}^{C'}_{AUF}$ and H an RDF graph. The certainty problem, $CERT_C(q, H, D)$, when the language of $\mathcal{L}$-constraints is one of* dePCL, diPCL, *or* RCL *is* coNP-*complete.*

Proof. We sketch the proof for dePCL. The proof is similar for the cases of diPCL and RCL.

Similar to the proof for ECL, to decide $CERT_C(q, H, D)$, we have to check that $H \subseteq \bigcap [\![q]\!]_{Rep(D)}$ which from Definition 4.4 is equivalent to checking that $H \subseteq [\![q]\!]_{v(D)}$ for all valuations v such that $\mathbf{M}_{dePCL} \models v(\phi)$. The complement of this problem is to check whether there exists a valuation v such that $\mathbf{M}_{dePCL} \models v(\phi)$ 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 $\mathbf{M}_{dePCL} \models v(\phi)$ 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. First, 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 [16], and Theorem 7.6 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 [16] 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. The coNP-hardness of $CERT_C(q, H, D)$ follows from Proposition 3.1 of [35] where a sublanguage 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 diPCL and RCL as well. $\square$

Proposition D.3. *Let $D = (G, \phi)$ be an* RDF$^{\text{i}}$ *database, q a query from $\mathcal{Q}^{C'}_{AUF}$ and H an RDF graph. The certainty problem, $CERT_C(q, H, D)$, when the language of $\mathcal{L}$-constraints is* TCL *is in EXPTIME.*

Proof. Since the satisfiability problem for conjunctions of TCL-constraints is known to be in PTIME [29], we can transform Formula (2) in DNF, construct a constraint network for each disjunct, and check them for consistency. This can be trivially solved in EXPTIME (DNF transformation). $\square$

Proposition D.4. *Let $D = (G, \phi)$ be an* RDF$^{\text{i}}$ *database, q a query from $\mathcal{Q}^{C'}_{AUF}$ and H an RDF graph. The certainty problem, $CERT_C(q, H, D)$, when the language of $\mathcal{L}$-constraints is* PCL *with the predicates of the RCC-5 calculus is in EXPTIME.*

Proof. The above procedure applies also to the case of PCL restricted to the topological relations of RCC-5 when the involved constants are polygons in V-representation. In [20] it is shown that the satisfiability problem for such constraints can be decided in PTIME. $\square$

E. ADDITIONAL PROPOSITIONS

The next 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 E.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.

Proposition E.2. *Let $D = (G, \phi)$ be an* RDF$^{\text{i}}$ *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, $v([\![P]\!]_D) = [\![P]\!]_{v(D)}$ implies $v([\![q]\!]_D) = [\![q]\!]_{v(D)}$.*

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

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

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

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

Likewise, let $[\![q]\!]_{v(D)}$ be the RDF graph H. According to the definition of the evaluation of CONSTRUCT queries on RDF graphs [25], H is the following set:

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

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

$$t \in (\mu(f_\mu(E)) \cap ((I \cup B) \times I \times T)) \quad (3)$$

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 $\mu = v(\mu')$. Since $\mu = v(\mu')$ relation (3) is written as

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

which is equivalent to the following

$$t \in v\left(\mu'(f_\mu(E)) \cap ((I \cup B) \times I \times T)\right). \quad (4)$$

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

$$v\left(\mu'(f_\mu(E)) \cap ((I \cup B) \times I \times T)\right) \subseteq v(D').$$

Because also of relation (4) we get

$$t \in v(D').$$

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 conditional mapping $\mu = (\nu, \theta) \in [\![P]\!]_D$ and conditional triple $t_c = (t', \theta) \in G'$ such that $\mathbf{M}_{\mathcal{L}} \models v(\theta)$, $v(t_c) = v(t') = t$. From (1) we then have

$$t' \in (\mu(f_\mu(E)) \cap ((I \cup B) \times I \times T)). \tag{5}$$

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 (5) we get

$$v(t') \in v\left(\mu(f_\mu(E)) \cap ((I \cup B) \times I \times T)\right)$$

which is equivalent to

$$t \in (v(\mu(f_\mu(E))) \cap ((I \cup B) \times I \times T)).$$

Since $\mu' = v(\mu)$, the above relation becomes

$$t \in \left(\mu'(f_\mu(E)) \cap ((I \cup B) \times I \times T)\right).$$

From the above and because of (2) and $\mu' \in [\![P]\!]_{v(D)}$ we have

$$t \in \left(\mu'(f_\mu(E)) \cap ((I \cup B) \times I \times T)\right) \subseteq H$$

and hence

$$t \in H.$$

$\square$

Bibliography

[All83] James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, November 1983.

[BCF+11] Carl Friedrich Bolz, Antonio Cuni, Maciej Fijalkowski, Michael Leuschel, Samuele Pedroni, and Armin Rigo. Runtime feedback in a meta-tracing JIT for efficient dynamic languages. In *ICOOOLPS*, 2011.

[BCFR09] Carl Friedrich Bolz, Antonio Cuni, Maciej Fijalkowski, and Armin Rigo. Tracing the meta-level: PyPy's tracing JIT compiler. In *ICOOOLPS*, 2009.

[BP10] Sotiris Batsakis and Euripides G. M. Petrakis. SOWL: spatio-temporal representation, reasoning and querying over the semantic web. In *Proceedings of the 6th International Conference on Semantic Systems*, I-SEMANTICS '10, New York, NY, USA, 2010. ACM.

[BSS96] Michael H. Boelen, Richard T. Snodgrass, and Michael D. Soo. Coalescing in temporal databases. *IEEE Computer*, 19:35–42, 1996.

[CDI+97] James Clifford, Curtis Dyreson, Tomas Isakowitz, Christian S. Jensen, and Richard Thomas Snodgrass. On the semantics of now in databases. *ACM TODS*, 22(2):171–214, June 1997.

[Cla81] Bowman L Clarke. A calculus of individuals based on "connection". *NDJFL*, 22:204–218, 1981.

[Coh97] Anthony G. Cohn. Qualitative Spatial Representation and Reasoning Techniques. In *KI*, 1997.

[DDK+12] Corneliu Octavian Dumitru, Mihai Datcu, Manolis Koubarakis, Michael Sioutis, Babis Nikolaou, and Consortium members. Ontologies for the VO for TerraSAR-X data. Deliverable D6.2.1, TELEIOS project, 2012.

[DDL02] Chris J. Date, Hugh Darwen, and Nikos A. Lorentzos. *Temporal data and the relational model.* Elsevier, 2002.

[DGA01] James P. Delgrande, Arvind Gupta, and Tim Van Allen. A comparison of point-based approaches to qualitative temporal reasoning. *Artif. Intell.*, 131(1-2):135–170, 2001.

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.

[Fre78] Eugene C. Freuder. Synthesizing constraint expressions. *CACM*, 21:958–966, 1978.

[GDH08] John Goodwin, Catherine Dolbear, and Glen Hart. Geographical Linked Data: The Administrative Geography of Great Britain on the Semantic Web. *Transactions in GIS*, 12:19–30, 2008.

[GHV05] Claudio Gutierrez, Carlos Hurtado, and Ro Vaisman. Temporal RDF. In *European Conference on the Semantic Web (ECSWï£¡05)*, pages 93–107, 2005.

[GHV07] Claudio Gutierrez, Carlos Hurtado, and Alejandro Vaisman. Introducing Time into RDF. *IEEE Transactions on Knowledge and Data Engineering*, 19(2):207–218, 2007.

[GLG+12] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12)*, Hollywood, October 2012.

[Got94] Nicholas Mark Gotts. How Far Can We 'C'? Defining a 'Doughnut' Using Connection Alone. In *KR*, 1994.

[Got96] Nicholas Mark Gotts. Topology From A Single Primitive Relation: Defining Topological Properties and Relations In Terms Of Connection. Technical report, School of Computer Studies, University of Leeds, 1996.

[Gra10] Fabio Grandi. T-sparql: a tsql2-like temporal query language for rdf. In *In International Workshop on on Querying Graph Structured Data*, pages 21–30, 2010.

[GS95] Alfonso Gerevini and Lenhart K. Schubert. Efficient algorithms for qualitative reasoning about time. *Artif. Intell.*, 74(2):207–248, 1995.

[GWW08] Zeno Gantner, Matthias Westphal, and Stefan Woelfl. GQR - A Fast Reasoner for Binary Qualitative Constraint Calculi. In *AAAI Workshop on Spatial and Temporal Reasoning*, 2008.

[HV06] Carlos A. Hurtado and Alejandro A. Vaisman. Reasoning with Temporal Constraints in RDF. In *Principles and Practice of Semantic Web Reasoning*, pages 164–178. Springer, 2006.

[KK10a] Manolis Koubarakis and Kostis Kyzirakos. Modeling and Querying Metadata in the Semantic Sensor Web: The Model stRDF and the Query Language stSPARQL. In *ESWC*, pages 425–439, 2010.

[KK10b] Manolis Koubarakis and Kostis Kyzirakos. Modeling and Querying Metadata in the Semantic Sensor Web: The Model stRDF and the Query Language stSPARQL. In *ESWC*, 2010.

[KKK+12a] Manolis Koubarakis, Manos Karpathiotakis, Kostis Kyzirakos, Charalampos Nikolaou, and Michael Sioutis. Data Models and Query Languages for Linked Geospatial Data. Invited papers from 8th Reasoning Web Summer School, 2012.

[KKK12b] Kostis Kyzirakos, Manos Karpathiotakis, and Manolis Koubarakis. Strabon: A Semantic Geospatial DBMS. In *11th International Semantic Web Conference*, 2012.

[KNF11] Manolis Koubarakis, Charalampos Nikolaou, and Vissarion Fisikopoulos. Theoretical results on query processing for RDF/SPARQL with time and space. Deliverable D2.3, European FP7 project TELEIOS, 2011. Available from: `http://www.earthobservatory.eu/deliverables/FP7-257662-TELEIOS-D2.3.pdf`.

[KPM+12] Charalabos (Haris) Kontoes, Ioannis Papoutsis, Dimitrios Michail, Themos Herekakis, and Consortium members. An implementation of the real-time fire monitoring application - Phase I. Deliverable D7.2, TELEIOS project, 2012.

[KPMm10] Charalabos (Haris) Kontoes, Ioannis Papoutsis, Dimitrios Michail, and Consortium members. Requirements specification for the real-time fire monitoring application. Deliverable D7.1, TELEIOS project, 2010.

[LM94] Peter B. Ladkin and Roger D. Maddux. On binary constraint problems. *JACM*, 41:435–469, 1994.

[LPSZ10] Nuno Lopes, Axel Polleres, Umberto Straccia, and Antoine Zimmermann. AnQL: SPARQLing Up Annotated RDFS. In *International Semantic Web Conference*, 2010.

[LR04] Gérard Ligozat and Jochen Renz. What Is a Qualitative Calculus? A General Framework. In *PRICAI*, 2004.

[Mot12] Boris Motik. Representing and querying validity time in rdf and owl: A logic-based approach. *Journal of Web Semantics*, 12, 2012.

[ogc12] Open Geospatial Consortium. OGC GeoSPARQL - A geographic query language for RDF data. OGC Candidate Implementation Standard, 02 2012.

[Ope12] Open Geospatial Consortium. OGC GeoSPARQL - A geographic query language for RDF data. OGC® Implementation Standard, 2012.

[Per08] Matthew Perry. *A Framework to Support Spatial, Temporal and Thematic Analytics over Semantic Web Data*. PhD thesis, Wright State University, 2008.

[PUS08] Andrea Pugliese, Octavian Udrea, and V. S. Subrahmanian. Scaling rdf with time. In *Proceedings of the 17th International Conference on World Wide Web*, pages 605–614, New York, NY, USA, 2008. ACM.

[RCC92a] David A. Randell, Zhan Cui, and Anthony Cohn. A Spatial Logic Based on Regions and Connection. In *KR*. 1992.

[RCC92b] 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 Knowledge Representation and Reasoning*, 1992.

[RL05] Jochen Renz and Gérard Ligozat. Weak Composition for Qualitative Spatial and Temporal Reasoning. In *CP*, 2005.

[RN99a] J. Renz and B. Nebel. On the Complexity of Qualitative Spatial Reasoning: A Maximal Tractable Fragment of the Region Connection Calculus. *Artificial Intelligence*, 1-2:95–149, 1999.

[RN99b] Jochen Renz and Bernhard Nebel. On the Complexity of Qualitative Spatial Reasoning: A Maximal Tractable Fragment of the Region Connection Calculus. *AI*, 108:69–123, 1999.

[RN01] Jochen Renz and Bernhard Nebel. Efficient methods for qualitative spatial reasoning. *Journal of Artificial Intelligence Research (JAIR)*, 15:289–318, 2001.

[RN07] Jochen Renz and Bernhard Nebel. Qualitative spatial reasoning using constraint calculi. In *Handbook of Spatial Logics*, pages 161–215. Springer, 2007.

[SDMD10] Gottfried Schwarz, Corneliu Octavian Dumitru, Katrin Molch, and Mihai Datcu. Requirements specification for the VO for TerraSAR-X data and applications. Deliverable D6.1, TELEIOS project, 2010.

[SK12] Michael Sioutis and Manolis Koubarakis. Consistency of Chordal RCC-8 Networks. In *ICTAI*, 2012. Athens, Greece, November 07–09, 2012.

[Sno95] Richard T. Snodgrass, editor. *The TSQL2 Temporal Query Language*. Kluwer, 1995.

[SS09a] Markus Stocker and Evren Sirin. PelletSpatial: A Hybrid RCC-8 and RDF/OWL Reasoning and Query Engine. In *OWLED*, 2009.

[SS09b] Markus Stocker and Evren Sirin. PelletSpatial: A hybrid RCC-8 and RDF/OWL reasoning and query engine. In *OWLED*, 2009.

[TB09] Jonas Tappolet and Abraham Bernstein. Applied temporal RDF: Efficient temporal querying of RDF data with SPARQL. In *ESWC*, pages 308–322, 2009.

[WM09] M. Wessel and R. Moller. Flexible software architectures for ontology-based information systems. *Journal of Applied Logic*, 7(1):75–99, 2009.