Theoretical results on query processing for RDF/SPARQL with time and space

Manolis Koubarakis, Charalampos Nikolaou, and Vissarion Fisikopoulos

November 9, 2011
Executive Summary

In this deliverable we extend the theoretical foundations of the data models stRDF and stRDF\textsuperscript{i} and query language stSPARQL presented in Deliverable D2.1 “A data model and query language for an extension of RDF with time and space”. We study the semantics and the computational complexity of evaluating stSPARQL queries over stRDF graphs and stRDF\textsuperscript{i} databases. The results of this deliverable are original and extend the state of the art in theory of geospatial extensions of W3C standards RDF and SPARQL.
## Document Information

<table>
<thead>
<tr>
<th>Contract Number</th>
<th>FP7-257662</th>
<th>Acronym</th>
<th>TELEIOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full title</td>
<td>TELEIOS: Virtual Observatory Infrastructure for Earth Observation Data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project URL</td>
<td><a href="http://www.earthobservatory.eu/">http://www.earthobservatory.eu/</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EU Project Officer</td>
<td>Francesco Barbato</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Number</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2.3</td>
<td></td>
<td>Theoretical results on query processing for RDF/SPARQL with time and space</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task</th>
<th>Number</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2.1</td>
<td></td>
<td>Extending RDF and SPARQL with time and space</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Work package</th>
<th>Number</th>
<th>WP2</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Date of delivery</th>
<th>Contract</th>
<th>M12 (Sept. 2011)</th>
<th>Actual</th>
<th>September 5, 2011</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th></th>
<th>Draft □</th>
<th>Final □</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Nature</th>
<th>Prototype □</th>
<th>Report □</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Distribution Type</th>
<th>Public □</th>
<th>Restricted □</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Responsible Partner</th>
<th>NKUA</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>QA Partner</th>
<th>CWI</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Contact Person</th>
<th>Manolis Koubarakis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email</td>
<td><a href="mailto:koubarak@di.uoa.gr">koubarak@di.uoa.gr</a></td>
</tr>
<tr>
<td>Phone</td>
<td>+30 210 727 5213</td>
</tr>
<tr>
<td>Fax</td>
<td>+30 210 727 5214</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>National and Kapodistrian University of Athens Department of Informatics and Telecommunications</td>
<td>NKUA</td>
<td>Prof. Manolis Koubarakis National and Kapodistrian University of Athens Department of Informatics and Telecommunications Panepistimiopolis, Illisia, GR-15784 Athens, Greece Email: (<a href="mailto:koubarak@di.uoa.gr">koubarak@di.uoa.gr</a>) Tel: +30 210 7275213, Fax: +30 210 7275214</td>
</tr>
<tr>
<td>Fraunhofer Institute for Computer Graphic Research</td>
<td>Fraunhofer</td>
<td>MSc. Thorsten Reitz Fraunhofer Institute for Computer Graphic Research Fraunhofer Strasse 5, D-64283 Darmstadt, Germany Email: (<a href="mailto:thorsten.reitz@igd.fraunhofer.de">thorsten.reitz@igd.fraunhofer.de</a>) Tel: +49 6151 155 416, Fax: +49 6151 155 444</td>
</tr>
<tr>
<td>German Aerospace Center The Remote Sensing Technology Institute Photogrammetry and Image Analysis Department Image Analysis Team</td>
<td>DLR</td>
<td>Prof. Mihai Datcu German Aerospace Center The Remote Sensing Technology Institute Oberpfaffenhofen, D-82234 Wessling, Germany Email: (<a href="mailto:mihai.datcu@dlr.de">mihai.datcu@dlr.de</a>) Tel: +49 8153 28 1388, Fax: +49 8153 28 1444</td>
</tr>
<tr>
<td>Stichting Centrum voor Wiskunde en Informatica Database Architecture Group</td>
<td>CWI</td>
<td>Prof. Martin Kersten Stichting Centrum voor Wiskunde en Informatica P.O. Box 94097, NL-1090 GB Amsterdam, Netherlands Email: (<a href="mailto:martin.kersten@cwi.nl">martin.kersten@cwi.nl</a>) Tel: +31 20 5924066, Fax: +31 20 5924199</td>
</tr>
<tr>
<td>National Observatory of Athens Institute for Space Applications and Remote Sensing</td>
<td>NOA</td>
<td>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: (<a href="mailto:kontoes@space.noa.gr">kontoes@space.noa.gr</a>) Tel: +30 210 8109186, Fax: +30 210 6138343</td>
</tr>
<tr>
<td>Advanced Computer Systems A.C.S S.p.A</td>
<td>ACS</td>
<td>Mr. Ugo Li Giammatteo Advanced Computer Systems A.C.S S.p.A Via Della Bufalotta 378, RM-00139 Rome, Italy Email: (<a href="mailto:udig@acsys.it">udig@acsys.it</a>) Tel: +39 06 8706944, Fax: +39 06 87201502</td>
</tr>
</tbody>
</table>

D2.3 Theoretical results on query processing for RDF/SPARQL with time and space
Contents

1 Introduction 1

2 Background 4
   2.1 First-Order Languages, Structures, and Theories  4
      2.1.1 The Complexity of Logical Theories for Algebra and Geometry  5
      2.1.2 Complexity Results for $\mathbb{R}$  6
      2.1.3 Complexity Results for $\mathbb{R}^+$  6
   2.2 First-Order Languages of Spatial Constraints  8
      2.2.1 The Language TCL  8
      2.2.2 The Language PCL  9
      2.2.3 The Language RCL  9
      2.2.4 The Language TRCL  10
   2.3 Complexity Results for First-Order Languages of Spatial Constraints  11
      2.3.1 Satisfiability and Validity  11
      2.3.2 Variable and Quantifier Elimination  11
      2.3.3 Results for TCL  15
      2.3.4 Results for PCL  15
      2.3.5 Results for RCL  16
      2.3.6 Results for TRCL  16
   2.4 The Constraint Database Model  17
      2.4.1 Some Useful Complexity Results for Linear Constraint Databases  18
   2.5 Polyhedral Computation  18
   2.6 Conclusions  19

3 The Model stRDF and the Query Language stSPARQL  20
   3.1 Data Model  20
      3.1.1 Linear Constraints  20
      3.1.2 The stRDF Data Model  20
      3.1.3 The stRDF Data Model  21
   3.2 The stSPARQL Query Language: Formal Definition and Semantics  22
   3.3 Conclusions  26

4 The Complexity of stSPARQL Query Evaluation  27
   4.1 Evaluation of Spatial Conditions  27
      4.1.1 Evaluation of k-ary Spatial Terms  29
      4.1.2 Evaluation of Metric Spatial Terms  31
      4.1.3 Evaluation of Topological Conditions  32
      4.1.4 Evaluation of Spatial Conditions  34
   4.2 A Decision Procedure for stSPARQL Query Evaluation  35
   4.3 Conclusions  38

5 The stRDF Data Model: Motivation, Semantics, and Complexity of Query Evaluation  39
   5.1 Motivation  39
      5.1.1 Indefinite Information in the NOA Use Case  39
   5.2 The stRDF Data Model  41
   5.3 Semantics of the stRDF Data Model  43
   5.4 Evaluating stSPARQL on stRDF Databases  44
   5.5 A Representation System for stRDF and stSPARQL  49
   5.6 Complexity Results  57
      5.6.1 Data Complexity of CERT($q, M, D$)  58
   5.7 Conclusions and Open Problems  59

D2.3 Theoretical results on query processing for RDF/SPARQL with time and space iv
6 Conclusions

D2.3 Theoretical results on query processing for RDF/SPARQL with time and space
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Three boxes</td>
<td>11</td>
</tr>
<tr>
<td>4.1</td>
<td>Syntactic structure and evaluation of spatial conditions</td>
<td>28</td>
</tr>
<tr>
<td>4.2</td>
<td>Examples of binary topological operators between regions A (dotted) and B (grey)</td>
<td>33</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Complexity results for the structures $\mathcal{R}_+$ and $\mathcal{R}$ (symbol “—” means that the case is not applicable)</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Definition of the various relations of RCC (relations in bold are included in RCC-8)</td>
<td>9</td>
</tr>
<tr>
<td>4.1</td>
<td>Semantics of topological operators</td>
<td>32</td>
</tr>
<tr>
<td>4.2</td>
<td>Complexity results. (*) denotes non overlapping polytopes</td>
<td>35</td>
</tr>
<tr>
<td>5.1</td>
<td>A conditional answer in stRDF$^i$</td>
<td>41</td>
</tr>
</tbody>
</table>
1. Introduction

In this deliverable we extend the theoretical foundations of the data models stRDF and stRDF\textsuperscript{i} and query language stSPARQL presented in Deliverable D2.1 “A data model and query language for an extension of RDF with time and space” [KKN+11]. We study the semantics and analyse the computational complexity of evaluating stSPARQL queries over stRDF graphs and stRDF\textsuperscript{i} databases.

Analysing the computational complexity of query evaluation for a given query language and data model is a typical theoretical problem in databases and knowledge bases [AHV95]. The literature defines various methods to compute this complexity. The differences in existing methods have to do with the parameters with respect to which the complexity is measured and the nature of the problem which is used to define the complexity.

When we consider the parameter with respect to which the complexity of query evaluation is measured, the three main possibilities in the literature are the following [CH80a, CH80b, Var82, AHV95]:

- **data complexity**: this is the complexity of evaluating a fixed query for variable database inputs; in this case, the database is the parameter to the problem of measuring the complexity. Thus, the complexity is given as a function of the length of the database instance.

- **expression complexity**: this is the complexity of evaluating on a fixed database instance the various queries specifiable in a given query language; in this case, the query is the parameter to the problem of measuring the complexity. Thus, the complexity is given as a function of the length of the query.

- **combined complexity**: this is the complexity of evaluating arbitrary queries over arbitrary database instances. In this case, both the query and the database instance are the parameters to the problem of measuring the complexity. Thus, the complexity is given as a function of the combined size of the query and the database instance.

Clearly, these measures provide different perspectives on the complexity of query evaluation. Data complexity, on the one hand, measures the expressive power of the query language, or alternatively, how difficult it is to answer a specific query. Expression complexity, on the other hand, measures the succinctness of the query language, or alternatively, how difficult it is to answer different queries. Last, combined complexity combines them both.

The second way of measuring the complexity of query evaluation concerns the kind of problem which is to be solved. It is standard to make a distinction between the decision problem associated with the query, and that of actually constructing the result of the query [AHV95]. For relational database queries, the decision problem is defined as the problem of checking whether a given tuple belongs to the answer of the query evaluated over a database instance. More generally, the decision problem can be defined as the problem of checking whether a given relation instance is the answer to a query.

These two problem definitions are in most cases interchangeable [AHV95]. In particular, for complexity classes insensitive to a polynomial factor (i.e., the classes that are proper supersets of PTIME), the definitions are equivalent in that they classify their respective problems under the same complexity class. However, the definition of the result construction is more general than that of the recognition one, in that the second definition can be employed in order to construct the answer required by the first.

In this deliverable we concentrate only on the combined and data complexity of various decision problem corresponding to evaluating stSPARQL queries.
The contributions of this deliverable are the following:

• We prove that stSPARQL queries can be evaluated over stRDF graphs with 2-EXPTIME combined complexity and 2-EXPTIME data complexity.

An important special case is when the input stRDF graph contains geospatial objects of fixed dimension (e.g., 2-dimensional or 3-dimensional semi-linear sets). This case corresponds to the version of stSPARQL that uses the OGC standard Well Known Text (WKT) instead of linear constraints as in the original language defined in [KK10]. This version was defined in Section 4 of Deliverable 2.1 [KKN+11] and was called SPARQL++ to distinguish it from the original stSPARQL language of [KK10]. This is the version of stSPARQL we use in TELEIOS and it has already partially implemented in the system Strabon available by NKUA. stSPARQL++ is very close syntactically and semantically to GeoSPARQL, a recent OGC proposal for geospatial extension of SPARQL.

The combined complexity of stSPARQL query evaluation over stRDF graphs in this case is in EXPTIME and the data complexity is in EXPTIME. If the query does not contain the operators for computing the volume or area of a geospatial object, then the combined complexity goes down to PSPACE and the data complexity to PTIME.

If we compare these above results with the fact that SPARQL queries can be evaluated with PSPACE combined complexity and LOGSPACE data complexity [PAG06, PAG09], we see that, in the general case, we have an increase both in combined and data complexity which is due to the presence of geospatial objects (more precisely: semi-linear point sets) of arbitrary dimension in stRDF graphs, and the availability of a rich set of spatial operators in stSPARQL. Evaluating stSPARQL queries over stRDF graphs containing these geospatial objects is much harder than standard SPARQL query evaluation over RDF graphs. However, if we fix the dimension of the geospatial objects we have in the database and leave out the most expensive operators from the query, then we obtain complexity upper bounds that are similar to the ones for SPARQL.

• We study the semantics of the model stRDF³ which was originally defined in Section 5 of Deliverable D2.1 [KKN+11]. stRDF³ extends stRDF with a new kind of literals that represent spatial regions for which we do not have exact coordinates because the known information about them is incomplete, indefinite, or qualitative in nature (e.g., region A is inside a rectangle R for which we have exact coordinates but we do not know A’s exact geographic location, or region A is north of region B and it overlaps region C, but we do not know anything else about it, etc.). In many cases, such information can be expressed in terms of disjunctive qualitative spatial constraints representing size (e.g., “large”, “small”), direction (e.g., “right of”, “above”), distance (e.g., “far”, “near”), shape (e.g., “convex”), or topology (e.g., “overlaps”, “contains”) of spatial objects [KND07]. stRDF³ is essentially an extension of stRDF in the spirit of indefinite constraint databases developed by the NKUA group in the past [Kon94c, Kon94b, Kon97].

The most important of our results here shows that the well-known concept of representation system from the seminal paper of Imielinski and Lipski on incomplete information in relational databases [IL81] can be easily defined in the case of stRDF³ as well. To be able to study representation systems in the case of stRDF³, we define the concepts of monotonicity of stSPARQL queries, coinitiality of classes of sets of RDF graphs and other interesting concepts. All of these concepts and results are original in the literature of RDF and SPARQL (but notice that monotonic SPARQL queries have also been studied in the recent tutorial paper [AP11]).

• We study the complexity of evaluating queries expressed in a monotone subset of the stSPARQL query language over stRDF³ databases. As it is standard in incomplete information settings, we concentrate on the data complexity of computing the certain answer to monotone stSPARQL queries. For this problem, we give upper bounds that correspond to the cases when the query and the database contain spatial constraints from the various spatial constraint languages we have already defined in Deliverable 2.1.
The technical development of this deliverable forms the theoretical basis of the work on stRDF and stSPARQL foreseen in WP2 of TELEIOS. The implementation of stSPARQL is now the next step of the relevant work in the project and it is undertaken in WP4 “Query processing - Image annotations”.

The theoretical work in this deliverable will guide the implementation work we undertake in WP4 in the context of system Strabon. The semantics and computational complexity analysis of Chapters 3 and 4 have already been guiding us to implement stRDF and stSPARQL in Strabon. The material in Chapter 5 of this deliverable will guide our implementation of stRDF.

The organization of this deliverable is the following. Chapter 2 presents background which is necessary for following the technical material given in the rest of this deliverable. Chapter 3 presents the model stRDF and stSPARQL verbatim from [KK10] so that the reader has at his or her disposal the formal definition and semantics of stSPARQL before the complexity results are presented. Chapter 4 studies the complexity of query evaluation for stSPARQL over stRDF graphs. Then, Chapter 5 considers the case of stRDF and continues the work on its semantics presented in Deliverable 2.1. In addition, it studies the complexity of query evaluation. Finally, Chapter 6 concludes this deliverable and outlines our plan for future work.
2. Background

In this chapter we give some background information which is necessary for the reader to be familiar with, so that the technical material that follows in Chapters 3, 4, and 5 is understood.

The organization of the chapter is as follows. Section 2.1 discusses first-order languages, structures, and theories, and the complexity of deciding whether a formula of a first-order language is true in the intended structure. It also gives well-known examples of first-order languages, structures, and theories that are useful for modelling geometric objects. Section 2.2 introduces a few more first-order languages that we take verbatim from Section 5.2 of Deliverable 2.1. These are languages expressing qualitative relations among regions in space and are useful in the data model stRDF studied in Chapter 5. Section 2.3 presents well-known theoretical results for the languages defined in Section 2.2 that are about satisfiability and validity, and variable and quantifier elimination. These results will be useful for studying the complexity of evaluating stSPARQL over the stRDF data model, when considering different spatial constraint languages. Section 2.4 defines the relational constraint database model which has been the inspiration for the models stRDF and stRDF∗, and the query language stSPARQL discussed in Chapters 3, 4, and 5. Section 2.5 discusses a few computational problems involving polyhedra. These will be useful in the complexity analysis carried out for stSPARQL in Chapter 4. Finally, Section 2.6 concludes the chapter.

2.1 First-Order Languages, Structures, and Theories

First of all we give some brief definitions from first-order logic. For an introduction to logic, see [End01].

A first-order language \( L \) consists of a set of function symbols (e.g., +, *), a set of predicate symbols (e.g., <, =) and a set of constant symbols. A term is recursively defined as follows: a constant or a variable is a term, \( f(t_1, \ldots, t_n) \) is a term if \( t_1, t_2, \ldots, t_n \) are terms and \( f \) is an \( n \)-ary function symbol. An atomic formula is defined as \( P(t_1, t_2, \ldots, t_n) \) where \( t_1, \ldots, t_n \) are terms and \( P \) an \( n \)-ary predicate. A formula is recursively defined as follows: an atomic formula is a formula, \( \lnot \varphi, \varphi \lor \psi, \varphi \land \psi, \exists x \varphi, \forall x \varphi \) are formulae, where \( x \) is a variable and \( \varphi, \psi \) are formulae. A sentence is a formula with no free occurrences of variables. A formula is quantifier-free if no quantifier occurs in it.

A structure for a first-order language \( L \) is a pair \( D = \langle D, \delta \rangle \), where \( D \) is a non-empty set called the domain (e.g., \( \mathbb{R}, \mathbb{Q}, \mathbb{Z}, \mathbb{N} \)) and \( \delta \) is a mapping of function symbols, predicate and constant symbols of \( L \) to appropriate functions, relations and elements of \( D \), respectively. Sometimes we also denote a structure for \( L \) by \( M_{L} \) instead of \( D \). A valuation of a formula \( \varphi \) in \( D \) is a mapping of all variables in \( \varphi \) to elements of \( D \). We say that a formula \( \psi \) is satisfied by a structure \( D \) with valuation \( v \), denoted by \( D \models \varphi[v] \) if \( \varphi \) is true in \( D \) after substituting the variables of \( \varphi \) by the elements of \( D \) according to \( v \). If \( \psi \) is a sentence then we say that \( D \) is a model of \( \psi \) iff \( \psi \) is true in \( D \). Note that for a sentence \( \psi \), either \( \psi \) is true for all valuations \( v \) of \( D \) or it is false. Thus, the truth of a sentence \( \psi \) is independent of any valuation. If \( \psi \) is a sentence, we write that \( D \models \psi \) iff \( \psi \) is true in \( D \). For any structure \( D \), we define the theory of \( D \), written as \( ThD \), to be the set of all sentences that are true in \( D \).

Symbols of a first-order language \( L \) are the function, predicate, and constant symbols, logic connectives and quantifiers, variables and parentheses. In Sections 2.4 and 2.3 of this deliverable we use languages for arithmetic that have the usual function symbols +, −, and *, and predicate symbol <. We often start from a well-known structure and assume the obvious first-order language for writing formulae about elements of this structure.
Two interesting structures that we use heavily in Sections 2.4 and 2.3 are the reals with order and addition, \( \langle \mathbb{R}, <, + \rangle \) denoted \( \mathcal{R}_+ \), and reals with order, addition and multiplication, \( \langle \mathbb{R}, <, +, * \rangle \), denoted \( \mathcal{R} \). An example of a formula of \( \mathcal{R}_+ \) is

\[
(\exists \varepsilon)(\varepsilon > 0 \land (\forall y)((-\varepsilon < y - x \land y - x < \varepsilon \rightarrow y > 3 \land y < 10}))
\]

which is true for values of \( x \in [3, 10] - \{3, 10\} \). Additionally, we can easily check that the sentence

\[
(\forall x)(\exists y x = y + y)
\]

is true in \( \mathcal{R}_+ \) and false in \( \langle \mathbb{N}, <, + \rangle \). Similarly, an example formula of \( \mathcal{R} \) is

\[
(\exists \varepsilon)(\varepsilon > 0 \land (\forall y)((-\varepsilon < y_1 - x_1 \land y_1 - x_1 < \varepsilon) \land (-\varepsilon < y_2 - x_2 \land y_2 - x_2 < \varepsilon) \rightarrow (y_1 + y_2 \ast y_2 < 4}))
\]

which is true for values of \( x_1, x_2 \) that lay in the interior of the disk with center in the origin and radius 2. Moreover, the sentence \( (\exists x) x \ast x = -1 \) is false in \( \mathcal{R} \) but true in \( \langle \mathbb{C}, <, +, * \rangle \).

### 2.1.1 The Complexity of Logical Theories for Algebra and Geometry

We define the length of a formula \( \varphi \) to be the number of bits needed to represent each symbol of the formula. We focus on the following decision problem for a structure \( D \) of a first-order language \( \mathcal{L} \):

\[
\text{Given a sentence } \psi \text{ of } \mathcal{L} \text{ with length } n, \text{ is } \psi \in \text{Th}_D ?
\]

Sometimes this problem can be solved by eliminating all quantifiers of \( \psi \), i.e., reducing \( \psi \) to a formula equivalent to \( \text{true} \) or \( \text{false} \). In this case, the decision problem reduces to a quantifier elimination problem.

We assume that the reader is familiar with computational complexity and the following hierarchy for complexity classes [Pap94]:

\[
AC^0 \subseteq \text{LOGSPACE} \subseteq \text{NL} \subseteq AC, NC \subseteq \text{PTIME} \subseteq NP \subseteq \#\text{PTIME} \subseteq \text{PSPACE}
\]

\[
\text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{EXPSPACE} \subseteq 2\text{-EXPTIME} \subseteq 2\text{-EXPSPACE}
\]

As in [Rev98], we denote by \( \exists QE \text{Th}(M_\mathcal{L}) \) the problem of whether the existential theory of \( M_\mathcal{L} \) admits quantifier elimination. Note here the distinction between an algorithm for the decision problem and an algorithm for quantifier elimination. An algorithm for the first problem is more specific in the sense that it determines whether a formula in a theory \( M_\mathcal{L} \) is \( \text{true} \) or \( \text{false} \). Thus, the algorithm is concerned only with sentences. An algorithm for the second case is more general, because given any quantified formula that is expressed in a language \( \mathcal{L} \) and interpreted over the structure \( M_\mathcal{L} \) and possibly containing free variables, it outputs an unquantified formula that is equivalent to the input formula.

In the following we present known complexity results for the decision and quantifier elimination problem for the reals with order and addition \( \mathcal{R}_+ \), and reals with order, addition and multiplication \( \mathcal{R} \). We also present complexity results for interesting subcases, such as the respective existential theories (i.e., \( \exists \text{Th}(\mathcal{R}_+) \) and \( \exists \text{Th}(\mathcal{R}) \)), formulae containing free-variables and only existential quantifiers, formulae with fixed number of quantifier alternations, and formulae with fixed number of quantified or free variables.
2.1.2 Complexity Results for $\mathcal{R}_+$

One of the first works that studied the computational complexity for the decision problem of $\mathcal{R}_+$ is that of Ferrante and Rackoff [FR75]. They showed that the decision problem for sentences expressed in $\mathcal{R}_+$ requires exponential space and doubly exponential time (see [FR75, Theorem 2, p. 75]). The exact complexity bounds for the decision problem for sentences with $n$ atoms are $O(2^{cn})$ space and $O(2^{2dn})$ time, for some constants $c$ and $d$.

Another significant work on the computational complexity of $\mathcal{R}_+$ is that of Weispfenning [Wei88]. Weispfenning studies the decision and quantifier elimination problem for sentences expressed in $\mathcal{R}_+$. His technique employs quantifier elimination via Skolem terms based on [FR75]. He also gives lower bounds for these two problems and studies the case when the number of quantified variables or the number of quantifier alternations is fixed. In particular, he shows (Theorem 2.2, p. 13) that quantifier elimination for $\mathcal{R}_+$ requires doubly exponential space and doubly exponential time. The exact bound for the length of the output formula is $O(2^{2cn})$ and the time bound for quantifier elimination is $O(2^{2dn})$, where $n$ is the length of the input formula and $c$ some positive constant. Regarding the decision problem, he shows (Theorem 2.6, p. 14) that it can be solved in EXPSPACE and 2-EXPTIME by giving upper bounds that are similar with those of Ferrante and Rackoff [FR75]. He then proves that these bounds are tight, i.e., they have corresponding lower bounds. Moreover, he considers the interesting case of fixed dimension, i.e., when the number of quantified variables is bounded by a fixed non-negative integer $q$. Then, quantifier elimination requires $O(cn^k)$ space and time for some natural numbers $c, k$, and thus it is in PTIME. As for the decision problem, it is solvable in PTIME, by first eliminating quantifiers and then deciding the truth value of the resulting quantifier-free formula, which can be done again in PTIME. Last, Weispfenning studies the case of having the number of quantifier alternations be bounded by a natural number $a$ and the number of quantified variables in each block of quantifiers be bounded by a natural number $b$. In this case, quantifier elimination requires $O(n^{a(b)})$ space and time for some natural number $c$. The same time bound applies for the decision problem as well. Thus, for bounded number of alternations and bounded number of quantifiers in each block, both the quantifier elimination and decision problems are in PTIME. Note that for the less constrained case of formulae with a bounded number of quantifier alternations, but with each block of quantifiers involving an unbounded number of variables then both problems are in EXPTIME. From the above bound, we can infer another upper bound for the problem of quantifier elimination with only existential quantifiers (i.e., $\exists \text{Th}(\mathcal{R}_+)$). In this case, $a$ becomes equal to 1, and thus, we get a bound of $O(n^{2b})$, which is exponential in the number of variables.

Another, interesting result for the decision problem of formulae of $\mathcal{R}_+$ with a bounded number of quantifier alternations is that of [Son85]. Sontag shows that deciding a formula with $k$ (a constant number) quantifier alternations is complete for the class $\Sigma_p^k$, i.e., the polynomial hierarchy. A special case is the decision problem of the existential theory of $\mathcal{R}_+$, in which case we have that $k = 1$ and, thus, the problem is NP-complete.

2.1.3 Complexity Results for $\mathcal{R}$

In the more complex case of $\mathcal{R}$, there are many more works that study the computational complexity of the decision problem [GV92, Can88, Can93, Ren88, Ren92, BPR96, Bas97, Bas99, BOKR84] and elimination of quantifiers [Col75, DH88, Can93, Ren92, BPR96, Bas97, Bas99, BD04]. While someone might expect better worst-case complexity results for the case of $\mathcal{R}_+$ than those for $\mathcal{R}$, actually this is not the case, thus, making arithmetic over reals in general a very hard problem. The first algorithm for quantifier elimination in $\mathcal{R}$ was proposed by Collins [Col75], who developed the well-known technique known as Cylindrical Algebraic Decomposition (CAD). CAD has been widely adopted in works concerning quantifier elimination for reals. Though it is sensitive to the ordering of elimination of variables, it produces formulae without redundancies that are also the simplest than any other quantifier elimination technique can produce. This is the principal feature
that makes it very popular. However, the running time of the CAD technique does not depend on the number of quantifier alternations.

Collins showed that the time complexity of CAD is $(md)^c r^c$, where $n$ is the number of free and bound variables in the input formula, $m$ the number of polynomials, $d$ the maximum degree of each such polynomial in any variable, $r$ is the maximum length of any integer coefficient of any such polynomial, and $c$ is some constant. Thus, the complexity of his algorithm has a running time that is in 2-EXPTIME. In [DH88], Davenport and Heintz show that any quantifier elimination algorithm for $\mathcal{R}$ has worst-case running time $\Omega(2^{2^{n-1/3}})$ even when all the polynomials are linear (i.e., for $\mathcal{R}_+$). In a recent paper [BD07], Brown improves this lower bound to $\Omega(2^{2^{n-1/3}})$.

Another significant result is that of Ben-Or, Kozen, and Reif [BOKR84], who presented a decision procedure for $\mathcal{R}$, which requires singly exponential space and parallel exponential time, while for the case of fixed dimension it runs in NC. It has to be noted, that their technique has been used widely in subsequent works, such as [Ren88, KY85].

Grigor’ev and Vorobjov [GV92] studied the case of the existential theory of $\mathcal{R}$ providing a decision procedure with time complexity which is singly exponential in the number of variables. The best space result for the same case is due to Canny [Can88], who gave a decision procedure which requires polynomial space and singly exponential time even in the case of having a fixed number of variables. In [Can93], Canny improved the results of [Can88] by giving a decision procedure and a quantifier elimination algorithm for the existential theory of $\mathcal{R}$ which is in EXPTIME, with exact time complexity $m^{(n+1)}d^{O(n^3)}r^{(1+\epsilon)}$ for any $\epsilon > 0$. Thus, for fixed dimension his algorithms run in PTIME. Renegar [Ren88] improved [Can88] also by keeping the space complexity in PSPACE and the running time of the algorithm in PTIME for fixed dimension. The exact time complexity is $(md)^{O(n)}$.

Renegar extended his results in [Ren88] to the decision problem and quantifier elimination for the general case, and showed that the time complexity for the decision problem is $(md)^{\Pi(n)}$, where $n_i$ is the number of variables in each block of quantifiers, while for quantifier elimination is $(md)^{(l+1)\Pi(n)}$, where $l$ is the number of free variables [Ren92]. Notice that these bounds are exponential in the number of variables, and doubly exponential in the number of quantifier alternations. To see this, suppose that the input formula contains $a$ blocks of quantifiers, each one containing $n_i$ variables, such that $\sum_{i=1}^a n_i = n$. Then, $\Pi(n)$ can be bounded as follows: $\Pi(n_i) \leq n^a$. Thus, $(md)^{\Pi(n)} \leq (md)^{n^a}$.

Basu [BPR95] has presented both a decision procedure and a quantifier elimination algorithm the time and space complexities of which are similar to Renegar’s [Ren92], but the combinatorial (the dependence on the number of polynomials, $m$) and algebraic (the dependence on the degrees of the input polynomials, $d$) parts in the complexity bounds are cleanly separated in his bounds. In particular, the time complexity for the decision problem is $m^{\Pi(n+1)}d^{\Pi(n)}$, while the time and space complexity for the quantifier elimination problem is $m^{(l+1)\Pi(n+1)}d^{(l+1)\Pi(n)}$. In subsequent works [Bas97, Bas99], Basu improves the algorithm for quantifier elimination in a way that the combinatorial part of the complexity bound is independent of the number of free-variables, i.e., parameter $l$, deriving a time and space complexity bound of $m^{\Pi(n+1)}d^{\Pi(n)}$ and $m^{\Pi(n+1)}d^{\Pi(n)}|\phi|$ respectively, where $l' = \min(l, \tau\Pi(n_i + 1))$ for some constant $\tau$, and $|\phi|$ is the size of the input formula. Note that if each polynomial in the input formula depends only on a constant number of free variables, then the algebraic part in the complexity bounds of [Bas97, Bas99] is also independent of the number of free variables. Using a similar analysis, these time and space bounds both for the decision and the quantifier elimination problem are still singly exponential in the number of variables and doubly exponential in the number of quantifier alternations.

In the context of polynomial constraint databases, [KKR95] uses the work of [KY85] that provides an algorithm for cell decomposition for $\mathcal{R}$ to output a formula in DNF which is the result of the evaluation of a relational calculus query with real polynomial inequality constraints over polynomial.
constraint databases. Using the same ideas, one can see that the cell decomposition method of [KY85] provides a quantifier elimination algorithm for \( \mathcal{R} \) which is in NC for fixed dimension.

Table 2.1 summarizes all complexity results discussed above by giving the relevant complexity classes for each problem.

Table 2.1: Complexity results for the structures \( \mathcal{R}_+ \) and \( \mathcal{R} \) (symbol “—” means that the case is not applicable)

<table>
<thead>
<tr>
<th>General Formula</th>
<th>Fixed Dimension</th>
<th>Fixed Number of Quantifier Alternations</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Th(\mathcal{R}_+) )</td>
<td>EXPSPACE [FR75, Wei88]</td>
<td>NC [BOKR84]</td>
</tr>
<tr>
<td>( QE \mathcal{R}_+ )</td>
<td>2-EXPTIME [Wei88]</td>
<td>NC [KKR95]</td>
</tr>
<tr>
<td>( \exists Th(\mathcal{R}_+) )</td>
<td>NP [Son85]</td>
<td>PTIME [Ren92]</td>
</tr>
<tr>
<td>( \exists QE \mathcal{R}_+ )</td>
<td>EXPTIME [Wei88, Ren92]</td>
<td>NC [KKR95]</td>
</tr>
<tr>
<td>( Th(\mathcal{R}) )</td>
<td>EXPSPACE [BOKR84]</td>
<td>NC [BOKR84]</td>
</tr>
<tr>
<td>( QE \mathcal{R} )</td>
<td>2-EXPTIME [Ren92, Bas97]</td>
<td>NC [KKR95]</td>
</tr>
<tr>
<td>( \exists Th(\mathcal{R}) )</td>
<td>PSPACE [Can88, Ren92]</td>
<td>PTIME [Ren88, Can93]</td>
</tr>
<tr>
<td>( \exists QE \mathcal{R} )</td>
<td>EXPTIME [Ren92]</td>
<td>NC [KKR95]</td>
</tr>
</tbody>
</table>

2.2 First-Order Languages of Spatial Constraints

We now recall the first-order languages of the spatial constraints we defined in Section 5.2 of Deliverable 2.1 and used in the context of the model stRDF to represent information about spatial regions for which we have no exact coordinates i.e., the known information about them is incomplete, indefinite or qualitative in nature. We will revisit these languages in Chapter 5 when we are discussing query evaluation for stRDF.

2.2.1 The Language TCL

The language TCL (Topological Constraint Language) allows us to represent topological properties of non-empty, regular closed subsets of \( \mathbb{Q}^2 \) (we will call these subsets regions for brevity). We prefer to have \( \mathbb{Q}^2 \) instead of \( \mathbb{R}^2 \) as our set, since in an implementation, relevant information will be represented by floating point numbers. Thus, rational numbers are the closest abstraction. 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. A TCL-constraint is a disjunction of atomic formulæ of TCL involving the same two variables. For example, the following are TCL-constraints:

\[ r_1 NTPP r_2, \quad r_2 PO r_3 \lor r_2 EQ r_3 \]

In TCL and all other languages of this section, we also assume the existence of constraints true and false with obvious semantics.

In Chapter 5 where the constraint languages of this section such as TCL will be used in the context of the data model stRDF, we will often use the terminology atomic \( L \)-constraints to refer to atomic formulæ of an arbitrary constraint language \( L \).

The intended structure for TCL, denoted by \( M_{TCL} \), has the set of regions as its domain, and interprets each of the predicate symbols given above by the corresponding topological relation of
The theoretical results on query processing for RDF/SPARQL with time and space and their syntax. This will not create any technical problems for us for the development of this deliverable.

Table 2.2: Definition of the various relations of RCC (relations in bold are included in RCC-8)

<table>
<thead>
<tr>
<th>Relation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC(x, y)</td>
<td>$\equiv_{df} \neg C(x, y)$</td>
</tr>
<tr>
<td>P(x, y)</td>
<td>$\equiv_{df} \forall z [C(z, x) \rightarrow C(z, y)]$</td>
</tr>
<tr>
<td>PP(x, y)</td>
<td>$\equiv_{df} P(x, y) \land \neg P(y, x)$</td>
</tr>
<tr>
<td>EQ(x, y)</td>
<td>$\equiv_{df} P(x, y) \land P(y, x)$</td>
</tr>
<tr>
<td>O(x, y)</td>
<td>$\equiv_{df} \exists z[P(z, x) \land P(z, y)]$</td>
</tr>
<tr>
<td>PO(x, y)</td>
<td>$\equiv_{df} O(x, y) \land \neg P(x, y) \land \neg P(y, x)$</td>
</tr>
<tr>
<td>DR(x, y)</td>
<td>$\equiv_{df} \neg O(x, y)$</td>
</tr>
<tr>
<td>TPP(x, y)</td>
<td>$\equiv_{df} PP(x, y) \land 3\exists [EC(z, x) \land EC(z, y)]$</td>
</tr>
<tr>
<td>EC(x, y)</td>
<td>$\equiv_{df} C(x, y) \land \neg O(x, y)$</td>
</tr>
<tr>
<td>NTPP(x, y)</td>
<td>$\equiv_{df} PP(x, y) \land \neg 3\exists [EC(z, x) \land EC(z, y)]$</td>
</tr>
<tr>
<td>Pi(x, y)</td>
<td>$\equiv_{df} P(y, x)$</td>
</tr>
<tr>
<td>Pi(x, y)</td>
<td>$\equiv_{df} PP(y, x)$</td>
</tr>
<tr>
<td>TPPi(x, y)</td>
<td>$\equiv_{df} TPP(y, x)$</td>
</tr>
<tr>
<td>NTPPi(x, y)</td>
<td>$\equiv_{df} NTPP(y, x)$</td>
</tr>
</tbody>
</table>

RCC-8 [RCC92a] shown in Table 2.2 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 languages PCL and RCL considered below deal with regions with specific shapes (polygons and rectangles respectively) that are useful for encoding the geospatial data sets considered in TELEIOS.

### 2.2.2 The Language PCL

The language PCL (Polygon Constraint Language) allows us to represent topological properties of polygons in $\mathbb{Q}^2$. TCL is a first-order language with the same 6 predicate symbols as TCL, but also constant symbols representing polygons in $\mathbb{Q}^2$. From the two standard notations for polygons (sequence of vertices or conjunctions of linear constraints), we choose to write polygons using the constraint notation and include these constraints in quote $^*$ (e.g., ”$x \leq 1, x \geq 0, y \leq 1, y \geq 0$" is a rather long constant that denotes the unit rectangle with lower-left vertex (0,0)). The terms and atomic formulae of PCL are defined as follows. Constants and variables are terms. An atomic formula of PCL is a formula of the form $t_1 R t_2$ where $t_1, t_2$ are terms and $R$ is one of the above predicates. A PCL-constraint is a disjunction of atomic formulae of PCL involving the same pair of terms. For example, the following are PCL-constraints:

$$r_1 \ NTPP \ r_2 \lor r_1 \ TPP \ r_2, \ r_2 \ NTPP \ "x \leq 1, x \geq 0, y \leq 1, y \geq 0"$$

The intended structure for PCL, denoted by $M_{PCL}$, has the set of polygons in $\mathbb{Q}^2$ as its domain. $M_{PCL}$ interprets each constant symbol by the corresponding polygon in $\mathbb{Q}^2$, and each of the predicate symbols by the corresponding topological relation of RCC8 [RCC92a].

### 2.2.3 The Language RCL

The language RCL (Rectangle Constraint Language) allows us to capture spatial constraints (e.g., topological or directional) involving rectangles with sides parallel to the axes in $\mathbb{Q}^2$ (we will call

$^*$We choose to live with this non-standard way of writing formulae in quotes without going into the details of their syntax. This will not create any technical problems for us for the development of this deliverable.
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 \( Q \) for rational constants, and the sort \( R \) for boxes. The set of non-logical symbols of RCL includes: all rational constants of sort \( Q \), function symbols \( LL_x(\cdot) \), \( LL_y(\cdot) \), \( UR_x(\cdot) \), \( UR_y(\cdot) \) of sort \((R,Q)\), and predicate symbol \(<\) of sort \((Q,Q)\).

The terms and atomic formulae of RCL are defined as follows. Constants of sort \( Q \) and variables of sort \( R \) are terms. If \( r \) is a variable of sort \( R \) then \( LL_x(r), LL_y(r), UR_x(r) \) and \( UR_y(r) \) is a term of sort \( Q \). An atomic formula of RCL is a formula of the form \( t_1 \sim t_2 \) where \( \sim \) is \( = \) or \( \leq \) and \( t_1, t_2 \) are terms involving function symbols with the same subscript \((x, y)\) or rational constants. \( = \) is the equality predicate for sort \( Q \); we will not use the equality predicate for sort \( R \) in our formulae.

The intended structure for RCL, denoted by \( \mathcal{M}_{RCL} \), interprets each non-logical symbol as follows. Each rational constant is interpreted by its corresponding rational number. 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 \( t_1 \sim t_2 \) where \( \sim \) is \( = \), \( < \), \( > \), \( \leq \) or \( \geq \) and \( t_1, t_2 \) are terms (the predicates \(<\), \( \leq \), and \( \geq \) are defined as usual).

**Example 2.1.** The formulae

\[
LL_x(r_2) < LL_x(r_1), \quad UR_y(r_1) < LL_y(r_2), \quad UR_x(r_1) < UR_y(r_2)
\]

\[
LL_x(r_2) < LL_x(r_3), \quad LL_y(r_2) < LL_y(r_3), \quad UR_x(r_3) < UR_y(r_2), \quad UR_y(r_3) < UR_y(r_2)
\]

are RCL-constraints. The conjunction of these constraints tells us that box \( r_1 \) is to the south of box \( r_2 \) (first line) and box \( r_3 \) is a non-tangential proper part of \( r_2 \) (second line). Three boxes satisfying the constraints are depicted in Figure 2.1.

### 2.2.4 The Language TRCL

We also define the language TRCL which essentially extends TCL with the ability to express a topological relation of a region with a box in \( \mathbb{Q}^2 \). TRCL has 2 sorts: the sort of regions \( R \) and the sort of boxes \( B \). In TRCL, we only allow constants of sort \( B \). These constants are written using order constraints of the form \( x \leq c, x \geq c, y \leq c \) and \( y \geq c \), where \( c \) is a rational constant. An example of such a constant is “\( x \geq 0 \land x \leq 3000 \land y \geq 0 \land y \leq 3000^\prime \)”. We assume that the order constraints defining these constants are in normal form, i.e., there are exactly four constraints, two on variable \( x \) and two on variable \( y \) defining a box in \( \mathbb{Q}^2 \) in the obvious way.

The atomic formulas of TRCL are the ones allowed by TCL but we also allow constants of sort \( B \) to appear in these formulae, e.g., \( x \ NTPP \ “x \geq 0 \land x \leq 3000 \land y \geq 0 \land y \leq 3000^\prime “ \). The rest of the formal definition of TRCL is obvious, so we omit it for brevity.

The above four languages have been defined so that the reader can appreciate the scope of modeling possibilities for geospatial applications like the ones studied in TELEIOS. Obviously, many more possible languages can be defined depending on the richness of spatial information that one would like to represent. There is a well-known trade-off among the expressivity of a given language and its computational properties. This is something that we explore briefly in the following section.
2.3 Complexity Results for First-Order Languages of Spatial Constraints

This section presents some known algorithms and related complexity results for the constraint languages presented earlier in Section 2.2. The focus is on the satisfiability and validity of formulae of each language, the variable and quantifier elimination procedures (if any) and the computational complexity of these problems.

2.3.1 Satisfiability and Validity

When studying the satisfiability and validity problems for a language $\mathcal{L}$ we distinguish between the problems of satisfiability of a first-order formula of $\mathcal{L}$ (i.e., an arbitrary first-order formula possibly with existential and universal quantifiers) and the satisfiability of a $\mathcal{L}$-constraint formula. We denote the first problem as $\text{SAT}_L$ and the second as $\text{SAT}_{L^c}$. We shall refer to these problems either by describing their formulae explicitly or using the $\text{SAT}$ name convention. Many of the results of this section come verbatim from [Kou97] where they have been used in the context of indefinite constraint databases.

2.3.2 Variable and Quantifier Elimination

We now define two operations that will be useful for the rest of our developments: variable elimination and quantifier elimination. Variable elimination is an algebraic operation; its logical counterpart is quantifier elimination. We will now define these two notions and then use the one which is more appropriate in each case. We will always assume that we have to deal with formulae of finite length.

Notation 2.1. The vector of symbols $(o_1, \ldots, o_n)$ will be denoted by $\vec{o}$. The natural number $n$ will be called the size of $\vec{o}$ and will be denoted by $|\vec{o}|$. This notation will be used for vectors of
variables but also for vectors of domain elements. Variables will be denoted by \( x, y, z, t \), etc. and vectors of variables by \( \mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{t} \), etc. If \( \mathbf{x} \) and \( \mathbf{y} \) are vectors of variables then \( \mathbf{x} \setminus \mathbf{y} \) will denote the vector obtained from \( \mathbf{x} \) by deleting the variables in \( \mathbf{y} \). If \( \mathbf{x} \) and \( \mathbf{y} \) are vectors of variables and every variable in \( \mathbf{x} \) is also contained in \( \mathbf{y} \) then we will write \( \mathbf{x} \subseteq \mathbf{y} \). If \( \mathbf{x} \) is a vector of variables then \( \mathbf{x}^0 \) will be a vector of constants of the same size. The notation \( \mathbf{x}^0 \setminus \mathbf{y}^0 \) is similarly defined for vectors of constants.

**Definition 2.1.** Let \( \mathcal{L} \) be a many-sorted first-order language. The class of \( \mathcal{L} \)-constraints admits variable elimination iff for every boolean combination \( \phi \) of \( \mathcal{L} \)-constraints in variables \( \mathbf{z} \), and every vector of variables \( \mathbf{z} \subseteq \mathbf{x} \), there exists a disjunction \( \phi' \) of conjunctions of \( \mathcal{L} \)-constraints in variables \( \mathbf{x} \setminus \mathbf{z} \) such that

1. If \( \mathbf{x}^0 \) is a solution of \( \phi \) then \( \mathbf{x}^0 \setminus \mathbf{z}^0 \) is a solution of \( \phi' \).
2. If \( (\mathbf{x} \setminus \mathbf{z})^0 \) is a solution of \( \phi' \) then this solution can be extended to a solution \( \mathbf{x}^0 \) of \( \phi \).

The following definition will also be useful.

**Definition 2.2.** Let \( \mathcal{L} \) be a many-sorted first-order language. The class of \( \mathcal{L} \)-constraints is weakly closed under negation if the negation of every \( \mathcal{L} \)-constraint is equivalent to a disjunction of \( \mathcal{L} \)-constraints.

The following proposition, whose proof can easily be done by induction, shows that if the class of \( \mathcal{L} \)-constraints is weakly closed under negation then we can determine whether it also admits variable elimination by considering only conjunctions of \( \mathcal{L} \)-constraints.

**Proposition 2.1.** Let \( \mathcal{L} \) be a many-sorted first-order language. The class of \( \mathcal{L} \)-constraints admits variable elimination if

- it is weakly closed under negation, and
- for every conjunction \( \theta \) of \( \mathcal{L} \)-constraints in variables \( \mathbf{x} \) and every vector of variables \( \mathbf{z} \subseteq \mathbf{x} \), there exists a disjunction \( \theta' \) of conjunctions of \( \mathcal{L} \)-constraints in variables \( \mathbf{x} \setminus \mathbf{z} \) such that
  
  1. If \( \mathbf{x}^0 \) is a solution of \( \theta \) then \( \mathbf{x}^0 \setminus \mathbf{z}^0 \) is a solution of \( \theta' \).
  2. If \( (\mathbf{x} \setminus \mathbf{z})^0 \) is a solution of \( \theta' \) then this solution can be extended to a solution \( \mathbf{x}^0 \) of \( \theta \).

The importance of this proposition is that it provides us with the following algorithm for eliminating variables \( \mathbf{z} \) from a Boolean combination \( \phi \) of \( \mathcal{L} \)-constraints in variables \( \mathbf{x} \):

1. Transform \( \phi \) into a formula where negation applies only to \( \mathcal{L} \)-constraints (by applying De Morgan’s laws and laws for negation).
2. Substitute every negated \( \mathcal{L} \)-constraint by its equivalent disjunction of \( \mathcal{L} \)-constraints.
3. Transform \( \phi \) into disjunctive normal form \( \theta_1 \lor \cdots \lor \theta_n \) where each \( \theta_i \) is a conjunction of \( \mathcal{L} \)-constraints.
4. Perform variable elimination in each \( \theta_i \), i.e., substitute each disjunct \( \theta_i \) of \( \phi \) by the equivalent disjunction of conjunctions of \( \mathcal{L} \)-constraints in variables \( \mathbf{x} \setminus \mathbf{z} \).

For this algorithm to be effective, we must know how to perform Steps 2 and 4. For most languages of interest (e.g., TCL, PCL, RCL, or TRCL) step 2 will be obvious. Step 4 will be the challenging step.
Let us give an example of a variable elimination algorithm which is well-known. Consider a set (conjunction) of linear inequalities from the first-order language $\mathcal{L}_{LIN}$ of linear inequalities over the rational numbers considered among others in the stRDF/stSPARQL work of [KK10] which is presented in Chapter 3 of this deliverable. Variable elimination for linear inequalities can be performed using the well-known Fourier’s algorithm which can be summarized as follows [Sch86a]. Any weak linear inequality involving a variable $x$ can be written in the form $x \leq r_u$ or $x \geq r_l$, i.e., it gives an upper or a lower bound on $x$. Thus if we are given two linear inequalities, one of the form $x \leq r_u$ and the other of the form $x \geq r_l$, we can eliminate $x$ and obtain the inequality $r_l \leq r_u$. Obviously, $r_l \leq r_u$ is a logical consequence of the given inequalities. In addition, any solution of $r_l \leq r_u$ can be extended to a solution of the given inequalities (simply by choosing for $x$ any value between the values of $r_l$ and $r_u$). Following this observation, Fourier’s elimination algorithm forms all pairs $x \leq r_u$ and $x \geq r_l$, eliminates $x$ and returns the resulting inequalities together with the inequalities that do not involve variable $x$. If pairs of the form $x \leq r_u$ and $x \geq r_l$ cannot be formed (e.g., because all inequalities involving $x$ are of the form $x \leq r_u$), then all inequalities involving $x$ are discarded and the remaining inequalities are retained. The generalization of this algorithm to the case where we also have equalities and strict inequalities is obvious.

The following is an example of variable elimination using Fourier’s algorithm.

**Example 2.2.** Eliminating variable $x_1$ from the set of inequalities

\[
x_3 \leq x_1, \ x_5 \leq x_1, \ x_1 - 3x_2 + 5x_6 \leq 2, \ x_4 \leq x_5
\]

gives

\[
x_3 - 3x_2 + 5x_6 \leq 2, \ x_5 - 3x_2 + 5x_6 \leq 2, \ x_4 \leq x_5.
\]

If we also eliminate variable $x_5$, we get

\[
x_3 - 3x_2 + 5x_6 \leq 2, \ x_4 - 3x_2 + 5x_6 \leq 2.
\]

Let us now consider the languages and classes of constraints defined in Section 2.2.

**Proposition 2.2.** The class of RCL-constraints admits variable elimination and is weakly closed under negation.

The result is trivial concerning “weakly closed under negation”. Regarding variable elimination, we give an example instead of the easy formal proof.

**Example 2.3.** Let us consider the following set of RCL-constraints taken from Example 2.1

\[
LL_x(r_2) < LL_x(r_1), \ UR_y(r_1) < LL_y(r_2), \ UR_x(r_1) < UR_y(r_2)
\]

\[
LL_x(r_2) < LL_x(r_3), \ LL_y(r_2) < LL_y(r_3), \ UR_x(r_3) < UR_y(r_2), \ UR_y(r_3) < UR_y(r_2)
\]

We would like to eliminate variable $r_2$. We will proceed in two steps. First, we introduce the implicit constraints that capture the relationships between the $x$- and $y$-coordinates of each box. The given set of constraints is now transformed into the following equivalent set of order constraints:

\[
LL_x(r_2) < LL_x(r_1), \ UR_y(r_1) < LL_y(r_2), \ UR_y(r_1) < UR_y(r_2)
\]

\[
LL_x(r_2) < LL_x(r_3), \ LL_y(r_2) < LL_y(r_3), \ UR_x(r_3) < UR_x(r_2), \ UR_y(r_3) < UR_y(r_2)
\]

\[
LL_x(r_1) < UR_x(r_1), \ LL_y(r_1) < UR_y(r_1)
\]

\[
LL_x(r_2) < UR_x(r_2), \ LL_y(r_2) < UR_y(r_2)
\]

\[
LL_x(r_3) < UR_x(r_3), \ LL_y(r_3) < UR_y(r_3)
\]
We now treat terms as variables and use Fourier elimination for eliminating the term \( LL_y(r_2) \) and get:

\[
LL_x(r_2) < LL_x(r_1), \quad UR_y(r_3) < LL_y(r_2), \quad UR_x(r_1) < UR_x(r_2)
\]

\[
LL_x(r_2) < LL_x(r_3), \quad LL_y(r_2) < LL_y(r_3), \quad UR_x(r_3) < UR_x(r_2), \quad UR_y(r_3) < UR_y(r_2)
\]

\[
LL_x(r_1) < UR_x(r_1), \quad LL_y(r_1) < UR_y(r_1)
\]

\[
LL_x(r_2) < UR_x(r_2), \quad LL_y(r_2) < UR_y(r_2)
\]

\[
LL_x(r_3) < UR_x(r_3), \quad LL_y(r_3) < UR_y(r_3)
\]

\[
UR_y(r_1) < LL_y(r_3)
\]

\[
UR_y(r_1) < UR_y(r_2)
\]

If we also eliminate the terms \( LL_x(r_2), \ UR_x(r_2), \) and \( UR_y(r_2) \), we get:

\[
LL_x(r_2) < LL_x(r_1), \quad UR_x(r_1) < UR_x(r_2)
\]

\[
LL_x(r_2) < LL_x(r_3), \quad UR_x(r_3) < UR_x(r_2), \quad UR_y(r_3) < UR_y(r_2)
\]

\[
LL_x(r_1) < UR_x(r_1), \quad LL_y(r_1) < UR_y(r_1)
\]

\[
LL_x(r_2) < UR_x(r_2)
\]

\[
LL_x(r_3) < UR_x(r_3), \quad LL_y(r_3) < UR_y(r_3)
\]

\[
UR_y(r_1) < LL_y(r_3)
\]

\[
UR_y(r_1) < UR_y(r_2)
\]

From this set of constraints and by removing the implicit constraints, we get the single constraint:

\[
UR_y(r_1) < LL_y(r_3)
\]

Inspecting now the above constraint in relation to Figure 2.1, we observe that all constraints between boxes \( r_1, r_3 \) and boxes \( r_2, r_3 \) have been ruled out (due to the fact that we chose to eliminate variable \( r_2 \)), while a new one has been introduced (the above) that preserves the property between \( r_1 \) and \( r_3 \) (i.e., \( r_1 \) is south of \( r_3 \)).

**Proposition 2.3.** The Fourier-Motzkin variable elimination algorithm has time and space complexity \( O(m2^n) \), where \( m \) is the number of inequalities and \( n \) the number of variables.

Let us now consider elimination of quantifiers.

**Definition 2.3.** Let \( Th(\mathcal{L}) \) be the theory of language \( \mathcal{L} \). \( Th(\mathcal{L}) \) admits elimination of quantifiers \( \text{iff} \) for every formula \( \phi \) there is a disjunction \( \phi' \) of conjunctions of \( \mathcal{L} \)-constraints such that \( Th(\mathcal{L}) \models \phi \equiv \phi' \).

This definition is stronger than the traditional one where \( \phi' \) is simply required to be quantifier-free. We require \( \phi' \) to be in the above form because we do not want to deal with negations of \( \mathcal{L} \)-constraints.
2.3.3 Results for **TCL**

**Satisfiability and Validity**

The validity problem for arbitrary formulae of TCL is undecidable [Grz51, Dor98]. The satisfiability problem for conjunctions of TCL-constraints is NP-complete but there are tractable subcases [GPP95, RN99]. There have been implemented algorithms that determine the satisfiability of conjunctions of TCL-constraints. For the general case, these are backtracking algorithms [RN01], while for the tractable cases these are algorithms relying on path consistency for the corresponding constraint networks [RN99, RN07].

There are also interesting results on related spatial logics that consider more expressive languages with connectedness predicates, Boolean algebra operators for forming new regions, etc. [WZ00] extends the atomic regions RCC-8 to be boolean combinations of atomic regions. It is shown that \( \text{SAT}_{\mathbb{L}} \) is NP-complete when considering arbitrary topological spaces, but it becomes PSPACE-complete for connected topological spaces or Euclidean ones. [KPHZ10] shows that extending RCC-8 and Boolean RCC-8 with connectedness and/or interior operations, the respective languages become sensitive to the topological space in which they are interpreted.

Apart from these results, [JD97] presents a complete classification of tractability for RCC-5 for the \( \text{SAT}_{\mathbb{L}} \) problem.

**Variable and Quantifier Elimination**

**Proposition 2.4.** The language TCL does not admit quantifier elimination.

**Proof.** This is easy to see since the problem of satisfiability is undecidable; if it would admit quantifier elimination, then we would immediately have a decision procedure for these languages. \( \square \)

Bennett has discussed the issue of quantifier elimination in TCL in [Ben97].

2.3.4 Results for **PCL**

**Satisfiability and Validity**

The validity problem for arbitrary formulae of PCL is also undecidable [Dor98]. The satisfiability problem for conjunctions of PCL-constraints is also NP-hard since restricting our attention to polygons does not change the difficulty of the problem. This particular restriction of regions to be polygons has not been studied in the literature and no detailed complexity results or algorithms are known. [LL10] is a recent paper that studies binary topological relations among convex planar regions.

**Variable and Quantifier Elimination**

**Proposition 2.5.** The language PCL does not admit quantifier elimination.

**Proof.** This is easy to see since the problem of satisfiability is undecidable; if it would admit quantifier elimination, then we would immediately have a decision procedure for these languages. \( \square \)
2.3.5 Results for RCL

Satisfiability and Validity

Reasoning in RCL is easier than in the previous two languages. The restricted syntax of RCL allows us to reason about boxes by considering the constraints involving functions with a subscript $x$ and functions with a subscript $y$ independently. It is easy to see that the validity problem for arbitrary formulae of RCL is in PSPACE. The result follows from the results of Koubarakis on similar language of points and intervals [Kou94a]. The satisfiability problem for conjunctions of RCL-constraints can be solved in PTIME using well-known results from the Point Algebra [VKvB89, Kou06].

One could similarly define another language, let us call it RCL’, that has rectangles in $Q^2$ as constants, variables ranging over boxes and the $13 \times 13$ basic binary relations from the Rectangle Algebra studied by Balbiani and others [BCdC99, PT97] as predicates. The validity problem in RCL’ can similarly be proved to be in PSPACE. The satisfiability problem for conjunctions of RCL’-constraints is NP-complete but there are tractable subcases [BCdC99].

Recently, [LLR09] considered the problem of deciding consistency for the combination of a RCC-8 constraint network with either a constraint network in Rectangle Algebra (RA) or Cardinal Direction Calculus (CDC). The paper shows that deciding consistency over the combination of a RCC-8 and a RA network containing only the basic relations of RCC-8 and RA is PTIME, while that of RCC-8 and a CDC network again with basic relations only is NP-complete.

Variable and Quantifier Elimination

**Proposition 2.6.** The language RCL admits quantifier elimination.

*Proof.* Let us assume that $\phi$ is a formula of RCL. The standard algorithm to eliminate quantifiers proceeds by computing the prenex normal form of $\phi$ and then eliminating quantifiers starting from the innermost one. Eliminating quantifier ($Qx$) from ($Qx\theta$), where $\theta$ is quantifier-free can be done by transforming $\theta$ in disjunctive normal form and then eliminating variable $x$ from each disjunct. The result now follows from Proposition 2.2.

Thus, as with the computational complexity properties of the problems of satisfiability and validity, RCL is more “well-behaved” than TCL and PCL with respect to variable and quantifier elimination.

2.3.6 Results for TRCL

Satisfiability and Validity

The validity problem for TRCL is undecidable given the undecidability of TCL. It is not difficult to say that the satisfiability problem for conjunctions of TRCL constraints is NP-complete. No detailed complexity results or algorithms are known for this language.

From the above, it is easy to conclude that RCL and RCL’ are first-order languages with better computational properties than TCL, PCL, and TRCL.
Variable and Quantifier Elimination

**Proposition 2.7.** The language TRCL does not admit quantifier elimination.

*Proof.* This is easy to see since the problem of satisfiability is undecidable; if it would admit quantifier elimination, then we would immediately have a decision procedure for these languages. \qed

### 2.4 The Constraint Database Model

We give here the definitions of the constraint database model. This model has been originally introduced in [KKR95]. For a textbook introduction see [Rev02].

A class of constraints \( \Phi \) is a set of atomic formulae of a first-order language \( \mathcal{L} \) with a structure \( D = \langle D, \delta \rangle \). Some well-studied classes of constraints (some of them related to the theories of the previous section) are:

- **Real polynomial inequality constraints:** the atomic formulae in \( \langle \mathbb{R}, +, \ast \rangle \) with predicate symbols \(<, \leq, >, \geq, =, \neq \) and constant symbols \( c \in \mathbb{R} \).
- **Linear inequality constraints:** the atomic formulae in \( \langle \mathbb{Q}, + \rangle \) with predicate symbols \(<, \leq, >, \geq, =, \neq \) and constant symbols \( c \in \mathbb{Q} \).
- **Dense linear order inequality constraints:** the atomic formulae of the form \( x \circ o k \) where \( x \) is a variable, \( k \) is either a variable or a constant and \( o \) is one of \( <, \leq, >, \geq, =, \neq \). The domain is a countably infinite set with dense linear order (e.g., \( \mathbb{Q} \)).
- **Equality constraints:** the atomic formulae of the form \( x \circ o k \) where \( x \) is a variable, \( k \) is either a variable or a constant and \( o \) is one of \( =, \neq \). The domain is a countably infinite set (e.g., \( \mathbb{N} \)).

Let \( \Phi \) be a class of constraints. A generalised \( k \)-tuple over variables \( x_1, x_2, \ldots, x_k \) is a finite conjunction \( \varphi_1 \land \varphi_2 \land \ldots \varphi_n \), where each \( \varphi_i, 1 \leq i \leq n \), is a constraint in \( \Phi \) with variables among \( x_1, x_2, \ldots, x_k \). A generalised relation of arity \( k \) is a finite set \( r = \{ \psi_1, \psi_2, \ldots, \psi_m \} \), where each \( \psi_i, 1 \leq i \leq m \), is a generalised \( k \)-tuple over the same variables \( x_1, x_2, \ldots, x_k \). The formula \( \phi_r \equiv \psi_1 \lor \psi_2 \lor \ldots \lor \psi_m \) is the formula of the language \( \mathcal{L} \) that corresponds to \( r \). The generalized relation \( r \) represents in a finite way the unrestricted (possibly infinite) \( k \)-ary relation which consists of all \( (a_1, \ldots, a_k) \in D^k \) such that \( \phi_r(a_1, \ldots, a_k) \) is true. A generalised database is a finite set of generalised relations.

We can query this generalised database using relational calculus with constraints. Let \( R_1, \ldots, R_n \) be predicate symbols and \( y_1, \ldots, y_m \) free variables. Let \( r_1, \ldots, r_n \) be generalised relations with the same arities as \( R_1, \ldots, R_n \). A relational calculus query with constraints is a formula \( q \) of a first-order language with equality where every atomic formula is either \( R_i(x_1, \ldots, x_j) \) or a formula from the class \( \Phi \) of constraints. The answer to a relational calculus query is defined as follows:

\[
\rho \equiv q[r_1|R_1, r_2|R_2, \ldots, r_n|R_n] \equiv \{(a_1, \ldots, a_m) \in D^m \mid \langle D, \delta, r_1, \ldots, r_n \rangle \models \phi(a_1, \ldots, a_m)\}
\]

In other words the query is a mapping from unrestricted relations \( r_1, \ldots, r_n \) to an unrestricted relation \( \rho \). An important requirement is that \( \rho \) must be representable by a generalised relation \( r \) of arity \( m \).

The following is an example of a relational calculus query with constraints.
Example 2.4 (Example 1.7, [KKR95]). Let $\Phi$ be the class of dense linear order constraints. If $R_1$ is a predicate symbol or arity 2, then the following is a query:

$$\phi(x_1, x_2) \equiv R_1(x_1, x_2) \lor (\exists y)(R_1(x_1, y) \land R_1(y, x_2) \land (x_1 \leq x_2) \land (x_2 \leq y))$$

2.4.1 Some Useful Complexity Results for Linear Constraint Databases

The literature of constraint databases contains many interesting papers that concentrate on the complexity analysis of evaluation of relational queries over relational databases with various classes of constraints (e.g., polynomial, linear temporal, etc.).

Here we only remind the reader of the bounds known for the classes of constraints introduced above.

Proposition 2.8. [KKR95] The data and combined complexity of relational calculus with different classes of constraints is shown in the following table.

<table>
<thead>
<tr>
<th></th>
<th>Polynomial</th>
<th>Linear</th>
<th>Dense Order</th>
<th>Equality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational Calculus</td>
<td>NC</td>
<td>NC</td>
<td>LOGSPACE</td>
<td>LOGSPACE</td>
</tr>
</tbody>
</table>

Tighter bounds are possible in special cases. In particular, [GS95] shows that the evaluation of dense-order constraint databases defined using only integers in the constraints is in $\text{AC}^0$. The same holds for a restricted form of linear constraints, namely $k$-bounded linear constraints [GST94a, GST94b]. A $k$-bounded linear constraint is a relation that is finitely representable by a quantifier free formula in DNF, such that, in each atomic formula occurring in it, there are at most $k$ occurrences of the addition symbol, and all constants are integers.

2.5 Polyhedral Computation

When we study the complexity of constraints that are linear inequalities (as in the constraint database model introduced earlier in Section 2.4 but also the stRDF model of Chapter 3) we often need to use algorithms that come from the areas of Operations Research (more specifically linear programming) and Computational Geometry. In this section we give a short introduction to the relevant notions and discuss some well-known results that will be useful when we study the complexity of stSPARQL query evaluation in Chapter 4. For background on the theory of polytopes see [Zie95], and for linear programming focusing on computational complexity, see [Sch86b].

Let $p$ be a conjunction of equations or inequalities $h_1 \wedge h_2 \wedge \cdots \wedge h_m$. Let $f$ be an arbitrary function on the variables of $p$. Define $LP_{\text{min}}(f \mid P)$ (respectively, $LP_{\text{max}}(f \mid P)$) to be the minimization (respectively, maximization) linear program where $P$, the set of all $h_i$ present in $p$, define the feasible region and $f$ is an optimization function on the variables of $P$ (i.e., the variables of $p$).

The feasible region $P$ of a linear program is a convex polyhedron. When $P$ is bounded it is a convex polytope. For our introduction here we restrict ourselves to polytopes; similar things hold for polyhedra. Let $k$ denotes the dimension of the space where the polytope lives (i.e., the ambient space). Note that the dimension of the polytope could be lower than $k$. For example, consider a set of constraints on two variables defining a segment. A representation of a polytope by a set of linear constraints is called a half-space representation ($H$-representation). An equivalent representation can be given as the convex hull of finitely many points in which case it is called a vertex representation ($V$-representation). Let $m$ denote the minimal number of inequalities needed to represent $P$, that is excluding the redundant inequalities. Also let $n$ denote the number of its
vertices. Vertices are the points of \( P \) which cannot be represented as convex combinations of two other points in \( P \). A convex polytope is called a \( k \)-polytope, where \( k \) is the number of variables appearing in \( P \). A nice text for these concepts is [Zie95].

The transformation of \( H \) to \( V \) representation is known as the vertex enumeration problem and the converse is known as the facet enumeration problem or convex hull problem. Note that in order to describe the convex hull of \( n \) points, one may need a number of inequalities that is exponential in \( n \). This is given by MucMullen’s upper bound theorem [McM71] which states that a \( k \)-polytope with \( n \) vertices has \( \Theta(n^{k/2}) \) number of faces. This bound is attained by the cyclic polytopes [Zie95]. Similarly, the polyhedron defined by \( m \) inequalities may have a number of vertices that is exponential in \( m \). To see this consider the \( m \)-cube which is defined by \( 2m \) inequalities and has \( 2^m \) vertices, i.e., \( n \) equals \( 2^m \). Thus, the computational complexity of the vertex and facet enumeration problems is at least exponential only by reporting the output. Therefore, it is natural to consider algorithms whose running time is measured not only in the size of the input but also in the size of the output. Such algorithms are called output-sensitive algorithms. In general we say an algorithm is polynomial if it runs in time bounded by a polynomial in \( k, n, m \). This polynomiality coincides with the usual polynomiality when the output size is polynomially bounded by the size of input.

The vertex enumeration problem is called nondegenerate if there is no point \( x \in \mathbb{R} \) which satisfies \( d + 1 \) given inequalities with equality, and degenerate otherwise. The facet enumeration problem is called nondegenerate if there is no \( d + 1 \) given points which are on a common hyperplane, and degenerate otherwise. Under the nondegeneracy assumption there are polynomial algorithms for the convex hull problem [CC53, Dye83, CK70]. A more recent algorithm is presented in [AF91] with time complexity \( O(nkm) \) and space complexity \( O(nk) \). The novelty of this algorithm is that its space complexity is polynomial in the input size only. In the general case, there is no known polynomial algorithm. Some software packages for vertex and facet enumeration can be found in [Fuk98, Avi98]. To conclude with complexity results recall that the linear programming problem is in PTIME [Kar84, Kha79]. Additionally, checking a convex polytope for unboundedness can also be solved in polynomial time [Kar84, Kha79, Tod02, KS06].

### 2.6 Conclusions

In this chapter we presented some background material which is necessary for the reader to be familiar with, so that the technical material that follows in Chapters 3, 4, and 5 is appreciated. The next chapter introduces the data model stRDF and the query language stSPARQL.
3. The Model \textit{stRDF} and the Query Language \textit{stSPARQL}

In this chapter we give the definition of the stRDF data model and the stSPARQL query language from [KK10]. Our presentation follows [KK10] verbatim. The intention of this section is to acquaint the reader with all the details of stRDF and stSPARQL so that the complexity analysis of Chapter 4 can be appreciated.

3.1 Data Model

To develop stRDF, we follow closely the ideas of constraint databases [KK90, Rev02] and especially the work on CSQL [KRSS98]. First, we define the formulae that we allow as constraints. Then, we develop stRDF in two steps. The first step is to define the model sRDF which extends RDF with the ability to represent spatial data. Then, we extend sRDF to stRDF so that thematic and spatial data with a temporal dimension can be represented.

3.1.1 Linear Constraints

Constraints will be expressed in the first-order language $\mathcal{L} = \{\leq, +\} \cup \mathcal{Q}$ over the structure $\mathcal{Q} = (\mathbb{Q}, \leq, +, (q)_{q \in \mathbb{Q}})$ of the linearly ordered, dense and unbounded set of the rational numbers, denoted by $\mathbb{Q}$, with rational constants and addition. The atomic formulae of this language are linear equations and inequalities of the form: $\sum_{i=1}^{p} a_i x_i \Theta a_0$, where $\Theta$ is a predicate among =, or $\leq$, the $x_i$'s denote variables and the $a_i$'s are integer constants. Note that rational constants can always be avoided in linear equations and inequalities. The multiplication symbol is used as an abbreviation i.e., $a_i x_i$ stands for $x_i + \cdots + x_i (a_i$ times).

We now define semi-linear subsets of $\mathbb{Q}^k$, where $k$ is a positive integer.

**Definition 3.1.** Let $S$ be a subset of $\mathbb{Q}^k$. $S$ is called semi-linear if there is a quantifier-free formula $\phi(x_1, \ldots, x_k)$ of $\mathcal{L}$ where $x_1, \ldots, x_k$ are variables such that $(a_1, \ldots, a_k) \in S$ iff $\phi(a_1, \ldots, a_k)$ is true in the structure $\mathcal{Q}$.

We will use $\emptyset$ to denote the empty subset of $\mathbb{Q}^k$ represented by any inconsistent formula of $\mathcal{L}$.

3.1.2 The sRDF Data Model

We now define sRDF. As in theoretical treatments of RDF [PAC09], we assume the existence of pairwise-disjoint countably infinite sets $I$, $B$ and $L$ that contain IRIs, blank nodes and literals respectively. In sRDF, we also assume the existence of an infinite sequence of sets $C_1, C_2, \ldots$ that are pairwise-disjoint with $I$, $B$ and $L$. The elements of each $C_k, k = 1, 2, \ldots$ are the quantifier-free formulae of the first-order language $\mathcal{L}$ with $k$ free variables. We denote with $C$ the infinite union $C_1 \cup C_2 \cup \cdots$.

**Definition 3.2.** An sRDF triple is an element of the set $(I \cup B) \times I \times (I \cup B \cup L \cup C)$. If $(s, p, o)$ is an sRDF triple, $s$ will be called the subject, $p$ the predicate and $o$ the object of the triple. An sRDF graph is a set of sRDF triples.
In the above definition, the standard RDF notion of a triple is extended, so that the object of a triple can be a quantifier-free formula with linear constraints. According to Definition 3.1, such a quantifier-free formula with \(k\) free variables is a finite representation of a (possibly infinite) semi-linear subset of \(\mathbb{Q}^k\). Semi-linear subsets of \(\mathbb{Q}^k\) can capture a great variety of spatial geometries, e.g., points, lines, line segments, polygons, \(k\)-dimensional unions of convex polygons possibly with holes, thus they give us a lot of expressive power. However, they cannot be used to represent other geometries that need higher-degree polynomials e.g., circles.

**Example 3.1.** The following are sRDF triples:

\[
\text{ex:s1 rdf:type, ex:Sensor .}\ 
\text{ex:s1 ex:has_location "x=10 and y=20"^^strdf:SemiLinearPointSet}
\]

The above triples define a sensor and its location using a conjunction of linear constraints. The last triple is not a standard RDF triple since its object is an element of set \(C\).

In terms of the W3C specification of RDF, sRDF can be realized as an extension of RDF with a new kind of **typed literals**: quantifier-free formulae with linear constraints. The datatype of these literals is e.g., \(\text{strdf:SemiLinearPointSet}\) (see Example 3.1 above) and can be defined using XML Schema. Alternatively, linear constraints can be expressed in RDF using MathML and serialized as \(\text{rdf:XMLLiterals}\) as in \(\text{PS09}\). \(\text{PS09}\) specifies a syntax and semantics for incorporating linear equations in OWL 2. We now move on to define stRDF.

### 3.1.3 The stRDF Data Model

We will now extend sRDF with time. Database researchers have differentiated among user-defined time, valid time and transaction time. RDF (and therefore sRDF) supports user-defined time since triples are allowed to have as objects literals of the following XML Schema datatypes: \(\text{xsd:dateTime, xsd:time, xsd:date, xsd:gYearMonth, xsd: gYear, xsd:gMonthDay, xsd:gDay, xsd:gMonth}\).

stRDF extends sRDF with the ability to represent the **valid time** of a triple (i.e., the time that the triple was valid in reality) using the approach of Gutierrez et al. \(\text{GHV07}\) where the a fourth component is added to each sRDF triple.

The **time structure** that we assume in stRDF is the set of rational numbers \(\mathbb{Q}\) (i.e., time is assumed to be linear, dense and unbounded). Temporal constraints are expressed by quantifier-free formulae of the language \(\mathcal{L}\) defined earlier, but their syntax is limited to elements of the set \(C_1\). **Atomic temporal constraints** are formulae of \(\mathcal{L}\) of the following form: \(x \sim c\), where \(x\) is a variable, \(c\) is a rational number and \(\sim\) is \(<\), \(\le\), \(\ge\), \(>\), \(=\) or \(\neq\). **Temporal constraints** are Boolean combinations of atomic temporal constraints using a single variable.

The following definition extends the concepts of triple and graph of sRDF so that thematic and spatial data with a temporal dimension can be represented.

**Definition 3.3.** An stRDF **quad** is an sRDF triple \((a, b, c)\) with a fourth component \(\tau\) which is a temporal constraint. For quads, we will use the notation \((a, b, c, \tau)\), where the temporal constraint \(\tau\) defines the set of time points that the fact represented by the triple \((a, b, c)\) is valid in the real world. An stRDF **graph** is a set of sRDF triples and stRDF quads.
3.2 The stSPARQL Query Language: Formal Definition and Semantics

In this section, we give a formal definition of stSPARQL and define its semantics by following an algebraic approach like the one originally pioneered in \[PAG09\]. We only cover the spatial features of stSPARQL in detail and their interactions with existing SPARQL concepts. The temporal features of stSPARQL (quad patterns and temporal filters) can be formalized similarly and are omitted.

Let us recall from Section 3.1.2 the definitions of sets \(I, B, L, C_1, C_2, \ldots\) and \(C\). We define \(ILC = I \cup B \cup L\) and \(T = I \cup B \cup L \cup C \cup \mathbb{R}\). We need to include the set of real numbers \(\mathbb{R}\) in the set \(T\) since as we will see below (Definition 3.8) the application of certain metric functions such as \(\text{AREA}\) etc. can result in real numbers as answers to stSPARQL queries.

We also assume the existence of the following disjoint sets of variables: (i) the set of non-spatial variables \(V_{ns}\), (ii) an infinite sequence \(V_1^1, V_2^1, \ldots\) of sets of variables that will be used to denote elements of the sets \(C_1, C_2, \ldots\) and (iii) the set of real variables \(V_r\). We use \(V_s\) to denote the infinite union \(V_1^1 \cup V_2^1 \cup \ldots\) and \(V\) to denote the union \(V_{ns} \cup V_s \cup V_r\). The set \(V\) is assumed to be disjoint from the set \(T\).

Let us now define a concept of mapping appropriate for stSPARQL by modifying the definition of \[PAG09\]. A mapping \(\mu\) from \(V\) to \(T\) is a partial function \(\mu : V \rightarrow T\) such that \(\mu(x) \in I \cup B \cup L\) if \(x \in V_{ns}\), \(\mu(x) \in C_i\) if \(x \in V_i^1\) for all \(i = 1, 2, \ldots\) and \(\mu(x) \in \mathbb{R}\) if \(x \in V_r\).

Example 3.2. The following is a mapping:

\[
\{?S \rightarrow s_1, \ ?O \rightarrow John, \ ?GEO \rightarrow “x \geq 1 \land y \geq 0 \land y \leq 5”\}
\]

The notions of domain and compatibility of mappings is as in \[PAG09\]. The domain of a mapping \(\mu\), denoted by \(\text{dom}(\mu)\), is the subset of \(V\) where the mapping is defined. Two mappings \(\mu_1\) and \(\mu_2\) are compatible if for all \(x \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)\) we have \(\mu_1(x) = \mu_2(x)\). For two sets of mappings \(\Omega_1\) and \(\Omega_2\), the operations of join, union, difference and left outer-join are also defined exactly as in \[PAG09\].

\[
\begin{align*}
\Omega_1 \times \Omega_2 &= \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ are compatible mappings}\} \\
\Omega_1 \cup \Omega_2 &= \{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\} \\
\Omega_1 \setminus \Omega_2 &= \{\mu \mid \mu \in \Omega_1 \text{ for all } \mu' \in \Omega_2, \mu \text{ and } \mu' \text{ are not compatible}\} \\
\Omega_1 \bowtie \Omega_2 &= (\Omega_1 \times \Omega_2) \cup (\Omega_1 \setminus \Omega_2)
\end{align*}
\]

Using an algebraic syntax for stSPARQL graph patterns which extends the one introduced for SPARQL in \[PAG09\], we now define the result of evaluating a graph pattern over an stRDF graph.

Definition 3.4. Let \(G\) be an stRDF graph over \(T\), \(p\) a triple pattern and \(P_1, P_2\) graph patterns. Evaluating a graph pattern \(P\) over a graph \(G\) is denoted by \([P]_G\) and is defined as follows \[PAG09\]:

\[
\begin{align*}
(i) \ [p]_G &= \{\mu \mid \text{dom}(\mu) = \text{var}(p) \text{ and } \mu(p) \in G\}, \text{ where } \text{var}(p) \text{ is the set of variables occurring in } p. \\
(ii) \ [(P_1 \text{ AND } P_2)]_G &= [P_1]_G \times [P_2]_G \\
(iii) \ [(P_1 \text{ OPT } P_2)]_G &= [P_1]_G \bowtie [P_2]_G \\
(iv) \ [(P_1 \text{ UNION } P_2)]_G &= [P_1]_G \cup [P_2]_G
\end{align*}
\]
The semantics of \textit{FILTER} expressions in \textit{stSPARQL} are defined as in [PAG09] for filters that do not involve spatial predicates. To define the semantics of spatial filters formally, we first need the following definitions.

\textbf{Definition 3.5.} A \textit{k}-ary spatial term is an expression of the following form:

\begin{itemize}
\item[(i)] a quantifier-free formula of \( \mathcal{L} \) from the set \( C_k \) (in quotes).
\item[(ii)] a spatial variable from the set \( V^k_s \).
\item[(iii)] \( t \cap t' \) (intersection), \( t \cup t' \) (union), \( t \setminus t' \) (difference), \( BD(t) \) (boundary), \( MBB(t) \) (minimum bounding box), \( BF(t, a) \) (buffer) where \( t \) and \( t' \) are \( k \)-ary spatial terms and \( a \) is a rational number.
\item[(iv)] the projection \( t[i_1, \ldots, i_k'] \) of a \( k \)-ary spatial term \( t \) where \( i_1, \ldots, i_k' \) are positive integers less than or equal to \( k \).
\end{itemize}

\textbf{Example 3.3.} The following are examples of binary spatial terms:

\begin{itemize}
\item \( \text{"(}x \geq 1 \land x = y \text{)} \lor y = 7\text{"} \)
\item \( \text{?GEO} \cap \text{"(}x \geq 1 \land x = y \text{)} \lor y = 7\text{"} \)
\item \( \text{BD(}\text{?GEO} \cap \text{"(}x \geq 1 \land x = y \text{)} \lor y = 7\text{"}) \)
\item \( \text{"(}x \geq 1 \land x \leq 10 \land y \geq 0 \land x = y\text{)}\text{[1, 2]} \cap \text{"(}z \geq 0 \land z \leq 10\text{)}" \)
\end{itemize}

\textbf{Definition 3.6.} A metric spatial term is an expression of the form \( f(t) \) where \( f \) is one of the metric functions \( VOL \) (volume), \( AREA \) (area or surface), \( LEN \) (length), \( MAX \) (maximal value) or \( MIN \) (minimal value) and \( t \) is a \( k \)-ary spatial term. In the case of \( AREA \) we require \( k \geq 2 \). In the case of \( LEN \), \( MAX \) and \( MIN \), we require \( k = 1 \).

\textbf{Remark.} For metric spatial terms, we make the following assumptions. An input \( t \) to a metric function \( f \) is always proper and satisfiable, i.e., if \( f \) is \( MAX \), \( MIN \), or \( LEN \), then the dimension of \( t \) is 1 and \( t \) is such that its maximum (respectively minimum, and length) value of \( t \) is always defined. For an unsatisfiable input let us take \( t \) to be \( t_1 \cap t_2 \) with \( MAX(t_1) < MIN(t_2) \) (or \( MAX(t_2) < MIN(t_1) \)). We assume that the maximum (respectively minimum) value for an atomic formula of the form \( x < c \) is the number \( c \), since the set of rationals is dense. Last, a proper input for the \( AREA \) metric function is one with dimension 2.

\textbf{Example 3.4.} The following are examples of metric spatial terms:

\begin{itemize}
\item \( AREA(\text{"(}x \geq 1 \land x \leq 10 \land y \geq 0 \land x = y\text{)}\text{[1]}\) \)
\item \( MIN(\text{"(}x \geq 1 \land x \leq 10 \land y \geq 0 \land x = y\text{)}\text{[1]}\) \)
\end{itemize}

Note that Definition 3.6 is not recursive like Definition 3.5 i.e., \( f \) can only be applied once to a \( k \)-ary spatial term. The result of the application of \( f \) is a real number and the definition of mapping has already catered for this possibility.

\textbf{Definition 3.7.} A spatial term is a \( k \)-ary spatial term or a metric spatial term.

We will be interested in the value of a \( k \)-ary spatial term \( t \) for a given mapping \( \mu \) such that the variables of \( t \) are all among the spatial variables of \( \mu \). This is captured by the following definition.

\textbf{Definition 3.8.} Let \( t \) be a spatial term. Let \( \mu \) be a mapping such that all the spatial variables of \( t \) are elements of \( \text{dom}(\mu) \). The value of \( t \) for \( \mu \) is denoted by \( \mu(t) \) and is defined as follows:

\begin{itemize}
\item[(i)] If \( t \) is an element of \( C_k \) then \( \mu(t) = t \).
\item[(ii)] If \( t \) is a spatial variable \( x \) then \( \mu(t) = \mu(x) \).
\item[(iii)] If \( t \) is a projection expression of the form \( t'[i_1, \ldots, i_k'] \) then \( \mu(t) \) is a quantifier-free formula \( \phi \) of \( \mathcal{L} \) which is obtained after eliminating from \( \mu(t') \) the variables corresponding to all the other dimensions except \( i_1, \ldots, i_k' \).
\end{itemize}
(iv) If \( t \) is the intersection \( t' \cap t'' \) of two \( k \)-ary spatial terms then \( \mu(t) = \mu(t' \cap t'') = \mu(t') \land \mu(t'') \). \[2\]

(v) If \( t \) is the union \( t' \cup t'' \) of two \( k \)-ary spatial terms then \( \mu(t) = \mu(t' \cup t'') = \mu(t') \lor \mu(t'') \).

(vi) If \( t \) is the difference \( t' \setminus t'' \) of two \( k \)-ary spatial terms then \( \mu(t) = \mu(t') \setminus \mu(t'') \).

(vii) If \( t \) is \( \text{MBB}(t') \) where \( t' \) is a \( k \)-ary spatial term, then \( \mu(t) \) is a quantifier-free formula of the language \( \mathcal{L} \) that represents the minimum bounding box of \( \mu(t') \).

(viii) If \( t \) is \( \text{BD}(t') \) where \( t' \) is a \( k \)-ary spatial term, then \( \mu(t) \) is a quantifier-free formula of the language \( \mathcal{L} \) that represents the boundary of \( \mu(t') \).

(ix) If \( t \) is \( \text{BF}(t',a) \) where \( t' \) is a \( k \)-ary spatial term and \( a \) is a rational number, then \( \mu(t) \) is a quantifier-free formula of the language \( \mathcal{L} \) that represents the buffer of \( \mu(t') \) within distance \( a \). The buffer of \( t \) contains \( t \) and a zone of width \( a \) around \( t \).

(x) If \( t \) is \( \text{VOL}(t'), \text{AREA}(t') \) or \( \text{LEN}(t') \) where \( t' \) is a \( k \)-ary spatial term, then \( \mu(t) \) is a real number that represents the volume, surface (or area) or length of \( \mu(t') \).

(xi) If \( t \) is \( \text{MIN}(t'), \text{MAX}(t') \) where \( t' \) is a unary spatial term, then \( \mu(t) \) is a real number that represents the minimum or the maximum value of \( \mu(t') \).

To guarantee closure of stSPARQL, it is important to point out that the value \( \mu(t) \) in the above definition is a well-defined formula of \( \mathcal{L} \) in the cases \( 1 \), \( 2 \) and a real number in the cases of \( 3 \) and \( 4 \). This is easy to see for cases \( 1 \) and \( 2 \). For the case \( t = \text{MBB}(t'), \mu(t) \) is \( \bigwedge_{i=1}^{k} (d_i \leq x_i \land x_i \leq u_i) \) where \( d_i, u_i \) are the minimum and maximum values of \( x_i \) for which the formula \( \mu(t') \) holds in the structure \( Q \). For the case \( t = \text{BD}(t') \), the formula \( \mu(t) \) can be constructed by performing quantifier elimination in the quantified formula defining the boundary given in Proposition 3.1 of [VCG01].

For the case \( t = \text{BF}(t',a) \) and the standard definition of buffer that uses the Euclidean distance, the formula \( \mu(t) \) is not general an element of \( \mathcal{L} \) (e.g., \( \text{BF}(x=0 \wedge y=0^+,1) \) is the unit circle with center \((0,0)\)). There are two alternative non-standard definitions of \( \text{BF} \) that allow us to stay in the realm of linear constraints. In the first case, \( \text{BF} \) can be defined using the Manhattan distance which measures the distance between two points along axes at right angles. For example, in the case of two dimensions, the formula \( \mu(t) \) would now be the formula that remains if we eliminate variables \( x', y' \) from the formula:

\[
\begin{align*}
(\phi(x',y') \land 0 \leq x-x' \leq a \land 0 \leq y-y' \leq a) \lor \\
(\phi(x',y') \land 0 \leq x-x' \leq a \land 0 \leq y-y' \leq a) \lor \\
(\phi(x',y') \land 0 \leq x-x' \leq a \land 0 \leq y-y' \leq a) \lor \\
(\phi(x',y') \land 0 \leq x-x' \leq a \land 0 \leq y-y' \leq a)
\end{align*}
\]

where \( \phi(x', y') \) is the formula \( \mu(t') \). If using Manhattan distance seems like a crude alternative to the standard definition then more detailed alternatives are possible. For example, if \( t \) defines a polygon then \( \text{BF}(t, a) \) is a new polygon that contains \( t \) and the zone of width \( a \) around the polygon (however, “circular” curves are approximated by polylines). Note that the same approach is followed by vector data models e.g. the computational geometry library CGAL. The cases \( 3 \) and \( 4 \) are easy to see as well.

**Definition 3.9.** An atomic spatial condition is an expression in any of the following forms:

(i) a topological spatial condition \( t_1 R t_2 \) where \( t_1 \) and \( t_2 \) are \( k \)-ary spatial terms and \( R \) is one of the topological relationships DISJOINT, TOUCH, EQUALS, INSIDE, COVEREDBY, CONTAINS, COVERS, OVERLAP.

\[3\]

---

2In this and subsequent definitions, we assume that standardization of variables takes place before forming the conjunction, disjunction of formulae etc.

(ii) a linear equation or inequality of $\mathcal{L}$ with metric spatial terms in the place of variables.

It is important to mention here that evaluation of a topological operator is valid only between $k$-ary spatial terms of the same dimension. For objects of different dimension, a solution would be to project the object of the higher dimension to the dimension of the lower and evaluate the topological relation between these objects. For this case, we assume that $k$-ary spatial terms are of the same dimension.

Note that the form of (ii) does not destroy closure of our language since these equations/inequalities allows linear equations or inequalities with terms that evaluate to real numbers and they will only be checked for satisfaction (see Definition 3.11), not used as constraints i.e., as elements of sets $C_k$.

Example 3.5. The following are atomic spatial selection conditions:

- $\text{?GEO1 INSIDE } "x \geq 1 \land x \leq 5 \land y \geq 0 \land y \leq 5"$
- $\text{AREA(?GEO1)} \geq 2 \cdot \text{AREA(?GEO2)}$

Definition 3.10. A spatial condition is a Boolean combination of atomic spatial conditions.

Definition 3.11. A mapping $\mu$ satisfies a spatial condition $R$ (denoted $\mu \models R$) if

(i) $R$ is atomic and the spatial condition that results from substituting every spatial variable $x$ of $R$ with $\mu(x)$ holds for semi-linear sets in $\mathbb{Q}^n$.

(ii) $R$ is ($\neg R_1$), $R_1$ is a spatial condition, and it is not the case that $\mu \models R_1$.

(iii) $R$ is ($R_1 \lor R_2$), $R_1$ and $R_2$ are spatial conditions, and $\mu \models R_1$ or $\mu \models R_2$.

(iv) $R$ is ($R_1 \land R_2$), $R_1$ and $R_2$ are spatial conditions, and $\mu \models R_1$ and $\mu \models R_2$.

The semantics of spatial filters can now be defined as follows.

Definition 3.12. Given an stRDF graph $G$ over $T$, a graph pattern $P$ and a spatial condition $R$, we have: $[P \ \text{FILTER} \ R]_G = \{ \mu \in [P]_G \mid \mu \models R \}$.

Now we can define the semantics of the SELECT clause of an stSPARQL expression where variables (spatial or non-spatial) are selected and new spatial terms are computed. To capture the peculiarities of the SELECT clause of stSPARQL, we first need the following definitions.

Definition 3.13. Let $t$ be a spatial (resp. metric spatial) term and $z$ a spatial (resp. real) variable that does not appear in $t$. Then, $t \ AS z$ is called an extended spatial term with target variable $z$.

Example 3.14. A projection specification is a set consisting of non-spatial variables, spatial variables and extended spatial terms such that all the target variables of the extended spatial terms are different from each other and different from each spatial variable.

Definition 3.15. Let $\mu$ be a mapping and $W$ be a projection specification with spatial and non-spatial variables $x_1, \ldots, x_l$ and extended spatial terms $t_1 \ AS z_1, \ldots, t_m \ AS z_m$. Then, $\pi_W(\mu)$ is a new mapping such that

1. $\text{dom}(\pi_W(\mu)) = \{x_1, \ldots, x_l, z_1, \ldots, z_m\}$.
2. $\pi_W(\mu)(x_i) = \mu(x_i)$ for $1 \leq i \leq l$ and $\pi_W(\mu)(z_j) = \mu(t_j)$ for $1 \leq j \leq m$.

Example 3.7. Let $\mu$ be the mapping $\{\text{?S} \rightarrow \text{s1}, \text{?O} \rightarrow \text{John}, \text{?GEO} \rightarrow \text{"x} \geq 1 \land x \leq 5 \land y \geq 0 \land y \leq 5\"\}$ and $W$ the projection specification $\{\text{?O}, \text{BD(?GEO)} \cap \text{“x = 1”}, \text{AS ?L}\}$. Then $\pi_W(\mu)$ is the following mapping:

$\{\text{?O} \rightarrow \text{John}, \text{?L} \rightarrow \text{“x = 1 \land y \geq 0 \land y \leq 5”}\}$. 

D2.3 Theoretical results on query processing for RDF/SPARQL with time and space 25
The next definition gives the semantics of an arbitrary stSPARQL query.

**Definition 3.16.** An stSPARQL query is a pair $(W, P)$ where $W$ is a projection specification and $P$ is a graph pattern. The *answer* to an stSPARQL query $(W, P)$ over a graph $G$ is the set of mappings $\{\pi_W(\mu) \mid \mu \in [P]\}$.

**Example 3.8.** Let $G$ be the following stRDF graph:

$\{(s_1, \text{geom}, "x \geq 1 \land y \geq 0 \land y \leq 5"), (s_1, \text{owner}, \text{John})\}$

Consider now the query $(W, P)$, such that

$W = \{?O, BD(?GEO) \cap \text{"x = 1" AS } ?L\}$ and $P = (?S, \text{owner}, ?O) \text{ AND } (?S, \text{geom}, ?GEO)$

Then, the answer to $(W, P)$ over $G$ is the set that consists of the following mapping:

$\{?O \rightarrow \text{John}, ?L \rightarrow "x = 1 \land y \leq 5 \land y \geq 0"\}$

### 3.3 Conclusions

In this chapter we introduced the stRDF data model and the stSPARQL query language that were presented in [KK10]. We gave a formal definition of stRDF and presented a detailed semantics of stSPARQL using the algebraic approach pioneered for SPARQL in [PAG06]. This chapter is crucial for understanding the next chapter (Chapter 4), where the complexity of evaluating stSPARQL over stRDF is studied.
4. The Complexity of stSPARQL Query Evaluation

In this chapter we study the complexity of query evaluation for the query language stSPARQL. Our results build on the complexity results for SPARQL given in [PAG09], and the complexity results for the decision and quantifier elimination problem of the theory of real addition with order (see Chapter 2).

We also utilize results from the constraint database literature especially [KKR95, VGG01, GL95, GST94c]. To be able to calculate the complexity of stSPARQL query evaluation, we start with the feature that is new in stSPARQL namely spatial conditions appearing in FILTER expressions (see Definition 3.10 in Chapter 3). Once we have precise complexity bounds for the evaluation of spatial conditions (this is done in Section 4.1 below) then the complexity of evaluating an stSPARQL graph pattern is easily computed by relying on the results of [PAG09] for SPARQL (this is done in Section 4.2 below).

4.1 Evaluation of Spatial Conditions

In this section we develop the evaluation procedure for spatial conditions (Eval
spatial
condition in Figure 4.1) and all the procedures that it depends on. Then, Section 4.2 uses this procedure to develop a query evaluation procedure for stSPARQL. The data and combined complexity of these procedures is also studied in detail.

Formula Assumptions and Notation. Let $t$ be a formula of $L$ that represents a semi-linear set $S$. These formulae are the constraints used in the spatial conditions of stSPARQL. We denote by $\text{size}(t)$ the number of bits needed to write $t$ in binary. Likewise, if $s$ is a symbol appearing in $t$, then $\text{size}(s)$ gives its size.

The form of $t$ affects the computational complexity of the evaluation of spatial conditions. Thus, we examine two forms of input formulae $t$: arbitrary form and DNF. Note that when a formula $t$ is in DNF then every disjunction represents a convex polyhedron, which is not necessarily disjoint from the others. If $t$ is a formula in DNF, we denote by $|t|$ the number of convex polyhedra of $t$ (i.e., the number of disjuncts).

Let $p$ be a convex polyhedron. We denote by $|p|$ the number of its faces and we write it as $p = h_1 \land \cdots \land h_{|p|}$. Then, the face of $p$ that corresponds to inequality $h_j$ is denoted by $F_{h_j}(p)$, where $h_j$ is the inequality $h_j$ after replacing the inequality symbol by the symbol “=”.

Removing Redundant Inequalities. Let $p = h_1 \land \cdots \land h_m$ be a formula that represents a convex polyhedron using $k$ variables $x = (x_1, \ldots, x_k)$. For readability purposes, we opted here to group variables $x_1, \ldots, x_k$ and represent them as a vector instead of listing them explicitly. We follow the same approach where we feel that this facilitates understanding. Some inequalities may be redundant for $p$, i.e., they may not change the feasible space and thus they should be removed. The following linear program checks if an inequality $h_i \equiv c^T x \leq c_{k+1}$ is redundant: $LP_{max}\{c^T x \mid p \land c^T x \leq c_{k+1} + 1\}$. The inequality $h_i$ is redundant if and only if the optimal value of the previous linear program is less than or equal to $c_{k+1}$. If a redundant inequality is found, we
can remove it from \( p \). If we do this repeatedly, we end up with a formula without redundancies. Thus, we can assume that every convex polyhedron does not contain redundant inequalities (in the case it contains, the redundant inequalities can be eliminated as above with a polynomial time cost).

Interior, Closure, and Boundary. The interior, closure, and boundary of a semi-linear set represented by a formula \( t \) with variables \( x = (x_1, \ldots, x_k) \), denoted as \( t^o \), \( \overline{t} \), and \( \partial t \) respectively, can be expressed by the following formulae of \( L \) [VGG01, Proposition 3.1]:

\[
\begin{align*}
    t^o &\equiv (\exists \epsilon)(\epsilon > 0 \land (\forall y)(\epsilon < y - x < \epsilon \rightarrow t(x/y))) \\
    \overline{t} &\equiv (\forall \epsilon)(\epsilon > 0 \rightarrow (\exists y)(t(x/y) \land \epsilon < y - x < \epsilon)) \\
    \partial t &\equiv t \land \neg t^o
\end{align*}
\]

where \( t(x/y) \) means the substitution of every occurrence of variable \( x_i \) by variable \( y_i \) for \( 1 \leq i \leq k \). Each relation involving vectors is interpreted coordinate wise. For example, the relation \( -\epsilon < y - x < \epsilon \) is interpreted as \( \bigwedge_{i=1}^{k} (-\epsilon < y_i - x_i < \epsilon) \), which can be further simplified to be a valid formula in \( L \) as: \( \bigwedge_{i=1}^{k} ((-\epsilon < y_i - x_i) \land (y_i - x_i < \epsilon)) \). If \( x \) represents the free variables in that last formula, the above condition expresses that there is an open cube centered on \( x \) with side \( 2\epsilon \). Referring to the interior formula, the point is that, whenever there exists such an open cube contained in \( t \), there is also an open sphere centered on \( x \) and contained in \( t \). This is because the open sphere with radius \( \epsilon \) and center \( x \) is entirely contained in the open cube with side \( 2\epsilon \) centered on \( x \).

In what follows, we study the complexity of evaluating atomic spatial conditions according to the semantics of stSPARQL, in both input formula situations (DNF and general). Following Figure 4.1 we start from the evaluation of spatial terms that are then used to construct atomic spatial conditions. Recall that complexities are usually given in terms of a time or space complexity class and measured with respect to the size of the input (formulae of \( L \) in our case). See Chapter 2 for more details.

Figure 4.1: Syntactic structure and evaluation of spatial conditions
4.1.1 Evaluation of $k$-ary Spatial Terms

We will define a recursive evaluation procedure for $k$-ary spatial terms, $Eval_{k\text{-}ary\text{-}spatial}(t)$, that given as input the $k$-ary spatial term $t$, constructs a quantifier-free formula of $L$ from the set $C_k$ with variables $x_1, \ldots, x_k$. The evaluation procedure is structured according to the recursive definition of the $k$-ary spatial term (see Definition 3.5 of Chapter 3).

We also study the case of DNF formulae. In particular, we want to check if an operator used to construct a term maintains the DNF property. That is, if an operator is applied on some DNF formulae and the resulting formula can be obtained in DNF without the exponential explosion. In this case, the output quantifier-free formula of $L$ should represent a union of convex polyhedra.

In what follows we give the recursive definition of $Eval_{k\text{-}ary\text{-}spatial}(t)$ by considering the possible operators used to construct a $k$-ary spatial term.

**Union.** To evaluate the spatial union of the $k$-ary spatial terms $t_1$ and $t_2$ (i.e., $t_1 \cup t_2$), we take the disjunction of the formulae, i.e., $t = t_1 \lor t_2$. If $t_1$ and $t_2$ are in DNF, then $t$ is also in DNF. In both cases (whether or not the input formulae are in DNF), the evaluation can be done in PTIME since the resulting formula has size $\size(t_1) + \size(^\lor) + \size(t_2)$. Note that while we need just one operation to compute the union of two $k$-ary spatial terms, we have also to write down the output formula, which is dependent on the size of the input formulae.

**Intersection.** To evaluate the spatial intersection of the $k$-ary spatial terms $t_1$ and $t_2$ (i.e., $t_1 \cap t_2$) we take the conjunction of the formulae, i.e., $t = t_1 \land t_2$. If $t_1$ and $t_2$ are in DNF, then $t$ is also in DNF. If we are not interested in preserving the DNF property then evaluation can be done in PTIME like before ($\size(t) = \size(t_1) + \size(^\land) + \size(t_2)$).

**Difference.** To evaluate the spatial difference of the $k$-ary spatial terms $t_1$ and $t_2$ (i.e., $t_1 \setminus t_2$) we take the conjunction of $t_1$ and the negation of $t_2$ (according to the semantics of stSPARQL), i.e., $t = t_1 \land \neg t_2$. If $t_1$ and $t_2$ are in DNF, then formula $\neg t_2$ will be in CNF and in order to transform it into DNF, we shall obtain a formula with size exponential in the size of the input. Without maintaining the DNF property of $t$, the difference can be evaluated in PTIME; the size of the resulting formula is $\size(t_1) + \size(^\land) + \size(^\neg) + \size(t_2)$.

**Minimum Bounding Box.** To evaluate the minimum bounding box of the $k$-ary spatial term $t$ (i.e., $MBB(t)$), we distinguish between the following two cases regarding the form of the input formula:

- **Formula $t$ is in DNF form.** In this case, first we compute the minimum bounding boxes for every convex polyhedron and then compute the bounding box that confines all these. To compute the minimum bounding boxes, we solve the linear programs $LP_{\min}(x_i \mid p_j)$ and $LP_{\max}(x_i \mid p_j)$ for each convex polyhedron $p_j$, $1 \leq j \leq |t|$, and for each dimension $x_i$, $1 \leq i \leq k$. The solutions of the linear programs are the minimum ($u_i$) and maximum ($u_i$) values for each dimension $x_i$, which corresponds to the lower left and upper right vertices of each bounding box. Consequently,

$$MBB(p_j) \equiv \bigwedge_{i=1}^{k} (l_i \leq x_i \land x_i \leq u_i)$$
To compute the minimum bounding box of \( t \) what is needed is to compute the lower and upper vertices for every dimension \( x_i \). This can be done by the following formula:

\[
MBB(t) \equiv \bigwedge_{i=1}^{k} \left( \min_{j \in [1,|t_i|]} \{LP_{min}(x_i \mid p_j)\} \leq x_i \wedge x_i \leq \max_{j \in [1,|t_i|]} \{LP_{max}(x_i \mid p_j)\} \right)
\]

Clearly, all these computations can be done in PTIME, since the linear programming problem is in PTIME. Note that if the input is unbounded (i.e., it is the union of a set of polyhedra some of which is unbounded or it defines a hypersurface), then MBB is not defined. Testing for unboundedness can be solved in PTIME [Kar84, Kha79, KS06, Tod02].

- **Formula \( t \) is in arbitrary form.** In this case, there are two options to evaluate \( MBB(t) \):

  1. Transform \( t \) to DNF and then follow the aforementioned solution. Clearly, DNF transformation is in EXPTIME, and so is \( MBB(t) \).

  2. Find MBB without transforming \( t \) into DNF. To the best of our knowledge there are no known algorithms published especially for this in the literature. From a logical perspective, we could formulate this problem as finding the minimum and maximum values for each variable of \( t \) that are also solutions of \( t \). Then, the MBB could be computed by applying quantifier elimination on the resulting formula. For example, if \( t(x_1, x_2) \) is a 2-ary spatial term in free variables \( x_1, x_2 \), then the minimum values for \( x_1, x_2 \) (denoted as \( x_{1m}, x_{2m} \)) would be given by applying quantifier elimination to the following formula:

\[
(\forall x_1 x_2) \left[ t(x_1, x_2) \rightarrow x_{1m} \leq x_1 \wedge x_{2m} \leq x_2 \wedge t(x_{1m}, x_{2m}) \right]
\]

Note that the previous formula can be written in the form \( \neg (\exists x_1 x_2) F(x_1, x_2) \), i.e., in the general case it is a formula with a fixed number of alternations. Quantifier elimination for this kind of formulae is in EXPTIME [Wei88, Ren92] (see Table 2.1). For fixed dimension, the complexity of quantifier elimination and thus for evaluation of MBB is in NC [KKR93] (see Table 2.1).

**Boundary.** To evaluate the boundary of the \( k \)-ary spatial term \( t \) (i.e., \( BD(t) \)), we perform quantifier elimination to the quantified formula (4.1c) from page 28 defining the boundary. For the general case, this algorithm has time and space complexity \( O(2^{cn}) \), where \( c \) is a constant and \( n \) the size of \( t \) [Wei88] (see Table 2.1). In the case that the dimension is fixed, then evaluation of \( BD(t) \) is in NC [KKR95] (see Table 2.1).

Note here, that we have not made any distinction regarding the form of the input formula. This is because, even if \( t \) was in DNF, the computation of the boundary for each disjunct (i.e., convex polyhedron) does not solve the overall problem, since \( t \) may be a rather complex spatial object possibly with self-intersections, separations, overlapping segments, etc. We are unaware of any method that computes the boundary of such an object by considering the H-representation.

**Buffer.** To evaluate the buffer of the \( k \)-ary spatial term \( t(x_1, \ldots, x_k) \) with radius \( a \) (i.e., \( BF(t, a) \)) we work as follows: the standard definition of buffer operator that uses the Euclidean distance introduces non-linear constraints that are needed for the representation of the result. We study here the two alternatives proposed in Section 3.2 to solve this problem. The first one is a simple redefine of buffer using Manhattan distance which results in a new formula:

\[
\phi(x_1', \ldots, x_k') \equiv t \wedge \bigvee_{i=1}^{k} \left( (0 \leq x_i' - x_i \leq a \wedge 0 \leq x_i - x_i' \leq a) \right)
\]

If we eliminate variables \( x_1, \ldots, x_k \) from \( \phi \), we get a formula with variables \( x_1', \ldots, x_k' \) that represents \( BF(t, a) \). Applying quantifier elimination to \( \phi \) is in EXPTIME [Wei88], since it is equivalent.
to $\exists x_1 \ldots x_k \phi$ (see Table 2.1). In the special case of having $t$ in DNF, we observe that $\phi$ can be written in DNF only in $k|t|$ steps. Even in that case, quantifier elimination stays in EXPTIME. Like the previous cases, when we fix the dimension of $t$, quantifier elimination can be done in NC [KKR95] (see Table 2.1).

To sum up, the buffer operator can be evaluated in EXPTIME in the general case and in NC for fixed dimension.

**Projection.** To evaluate the projection of the $k$-ary spatial term $t$ on $k_1 < k$ dimensions (i.e., $t[x_1, \ldots, x_{k_1}]$) it suffices to see that projection can be expressed as the problem of eliminating all quantifiers $x_{k_1+1}, \ldots, x_k$ except $x_1, \ldots, x_{k_1}$ from the following formula:

$$(\exists x_{k_1+1} \ldots x_k) t(x_1, \ldots, x_{k_1}, x_{k_1+1}, \ldots, x_k)$$

As we have seen earlier, quantifier elimination for such formulae is in EXPTIME [Wei88] and for fixed dimension in NC [KKR95] (see Table 2.1).

If we distinguish the case of having $t$ in DNF, then we do not obtain any better complexity.

### 4.1.2 Evaluation of Metric Spatial Terms.

In the following, we study the complexity of evaluating a metric spatial term $Eval_{\text{metric}}$. Metric spatial terms are functions that take as input a $k$-ary spatial term $t$ and computes a real number. These functions of stSPARQL are $VOL(t)$, $LEN(t)$, $MAX(t)$, $MIN(t)$, and $AREA(t)$.

**Volume.** In general, the problem of computing the volume of a convex polytope, given in either $H$-representation or $V$-representation, is known to be #PTIME-complete [DF88]. There are algorithms with complexity $O(m^k)$ [BFF00], where $m$ is the number of the inequalities of the $H$-representation. For fixed dimension these algorithms are polynomial. In our case, to evaluate the volume of the $k$-ary spatial term $t$ (i.e., $VOL(t)$) we distinguish between two cases regarding the form of the input formula.

- **Formula $t$ is in DNF form.** The volume computation differs when $t$ contains overlapping convex polytopes. In this case, according to the inclusion-exclusion principle, volume computation requires $\sum_{i=1}^{n} \binom{n}{i} = O(2^n)$ computations, in order to measure the volume of all intersecting polytopes only once. Thus, the computation is in EXPTIME. In the case of non-overlapping convex polytopes, volume computation amounts to the sum of the volume of each polytope. Thus, the computation is in #PTIME.

- **Formula $t$ is in arbitrary form.** The naive method is to transform $t$ to DNF and then follow the aforementioned procedure. Then the complexity is determined by the DNF transformation, which is in EXPTIME.

**Length, Max, and Min.** The 1-ary spatial term $t$ that is input to a term $LEN(t)$, $MIN(t)$ or $MAX(t)$ is a formula of $\mathcal{L}$ with only one variable. The algorithm for computing the length, maximum, and minimum of such a spatial term first eliminates all negations in the input formula resulting in a new formula composed of atomic formulae of the form $x \theta c$, where $c$ is a constant and $\theta$ an operator from $<,>,=,\neq,\leq,\geq$. We can eliminate each negation ($\neg$) from $t$ by replacing $\neg(t_1 \land t_2)$, $\neg(t_1 \lor t_2)$ by $\neg t_1 \lor \neg t_2$, $\neg t_1 \land \neg t_2$, respectively. In the case of negated atomic formulae, $\neg t_1$, the negation can be eliminated by replacing $<,>,= \text{ in } x$ by $\geq,\leq,\neq$ respectively. Then, it is
Table 4.1: Semantics of topological operators

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>∅ ∩ ∅</th>
<th>∅ ∩ ∅</th>
<th>∅ ∩ ∅</th>
<th>∅ ∩ ∅</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISJOINT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TOUCH</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EQUALS</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INSIDE</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>COVEREDBY</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>CONTAINS</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>COVERS</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>OVERLAP</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

easy to determine the length, maximum, or minimum value of any logical combination of formulae of the previous form. MAX (and similarly MIN) can be evaluated using the following recursive procedure.

Algorithm 1: EvalMAX(t)

// negation has already been eliminated from t
switch t do
  case t is an atomic formula of \( L \):
    t is a formula \( x \theta c \), where \( c \) is a constant and \( \theta \) an operator \(<,>,=,\neq,\leq,\geq\). The maximum value is \( c, \infty, c, \infty, c, \infty \) respectively
  case t is \( t_1 \land t_2 \):
    EvalMAX(t) is \( \min\{\text{EvalMAX}(t_1), \text{EvalMAX}(t_2)\} \)
  case t is \( t_1 \lor t_2 \):
    EvalMAX(t) is \( \max\{\text{EvalMAX}(t_1), \text{EvalMAX}(t_2)\} \)

It is worth noting that, according to Remark 3.2, EvalMAX(t) does not consider the case of evaluating the maximum value for a formula that is unsatisfiable. For example, consider the case of \( t \) being \( t_1 \land t_2 \) and that EvalMAX(t1) < EvalMIN(t2) or EvalMAX(t2) < EvalMIN(t1). In such cases, the maximum value is not defined and we opt not to include it in the evaluation procedure. Besides, if we add such a test, the complexity of evaluation does not change. Another point to stress is that by convention the maximum value for an atomic formula of the form \( x < c \) is the number \( c \), since the set of rationals is dense.

Both the elimination of the negation and the evaluation of the recursive procedure can be evaluated in PTIME, just by parsing the input formula \( t \). The length of \( t \), Len(t), can be computed simply by subtracting the minimum value of \( t \) from its maximum value, i.e., \( \text{MAX}(t) - \text{MIN}(t) \). Thus, it can also be evaluated in PTIME. Consequently, the evaluation procedure can be done in PTIME.

Area. To evaluate the area or surface of the \( k \)-ary spatial term \( t \) with \( k = 2 \) (i.e., AREA(t)) amounts to evaluating VOL(t). The complexity results in this case are the same with VOL. We are unaware of any better results in the literature for the case of AREA.

4.1.3 Evaluation of Topological Conditions

In the following, we describe the evaluation procedure for topological conditions, namely Evaltopology(R) that given a topological condition \( R \), it returns whether it holds or not. According to the definition of stSPARQL (see Definition 3.9 of Chapter 3), a topological condition \( R \) is of the form \( t_1 \text{OP} t_2 \), where \( t_1, t_2 \) are \( k \)-ary spatial terms. Operator OP can be one of the following topological relations: DISJOINT, TOUCH, EQUALS, INSIDE, COVEREDBY, CONTAINS, COVERS, and
OVERLAP, as defined in GIS and spatial databases [EF91, Ege89, Ege91]. This set of topological relations has also been studied in knowledge representation in the context of the RCC-8 theory of topology [RCC92b]. It is worth noting that the topological relations have been studied by these two different lines of research, each one defining them using a slightly different model and semantics. For example, GIS and spatial databases often assume a point-based model for the represented spatial objects. Then, the semantics of a topological relation is defined according to all the possible ways the boundary and interior of two regions may intersect (see for example [EF91]). On the other hand, the area of spatial knowledge representation typically assumes a primitive connected relation, $C(x, y)$, denoting the fact that two regions, $x$ and $y$, share one or more points. Then, all the above topological relations are defined using first-order logic formulae with predicate $C$ or predicates that have been defined using $C$ as their basis [RCC92b].

Our work assumes the point-based model employing the 4-intersection method that defines the semantics for the topological relations presented earlier according to the possible ways the boundary and interior of two objects intersect [Ege89]. The semantics of these relations are given in Table 4.1. For each operator, the table specifies whether the interior and boundary of two objects may (value 1) or may not (value 0) intersect so that the operator holds. Further, notice that there are 16 possible ways of intersection, while only 8 are mentioned here. This is because the other 7 cannot happen in a spatial context. Figure 4.2 shows typical examples in the plane.

Before proceeding to the evaluation of topological relations, we summarize the structure of the spatial objects that the 4-intersection method assumes. In particular, the 4-intersection method imposes the following restrictions [Ege89]:

- All objects are defined by regular closed subsets of $\mathbb{R}^n$ (i.e., subsets $A$ of $\mathbb{R}^n$ with the property that the closure of their interior is equal to $A$).
- All objects are represented as simplicial complexes.
- All objects are cells, i.e., their boundary is not empty.
- All objects are not self-intersecting.
- All objects are connected.
- All objects are of genus 0, i.e., they have no holes.
- The underlying space must be topological and open.
Since semi-linear sets are a more general class of subsets of $\mathbb{R}^n$ than the regular closed sets, it is interesting to also consider a more refined topological operator model such as the 9-intersection method (9IM) [ECF94], the dimension extended method (DEM) [CFvO93], or their combination known as DE-9IM [CDF95]. We leave this for future work.

To evaluate a topological condition $R$, we construct a formula of $L$ that is true if and only if $R$ is true. This can be done by using the formulae (4.1a) and (4.1c) from page 28, which express the interior and boundary respectively, together with the semantics of the topological operators as defined in Table 4.1. For example, let us consider the spatial condition $t_1$ EQUALS $t_2$. By inspecting Table 4.1, $t_1$ would be equal to $t_2$ only if their boundaries and interiors intersect and the boundary of each one does not intersect with the interior of the other. This condition is reflected in the following formula:

$$
\phi \equiv (\exists x)(\partial t_1(x) \land \partial t_2(x)) \land (\exists y)(t_1^e(y) \land t_2^e(y)) \land \neg(\exists z)(t_1^c(z) \land \partial t_2(z)) \land \neg(\exists w)(\partial t_1(w) \land t_2^e(w))
$$

Notice, that $\phi$ contains quantifiers which have been introduced by the formulae representing the boundary (4.1c), closure (4.1b), and interior (4.1a) of $t_1$ and $t_2$, as well as the existential ones shown in $\phi$. The condition $t_1$ EQUALS $t_2$ is true, whenever $\phi$ is valid. Validity can be checked by applying quantifier elimination and checking whether the resulting quantifier-free formulae is true. Note here the usage of the top-level existential quantifiers; they are used to quantify the free variables of the formulae for the closure, interior, and boundary. Thus, $\phi$ is a general sentence of $Th(R_+)$ and we can check validity using a decision procedure. Thus, the complexity of checking validity is in EXPSPACE [FR75, Wei88] (see Table 2.1). When the dimension is fixed, validity can be checked in NC [BOKR84] (see Table 2.1).

### 4.1.4 Evaluation of Spatial Conditions

Let us now give our evaluation procedure for an atomic spatial condition $t$, namely $Eval_{atomic}(t)$. According to the definition of stSPARQL, such conditions can be of two kinds. The first one is a linear inequality (or equality) $c_1t_1 + \cdots + c_mt_m \leq c_{m+1}$, where $t_1, \ldots, t_m$ are metric spatial terms without spatial variables and $c_1, \ldots, c_{m+1}$ are constants. Evaluating,

$$
Eval_{atomic}(c_1t_1 + \cdots + c_mt_m \leq c_{m+1})
$$

is the same as checking whether the inequality

$$
c_1Eval_{metric}(t_1) + \cdots + c_mEval_{metric}(t_m) \leq c_{m+1}
$$

is true.

In the second case, condition $t$ is a topological condition and can be evaluated as we have shown in Section 4.1.3 above.

Given the results we presented in Section 4.1.2 for metric spatial terms, the complexity of evaluating an atomic spatial condition is 2-EXPTIME for general dimension and in EXPTIME for fixed.

A spatial condition is a Boolean combination of atomic spatial conditions and can be evaluated by the recursive procedure Algorithm 2 given below.

Given the above recursive procedure and the complexity results of atomic spatial conditions, the complexity of evaluating a spatial condition is 2-EXPTIME for general dimension and in EXPTIME for fixed.

In conclusion, the complexity results that were presented earlier are summarized in Table 4.2.
Algorithm 2: $\text{Eval}_{\text{spatial\ condition}}$ ($t$: spatial condition)

\[
\text{switch } t \text{ do}
\]
\[
\text{case } t \text{ is } \text{atomic:}\quad \text{return } \text{Eval}_{\text{atomic}}(t);
\]
\[
\text{case } t \text{ is } \neg t_1:\quad \text{return } \neg \text{Eval}_{\text{spatial\ condition}}(t_1);
\]
\[
\text{case } t \text{ is } t_1 \lor t_2:\quad \text{return } \text{Eval}_{\text{spatial\ condition}}(t_1) \lor \text{Eval}_{\text{spatial\ condition}}(t_2);
\]
\[
\text{case } t \text{ is } t_1 \land t_2:\quad \text{return } \text{Eval}_{\text{spatial\ condition}}(t_1) \land \text{Eval}_{\text{spatial\ condition}}(t_2);
\]
\[
\text{return } \text{false;}
\]

Table 4.2: Complexity results. (*) denotes non overlapping polytopes

<table>
<thead>
<tr>
<th>$k$-ARY</th>
<th>non DNF general dim</th>
<th>non DNF fixed dim</th>
<th>DNF general dim</th>
<th>DNF fixed dim</th>
</tr>
</thead>
<tbody>
<tr>
<td>MBB</td>
<td>EXPTIME</td>
<td>NC</td>
<td>PTIME</td>
<td>NC</td>
</tr>
<tr>
<td>BD</td>
<td>2-EXPTIME</td>
<td>NC</td>
<td>2-EXPTIME</td>
<td>NC</td>
</tr>
<tr>
<td>BF</td>
<td>EXPTIME</td>
<td>NC</td>
<td>EXPTIME</td>
<td>NC</td>
</tr>
<tr>
<td>PROJ</td>
<td>EXPTIME</td>
<td>NC</td>
<td>EXPTIME</td>
<td>NC</td>
</tr>
<tr>
<td>VOL</td>
<td>EXPTIME</td>
<td>EXPTIME</td>
<td>#PTIME (*)</td>
<td>EXPTIME</td>
</tr>
<tr>
<td>AREA</td>
<td>undef</td>
<td>EXPTIME</td>
<td>undef</td>
<td>EXPTIME</td>
</tr>
<tr>
<td>LEN, MIN</td>
<td>undef</td>
<td>PTIME</td>
<td>undef</td>
<td>PTIME</td>
</tr>
</tbody>
</table>

4.2 A Decision Procedure for stSPARQL Query Evaluation

To study the complexity of query evaluation for stSPARQL we present a decision procedure that solves the following decision problem (called Evaluation):

\[
\text{INPUT} \quad : \quad \text{An stRDF graph } G, \text{ a graph pattern } P, \text{ and a mapping } \mu.
\]
\[
\text{QUESTION} \quad : \quad \text{Does } \mu \in [P]_G?
\]
Algorithm 3: Eval ($\mu$: mapping, $P$: graph pattern, $G$: stRDF graph)

```
switch $P$ do
  case $P$ is a triple pattern $t$:
    if $\text{dom}(\mu) = \text{var}(t)$ and $\mu(t) \in G$ then return $true$;
    return $false$;
  case $P$ is a pattern of the form ($P_1$ FILTER $R$):
    if Eval($\mu$, $P_1$, $G$) = $true$ and Eval_condition($\mu$, $R$) = $true$ then return $true$;
    return $false$;
  case $P$ is a pattern of the form ($P_1$ UNION $P_2$):
    if Eval($\mu$, $P_1$, $G$) = $true$ or Eval($\mu$, $P_2$, $G$) = $true$ then return $true$;
    return $false$;
  case $P$ is a pattern of the form ($P_1$ AND $P_2$):
    foreach pair of mappings $\mu_1 \in \text{pos}(P_1, G)$ and $\mu_2 \in \text{pos}(P_2, G)$ do
      if Eval($\mu_1$, $P_1$, $G$) = $true$ and Eval($\mu_2$, $P_2$, $G$) = $true$ and $\mu = \mu_1 \cup \mu_2$ then return $true$;
    return $false$;
  case $P$ is a pattern of the form ($P_1$ OPT $P_2$):
    if Eval($\mu$, ($P_1$ AND $P_2$), $G$) = $true$ then return $true$;
    if Eval($\mu$, $P_1$, $G$) = $true$ then
      foreach mapping $\mu \in \text{pos}(P_2, G)$ do
        if Eval($\mu$, $P_2$, $G$) = $true$ and $\mu$ is compatible with $\mu$ then
          return false;
      return true;
    return false;
```

Algorithm 3 presents the recursive procedure Eval($\mu$, $P$, $G$) that decides VALUATION and is based on the corresponding procedure for SPARQL given in [PAG09]. The only difference is in the evaluation of FILTER where additionally to the built-in conditions we have to evaluate spatial conditions. All these conditions are evaluated using the recursive procedure of Algorithm 4.

Algorithm 4: Eval_condition ($\mu$: mapping, $R$: condition)

```
switch $R$ do
  case $R$ is a spatial condition:
    return Eval_spatial_condition($\mu(R)$);
  case $R$ is built-in SPARQL condition:
    return $\mu|_{\text{var}(R)} \models R$;
  case $R$ is $\neg t_1$:
    return $\neg$Eval_condition($t_1$);
  case $R$ is $t_1 \lor t_2$:
    return Eval_condition($t_1$) $\lor$ Eval_condition($t_2$);
  case $R$ is $t_1 \land t_2$:
    return Eval_condition($t_1$) $\land$ Eval_condition($t_2$);
  return false;
```

In Algorithm 4, the evaluation of a spatial condition $R$ given a mapping $\mu$ is performed by calling the recursive procedure Eval_spatial_condition($R$) we defined in Section 4.1.4. Mapping $\mu$ binds the spatial variables to a specific spatial term, thus we can use the evaluation procedures given in the previous section to check whether a spatial condition is satisfied.

Using the decision procedure Eval (Algorithm 3), we can now prove the following proposition for the complexity of the evaluation of filter graph patterns in stSPARQL.

**Theorem 4.1.** Evaluation is in 2-EXPTIME. If $P$ does not contain the boundary (BD) operator, Evaluation is in EXPSPACE. Additionally, if $P$ does not contain topological relations as well, Evaluation is in EXPTIME.
Proof. Note that Evaluation for SPARQL is in PSPACE [PAG09, Theorem 3.6]. Evaluation of spatial filters is in 2-EXPTIME (see Table 4.2), and thus, Evaluation for stSPARQL is in 2-EXPTIME. Evaluation of spatial filters without the BD operator is in EXPSPACE (see Table 4.2). Discarding also any topological relation, evaluation of spatial filters is in EXPTIME.

Let Evaluation\((P)\) be the Evaluation problem when considering \(P\) to be fixed (data complexity).

**Theorem 4.2.** Evaluation\((P)\) has the same upper bounds given above for Evaluation. The same applies to the restricted decision problems in which \(P\) does not contain the boundary operator or any of the topological relations.

Proof. Note that Evaluation\((P)\) for SPARQL is in LOGSPACE [PAG09, Theorem 3.6]. Still, by fixing \(P\), the dimension of any linear formula appearing in \(P\) is considered to be fixed, but this is not the case for the linear formulae appearing in the database. Thus, evaluation of spatial filters is still in 2-EXPTIME (see Table 4.2). The same applies to the rest of the decision problems considered in which \(P\) does not contain the boundary operator or any of the topological relations.

Comparing our result about the data complexity of stSPARQL with the data complexity of relational calculus with polynomial or linear constraints as given in [KKR95, GST94c], we see that the evaluation of stSPARQL is intractable (2-EXPTIME in general and EXPTIME in some subcases), while in the standard linear constraint database setting, evaluation of relational calculus with linear constraints has low complexity (NC). This difference emerges due to the following technical subtlety in the two models: for the case of standard constraint databases, by considering the size of the query to be fixed, both the schema of the query and the dimension of the constraints appearing in the query are also fixed. As a result, the schema of the database is considered fixed, and thus the dimension of the database is also fixed. This is because of the fact that in standard constraint databases, constraints, and thus variables they contain, are part of a relation’s schema. What is left is the number of tuples that defines the database size, which also dominates all other sizes. In our model, this is not the case, since linear constraints appear as constants from a set of constraints \(C\) (see the definition of the sRDF data model and stSPARQL in Sections 3.1.2 and 3.2).

Given the above discussion, a natural case to consider is to fix the dimension of the constraints appearing both in the query and the database, i.e., to consider a subset of the set \(C\). This could be for example the set \(C_1 = C_1 \cup C_2\) containing constraints with at most two variables. This is a very reasonable scenario for a GIS database like the ones we encounter in TELEIOS use cases. This case corresponds also to the version of stSPARQL that uses the OGC standard Well Known Text (WKT) instead of linear constraints as in the original language defined in [KK10]. This version was defined in Section 4 of Deliverable 2.1 [KKN+11] and was called SPARQL++ to distinguish it from the original stSPARQL language of [KK10]. This is the version of stSPARQL we use in TELEIOS and it has already partially implemented in the system Strabon available by NKUA. stSPARQL++ is very close syntactically and semantically to GeoSPARQL, a recent OGC proposal for geospatial extension of SPARQL.

For this case, we distinguish between the following decision problems: Evaluation\(_p\) and Evaluation\(_p\)(\(P\)).

**Theorem 4.3.** Evaluation\(_p\) is in EXPTIME. If \(P\) does not contain the volume \((VOL)\) and area \((AREA)\) operators, Evaluation\(_p\) is in PSPACE.

Proof. Note that Evaluation for SPARQL is in PSPACE [PAG09, Theorem 3.6]. Evaluation of spatial filters with fixed dimension is in EXPTIME (see Table 4.2), and thus, Evaluation\(_p\) for stSPARQL is in EXPTIME. Evaluation of spatial filters with fixed dimension and without considering the VOL and AREA operators is in PTIME (see Table 4.2), and thus, Evaluation\(_p\) is in PSPACE, that is, evaluation in this case is dominated by the evaluation of the graph patterns.

D2.3 Theoretical results on query processing for RDF/SPARQL with time and space 37
Theorem 4.4. \textit{Evaluation}_F(P) is in \textsc{EXPTIME}. If \( P \) does not contain the volume (VOL) and area (AREA) operators, \textit{Evaluation}_F(P) is in \textsc{PTIME} for every graph pattern expression \( P \).

Proof. Note that \textit{Evaluation}(P) for SPARQL is in \textsc{LOGSPACE} [PAC09, Theorem 3.6]. Evaluation of spatial filters with fixed dimension is in \textsc{EXPTIME} (see Table 4.2), and thus, \textit{Evaluation}_F(P) for stSPARQL is in \textsc{EXPTIME}. Evaluation of spatial filters without the VOL and AREA operators is in \textsc{PTIME} for fixed dimension (see Table 4.2), in which case \textit{Evaluation}_F(P) is in \textsc{PTIME}.

If we compare the above results with the fact that SPARQL queries can be evaluated with \textsc{PSPACE} combined complexity and \textsc{LOGSPACE} data complexity [PAC06, PAC09], we see that, in the general case, we have an increase both in combined and data complexity which is due to the presence of geospatial objects (more precisely: semi-linear point sets) of arbitrary dimension in stRDF graphs, and the availability of a rich set of spatial operators in stSPARQL. Evaluating stSPARQL queries over stRDF graphs containing these geospatial objects is much harder than standard SPARQL query evaluation over RDF graphs. However, if we fix the dimension of the geospatial objects we have in the database and leave out the most expensive operators from the query, then we obtain complexity upper bounds that are similar to the ones for SPARQL.

4.3 Conclusions

In this chapter, we studied the complexity of query evaluation for stSPARQL queries over the stRDF data model. We showed that the combined complexity of stSPARQL is 2-\textsc{EXPTIME}, but it can be lowered to \textsc{EXPSPACE} when the boundary (BD) operator is not included and to \textsc{EXPTIME} when the topological relations are also not included. As for the data complexity, we showed that it is the same with the combined complexity.

An interesting case that is considered is the study of the combined and data complexity of evaluating stSPARQL when the dimension of the linear sets appearing in the query and the database is considered to be fixed. Then, the combined complexity is shown to be \textsc{EXPTIME} due to the presence of volume (VOL) and area (AREA) operators; when these are not included in the query the complexity goes down to \textsc{PSPACE}. Correspondingly, the data complexity remains \textsc{EXPTIME} when considering the full language, but it goes down to \textsc{PTIME}, when the volume and area operators are not included in the query.
5. The stRDF\textsuperscript{i} Data Model: Motivation, Semantics, and Complexity of Query Evaluation

We now turn to the model stRDF\textsuperscript{i} that was first defined in Deliverable 2.1 of TELEIOS. This chapter introduces stRDF\textsuperscript{i} again for the convenience of the reader and “ties any loose ends” in our previous introduction to it. In this sense, this deliverable contains a more definitive version of this model.

This chapter also presents a series of results for the semantics of stRDF\textsuperscript{i} and the evaluation of stSPARQL queries over stRDF\textsuperscript{i} databases. These theoretical results go beyond the material in Deliverable 2.1 that consisted only of relevant definitions and examples (these definitions and examples are repeated here for completeness).

The organization of this chapter is as follows. In Section 5.1 we give the motivation for the stRDF\textsuperscript{i} data model. In Section 5.2 we give a formal definition of stRDF\textsuperscript{i} and then, in Section 5.3 we give its semantics. Section 5.4 defines formally how stSPARQL queries are evaluated on stRDF\textsuperscript{i} databases. Section 5.5 presents our new results concerning the semantics of stRDF\textsuperscript{i} and stSPARQL query evaluation in this context. Finally, Section 5.6 studies the computational complexity of evaluating stSPARQL queries over stRDF\textsuperscript{i} databases.

In this chapter we will often need the following notation.

**Notation 5.1.** Let \( G \) be a set of stRDF graphs and \( q \) an stSPARQL query. The expression \( q(G) \) is defined as the element-wise application of \( q \) to \( G \), that is, \( q(G) = \bigcup_{G_i \in G} q(G_i) \). Likewise, the expression \( \bigcap G \) is defined as \( \bigcap_{G_i \in G} G_i \).

### 5.1 Motivation

Before proceeding to the formal definition of the stRDF\textsuperscript{i} model, we introduce a simple example that will help us explain its features and motivate our approach towards extending the model stRDF with the ability to represent and query indefinite qualitative spatial information. The new model is named stRDF\textsuperscript{i} where “i” stands for “indefinite”.

#### 5.1.1 Indefinite Information in the NOA Use Case

Let us assume that we want to design an stRDF database for an organization such as NOA in TELEIOS. This database will be used by the NOA personnel to keep track of fires taking place all over Greece. The representation of such a database in stRDF will be a list of stRDF triples.

**Example 5.1.** The following is an stRDF database:

```
noa:hotspot1 rdf:type noa:Hotspot .
noa:fire1 rdf:type noa:Fire .
noa:hotspot1 noa:correspondsTo noa:fire1 .
noa:fire1 noa:occurredIn noa:region1 .
noa:region1 strdf:hasGeometry "x=24.82 \& y=35.31"^^strdf:SemiLinearPointSet .
```
The stRDF database of Example 5.1 represents *definite* information; it states that there is a hotspot (noa:hotspot1) and that the corresponding fire (noa:fire1) takes place at a point with geographic coordinates (24.82, 35.31). As we have explained in Deliverable 2.1 [KKN+11], the MSG SEVIRI instrument has medium resolution, therefore each image pixel representing a hotspot in the FMM-1 product corresponds to a 3km by 3km rectangle in geographic space. Accordingly, the FMM-1 product represents hotspots as points in geographic space using the center of the corresponding rectangle. This has led us to write databases and queries in the NOA use case accordingly so that the “3km by 3km” resolution is taken into account (see Query 2 of Section 2.2.4 of Deliverable 2.1).

In this case, another useful representation of the real world situation that corresponds to a hotspot would be to state that there is a geographic region with unknown exact coordinates where a fire is taking place, and that region is included in a known 3km by 3km rectangle. This is captured by the database of Example 5.2 below that introduces the hotspot, the fire corresponding to it, and the region corresponding to the fire. This region (_region1) is a new kind of literal, called an *unknown literal*, which is asserted to be inside the rectangle, formed by the points (24.81, 35.30) and (24.84, 35.33) on Q^2. By convention, identifiers for unknown literals in stRDF always start with an underscore.

**Example 5.2.** The following is an stRDF database:

```
noa:hotspot1 rdf:type noa:Hotspot .
noa:fire1 rdf:type noa:Fire .
noa:hotspot1 noa:correspondsTo noa:fire1 .
noa:fire1 noa:occurredIn _region1 .
_region1 NTPP "x ≥ 24.81 ∧ x ≤ 24.84 ∧ y ≥ 35.30 ∧ y ≤ 35.33" .
```

Unknown literals are like existentially quantified variables in first-order logic or Skolem constants, so one might wonder whether we could have represented them using blank nodes. Given the many interpretations of blank nodes in the relevant RDF literature and related W3C standards [PS08, Hay04, MAHP11], we chose to add a new concept to stRDF that can only be used in the object position of a triple and give it the precise semantics that capture our intentions. If the concept of unknown literal ends up finding good use in RDF practice then an appropriate standard body can choose how to introduce it into the RDF standard in an appropriate way.

stRDF databases like the one of Example 5.2 consist of two parts: a graph (i.e., a set of triples) and a *constraint store*. Constraint stores can in general be formulae of some constraint language. In this example the constraint store is a conjunction of TRCL-constraints, which have been defined in Section 2.2.4. Constraints in TRCL can be used to express qualitative and quantitative spatial properties of regions in Q^2. In the example of Example 5.2, NTPP is the “non-tangential-proper-part” relation of RCC-8 [CCR93].

stRDF databases are syntactic devices for the representation of indefinite spatial information. To be able to understand such databases the notion of possible world [Kri63] is needed. A set of possible worlds can capture all the different states the world can be given the indefinite information in a database. These issues have been studied in detail in the literature of incomplete information in relational databases [IL84, Gra91] and we will take our methodology from these works.

An stRDF database is semantically equivalent to a set of possible worlds which capture all the possible ways the domain of application could have been according to our indefinite spatial information. Each possible world can be represented by an stRDF graph. One can find all the possible worlds represented by an stRDF database as follows:

1. This rectangle is considered to be “3km by 3km” wide.
2. For example one can imagine extending the XML qualified name notation to cover for unknown literals (e.g., ex:_region1). Similarly, unknown literals can have a datatype associated with them (e.g., ex:_region1”^^strdf:SemiLinearPointSet) using the spatial literal datatype of [KK10]. In this report we do not deal with such issues of syntax or standardization at all.
• Find an assignment to the unknown literals which satisfies each constraint of the constraint store.

• Substitute these values for the unknown literals in the RDF database.

For example, the stRDF database of Example 5.1 represents one of the possible worlds corresponding to the stRDF\textsuperscript{i} database of Example 5.2.

**Example 5.3.** Let us now consider the query “Find all fires that have occurred in a region which is a non-tangential proper part of the box defined by the points (24.82, 35.31) and (24.83, 35.32)\textsuperscript{4}”. In the algebraic version of stSPARQL we presented in Section 3.2 of Chapter 3, this query can be expressed as follows:

\[
( \{ ?F \},
\ ( ?F, \text{type}, \text{Fire}) \ \text{AND} \ ( ?F, \text{occurredIn}, ?R) \ \text{AND}
\ \text{FILTER} \ ( ?R \ \text{NTPP} \ "x \geq 24.82 \land x \leq 24.83 \land y \geq 35.31 \land y \leq 35.32") )
\]

What is the answer to this query? If we examine the database of Example 5.2, we can see that the answer should be *conditional* \cite{IL84}. For every object that qualifies as an answer, the query answering procedure should also provide a *condition* characterizing this possible world subset. We cannot say for sure whether fire1 satisfies the requirements of the query because the information in the database is indefinite (the exact geometry of region1 is not known). Fire fire1 qualifies only in the possible worlds where region1 is a non-tangential proper part of the box mentioned in the query. Therefore, the answer to this query could be represented as in Table 5.1 following the ideas of conditional tables from \cite{IL84}.

<table>
<thead>
<tr>
<th>?F</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>noa:fire1</td>
<td>_region1 NTPP &quot;x ≥ 24.82 \land x ≤ 24.83 \land y ≥ 35.31 \land y ≤ 35.32&quot;</td>
</tr>
</tbody>
</table>

In some cases the user might know that the information in the database is indefinite. Thus, she might wish to find all triples that *certainly* satisfy some qualification (this is the well-known notion of certain answers in the incomplete databases literature \cite{Gra09}). Let us consider the previous query again. If we rephrase it to “Find fires that have *certainly* occurred in a region which is a non-tangential proper part of the box defined by the points (24.52, 35.11) and (24.93, 35.52)\textsuperscript{4}”, then fire1 satisfies the query unconditionally\textsuperscript{4}.

In incomplete information scenarios it is often the case that *possibility queries* are also of interest. In other words, a user might wish to find all triples that *possibly* satisfy some qualification. We notice that such queries are of interest only in scenarios that assume the Closed World Assumption. In stRDF\textsuperscript{i}, which is an extension of RDF and therefore follows the Open World Assumption, possibility queries can trivially be answered so we omit them from our discussion.

### 5.2 The stRDF\textsuperscript{i} Data Model

As in theoretical treatments of RDF \cite{PAG06}, we assume the existence of pairwise-disjoint countably infinite sets I, B, and L that contain IRIs, blank nodes, and literals respectively. We also assume the existence of a set C of *spatial* literals. We will not consider arbitrary semi-linear
sets as spatial literals as in Chapters 3 and 4 and paper [KK10]. Instead, $C$ is assumed to be the set of constants of a first-order constraint language $\mathcal{L}$. In our examples we will assume that $\mathcal{L}$ is $\text{TRCL}$, so $C$ is the set of all boxes in $\mathbb{Q}^2$ written in the normal form of Section 2.2 (e.g., "$x \geq 0 \land x \leq 1 \land y \geq 0 \land y \leq 1$).

$\mathcal{L}$ is assumed to be a first-order language with a distinguished equality predicate, denoted by $\text{EQ}$ in the following, that has the standard semantics. In addition, the class of $\mathcal{L}$-constraints is assumed to be weakly closed under negation (see Definition 2.2 in Section 2.2). All the languages of Section 2.2 earlier have these two properties.

In stRDF we assume the existence of one more countably infinite set disjoint from all the above: the set $U$ of unknown spatial literals. By convention, the identifiers of unknown spatial literals will start with an underscore, e.g., $_\text{region5}$. The set of stRDF terms will be denoted by $T$ and is the union $I \cup B \cup L \cup C \cup U$.

We now define the basic concepts of stRDF: triples, constraint stores, graphs, and databases. Triples in stRDF are as in stRDF but now unknown spatial literals are also allowed in the object position. Constraint stores are simply Boolean combinations of atomic constraints of $\mathcal{L}$. The combination of a graph and a constraint store is called database.

**Definition 5.1.** An stRDF triple is an element of the set $(I \cup B) \times I \times (I \cup B \cup L \cup C \cup U)$. If $(s, p, o)$ is an stRDF triple, $s$ will be called the subject, $p$ the predicate and $o$ the object of the triple.

**Definition 5.2.** A constraint store is a Boolean combination of atomic $\mathcal{L}$-constraints. We assume that constraint stores are always satisfiable.

**Example 5.4.** The following are constraint stores:

\[ \phi_1 = \text{._region1 NTPP "} x \geq 24.81 \land x \leq 24.84 \land y \geq 35.30 \land y \leq 35.33\text{"} \]

\[ \phi_2 = (\text{._region1 NTPP "} x \geq 24.81 \land x \leq 24.84 \land y \geq 35.30 \land y \leq 35.33\text{"} \land \\
\text{._region1 NTPP "} x \geq 24.815 \land x \leq 24.83 \land y \geq 35.305 \land y \leq 35.32\text{"}) \lor \\
\text{._region1 EQ "} x \geq 24.82 \land x \leq 24.83 \land y \geq 35.31 \land y \leq 35.32\text{"} \]

**Definition 5.3.** An stRDF graph is a set of stRDF triples. An stRDF database $D$ is a pair $D = (G, \phi)$ where $G$ is an stRDF graph and $\phi$ is a constraint store.

**Example 5.5.** The following pair is an stRDF database (namespaces are omitted for brevity):

\[
\{ (\text{hotspot1, type, Hotspot}), \\
(\text{fire1, type, Fire}), \\
(\text{hotspot1, correspondsTo, fire1}), \\
(\text{fire1, occuredIn, _region1})
\}, \\
\text{._region1 NTPP "} x \geq 24.81 \land x \leq 24.84 \land y \geq 35.30 \land y \leq 35.33\text{"}
\]

**Example 5.6.** The following pair is an stRDF database (namespaces are omitted for brevity):

\[
\{ (\text{hotspot1, type, Hotspot}), \\
(\text{fire1, type, Fire}), \\
(\text{hotspot1, correspondsTo, fire1}), \\
(\text{fire1, occuredIn, _region1}) \\
(\text{fire2, occuredIn, _region1}) \\
(\text{fire2, occuredIn, _region1})
\}
\]

\[
text{\text{._region1 NTPP "} x \geq 24.81 \land x \leq 24.84 \land y \geq 35.30 \land y \leq 35.33\text{"}}
\]
5.3 Semantics of the stRDF\textsuperscript{i} Data Model

Let us now give semantics to the concepts defined above.

An stRDF\textsuperscript{i} database $D = (G, \phi)$ represents a set of possible stRDF graphs each one corresponding to a possible state of the real world. This set of possible graphs captures completely the semantics of an stRDF\textsuperscript{i} database. The constraint store $\phi$ determines the number of possible stRDF graphs represented by $D$; there is one stRDF graph for each solution of $\phi$ obtained by considering the unknown literals of $\phi$ as variables and solving the constraints $\phi$.

The following example illustrates our discussion:

**Example 5.7.** Let $D = (G, \phi)$ be the stRDF\textsuperscript{i} database given in Example 5.5. The database $D$ mentions a hotspot, which is located in a region, which is inside but does not intersect with the boundary of the box defined by the points $(24.81, 35.30)$ and $(24.84, 35.33)$. The same knowledge can be represented by an (infinite) set of possible stRDF graphs, one for each region inside the box defined by the points $(24.81, 35.30)$ and $(24.84, 35.33)$. Two of these graphs are:

$$G_1 = \{ \text{hotspot1, type, Hotspot),}$$
$$\text{(fire1, type, Fire),}$$
$$\text{(hotspot1, correspondsTo, fire1),}$$
$$\text{(fire1, occuredIn, } "x \geq 24.815 \land x \leq 24.83 \land y \geq 35.305 \land y \leq 35.32") \}$$

$$G_2 = \{ \text{hotspot1, type, Hotspot),}$$
$$\text{(fire1, type, Fire),}$$
$$\text{(hotspot1, correspondsTo, fire1),}$$
$$\text{(fire1, occuredIn, } "x \geq 24.82 \land x \leq 24.83 \land y \geq 35.31 \land y \leq 35.32") \}$$

In order to be able to go from stRDF\textsuperscript{i} databases to the equivalent set of possible stRDF graphs, the notion of valuation is needed. Informally, a valuation maps an unknown literal to a specific constant from $C$. The set of valuations that satisfy the constraint store of an stRDF\textsuperscript{i} database can help us to define the set of possible stRDF graphs that correspond to it. This set of graphs is denoted by $\text{Rep}$ following the notation of [IL84].

**Definition 5.4.** A valuation $v$ is a function from $U$ to $C$ assigning to each unknown literal from $U$ a constant from $C$. We denote by $v(G)$ the application of a valuation $v$ to graph $G$. $v(G)$ is obtained from $G$ by replacing all unknown literals $J$ of $G$ by $v(J)$ and leaving all other terms the same. Likewise, if $\phi$ is a constraint store, we denote by $v(\phi)$ the application of a valuation $v$ to $\phi$. The expression $v(\phi)$ is obtained from $\phi$ by replacing all unknown literals $J$ of $\phi$ by $v(J)$.

**Definition 5.5.** Let $D = (G, \phi)$ be an stRDF\textsuperscript{i} database. The set of stRDF graphs represented by $D$ is the following:

$$\text{Rep}(D) = \{ H \mid \text{there exists a valuation } v \text{ such that } M_L \models v(\phi) \text{ and } H \supseteq v(G) \}$$
It is important to notice that the definition of $Rep$ uses the containment relation instead of equality. The reason for this is to reflect the OWA that the RDF model makes; by using the containment relation, $Rep(D)$ includes all the possible interpretations of $v(G)$, i.e., all graphs $H$ that contain at least the triples of $v(G)$.

**Example 5.8.** Going back to Example [5.7] $Rep(D)$ is infinite and contains all super graphs of all graphs formed by each solution of $\phi$ for variable $\_\text{region}_1$. $G_1$ and $G_2$ are two of these graphs.

Given that now we have given the semantics of an stRDF database as a set of possible stRDF graphs, what is an appropriate semantic definition for the answer to a certainty query? This is captured by the following definition of certain answer.

**Definition 5.6.** Let $q$ be a query and $S$ a set of stRDF graphs. The certain answer to $q$ over $S$ is the set

$$\bigcap_{G \in S} q(G).$$

**Example 5.9.** Let us consider the following query over the database of Example [5.5] “Find all fires that have occurred in a region which is a non-tangential proper part of the box defined by the points (24.80, 35.29) and (24.85, 35.34)”. The certain answer for this query is the following set of mappings:

$$\{\{\_F \rightarrow \text{fire1}\}\}$$

### 5.4 Evaluating stSPARQL on stRDF\textsuperscript{i} Databases

Let us now discuss how to evaluate stSPARQL queries on stRDF\textsuperscript{i} databases. Due to the presence of unknown literals, query evaluation now becomes more complicated and is similar to query evaluation for conditional tables [IL84, Gra91]. Since the language used to express constraints in stRDF is tightly connected to the kind of queries that can be answered precisely, we will consider a subset of stSPARQL. The exact details will be given later in this section. We will use the algebraic syntax of stSPARQL presented in Section [3.2] and [KK10].

We assume the existence of the following disjoint sets of variables: (i) the set of non-spatial variables $V_{ns}$ that will be used to denote IRIs, blank nodes, or RDF literals, and (ii) the set of spatial variables $V_s$ that will be used to denote spatial literals from the sets $C$ (known) or $U$ (unknown). We use $V$ to denote the set of all variables $V_{ns} \cup V_s$. Set $V$ is assumed to be disjoint from the set of terms $T$ we defined in Section [5.2].

We first define the concept of $e$-mapping (“e” from the word existential) which extends the concept of mapping of [KK10] with the ability to have an unknown literal as value of a spatial variable.

**Definition 5.7.** An $e$-mapping $\nu$ is a partial function $\nu : V \rightarrow T$ such that $\nu(x) \in I \cup B \cup L$ if $x \in V_{ns}$ and $\nu(x) \in C \cup U$ if $x \in V_s$.

**Example 5.10.** The following are $e$-mappings:

$$\{ \_F \rightarrow \text{fire1}, \_S \rightarrow \_x \geq 1 \land \_x \leq 2 \land \_y \geq 1 \land \_y \leq 2\}$$

$$\{ \_F \rightarrow \text{fire1}, \_S \rightarrow \_\text{region}_1\}$$

The notions of domain and restriction of an $e$-mapping are defined as for mappings in the obvious way [PAG06, PAG09] (we also use the same notation for them).

The notion of valuation is extended to $e$-mappings as follows: Let $\nu$ be an $e$-mapping and $v$ a valuation. We denote by $v(\nu)$ the application of valuation $v$ to $e$-mapping $\nu$. $v(\nu)$ is obtained from $\nu$ by replacing all unknown literals $v(x)$ of $\nu$ by $v(\nu(x))$ and leaving all other terms the same.
We now extend the concept of $e$-mapping and define conditional mappings. Conditional mappings are equipped with a local constraint which constraints the unknown literals that appear in the $e$-mapping.

**Definition 5.8.** A conditional mapping $\mu$ is a pair $(\nu, \theta)$ where $\nu$ is an $e$-mapping and $\theta$ is a conjunction of atomic $L$-constraints.

**Example 5.11.** The following are conditional mappings:

$$\mu_1 = (\{ ?F \rightarrow \text{fire1}, ?S \rightarrow \text{"x}$\geq 1 \wedge x \leq 2 \wedge y \geq 1 \wedge y \leq 2\}, \text{true})$$

$$\mu_2 = (\{ ?F \rightarrow \text{fire1}, ?S \rightarrow \text{\_region1} \}, \begin{array}{c}
\text{\_region1 NTPP “x}$\geq 0 \wedge x \leq 10 \wedge y \geq 0 \wedge y \leq 10^n\end{array})$$

$$\mu_3 = (\{ ?F \rightarrow \text{fire1}, ?S \rightarrow \text{\_region1} \}, \begin{array}{c}
\text{\_region1 NTPP \_region2} \wedge \\
\text{\_region2 DC “x}$\geq 0 \wedge x \leq 1 \wedge y \geq 0 \wedge y \leq 1^n\end{array})$$

$$\mu_4 = (\{ ?F \rightarrow \text{fire1}, ?S \rightarrow \text{\_region1} \}, \text{true})$$

Notice that conditional mappings with constraint true, such as $\mu_4$ above, are logically equivalent to $e$-mappings.

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

**Definition 5.9.** The domain of a conditional mapping $\mu = (\nu, \theta)$, denoted by $\text{dom}(\mu)$, is the domain of $\nu$, i.e., the subset of $V$ where the partial function $\nu$ is defined.

**Definition 5.10.** Let $\mu = (\nu, \theta)$ be a conditional mapping with domain $S$ and $W \subseteq S$. The restriction of the mapping $\mu$ to $W$ denoted by $\mu|_{\text{\_W}}$ is the mapping $(\nu|_{\text{\_W}}, \theta)$ where $\nu|_{\text{\_W}}$ is the restriction of mapping $\nu$ to $W$.

**Definition 5.11.** For a triple pattern $p$, we denote by $\text{var}(p)$ the variables appearing in $p$.

We now define what it means for a conditional mapping to be applied to a triple pattern.

**Definition 5.12.** Let $\mu = (\nu, \theta)$ be a conditional mapping. For a triple pattern $p$, we denote by $\mu(p)$ the triple obtained from $p$ by replacing each variable $x \in \text{var}(p) \cap \text{dom}(\mu)$ by $\nu(x)$.

We now introduce the notion of compatible conditional mappings as in [PAG06].

**Definition 5.13.** Two conditional mappings $\mu_1 = (\nu_1, \theta_1)$ and $\mu_2 = (\nu_2, \theta_2)$ are compatible if the mappings $\nu_1$ and $\nu_2$ are compatible, i.e., for all $x \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$, we have $\nu_1(x) = \nu_2(x)$.

**Example 5.12.** Mappings $\mu_1$ and $\mu_2$ from Example 5.11 are not compatible, while mappings $\mu_2$ and $\mu_3$ are.

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

**Definition 5.14.** Two conditional mappings $\mu_1 = (\nu_1, \theta_1)$ and $\mu_2 = (\nu_2, \theta_2)$ are possibly compatible if for all $x \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$, 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 unknown literal from $U$.

**Example 5.13.** Conditional mappings $\mu_1$, $\mu_2$, and $\mu_3$ from Example 5.11 are pairwise possibly compatible.
If two conditional mappings are possibly compatible, then we can define their join as follows.

**Definition 5.15.** 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 $(\nu_3, \theta_3)$ where:

1. $\nu_3(x) = \nu_1(x) = \nu_2(x)$ for each $x \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$ such that $\nu_1(x) = \nu_2(x)$.
2. $\nu_3(x) = \nu_1(x)$ for each $x \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$ such that $\nu_1(x)$ is an unknown literal and $\nu_2(x)$ is a known literal.
3. $\nu_3(x) = \nu_2(x)$ for each $x \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$ such that $\nu_2(x)$ is an unknown literal and $\nu_1(x)$ is a known literal.
4. $\nu_3(x) = \nu_1(x)$ for $x \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$ such that both $\nu_1(x)$ and $\nu_2(x)$ are unknown literals.
5. $\nu_3(x) = \nu_1(x)$ for $x \in \text{dom}(\mu_1) \setminus \text{dom}(\mu_2)$.
6. $\nu_3(x) = \nu_2(x)$ for $x \in \text{dom}(\mu_2) \setminus \text{dom}(\mu_1)$.
7. $\theta_3$ is $\theta_1 \land \theta_2 \land \xi_1 \land \xi_2 \land \xi_3$ where:
   - $\xi_1$ is $\bigwedge_i \nu_i \, \text{EQ} \, \lambda_i$, where the $\nu_i$’s and $\lambda_i$’s are all the pairs of unknown literals $\nu_1(x)$ and $\nu_2(x)$ from Case (iv) above. If there are no such pairs, then $\xi_1$ is true.
   - $\xi_2$ is $\bigwedge_i \nu_i \, \text{EQ} \, \lambda_i$, where the $\nu_i$’s and $\lambda_i$’s are all the pairs of unknown literals $\nu_1(x)$ and known literals $\nu_2(x)$ from Case (ii) above. If there are no such pairs, then $\xi_2$ is true.
   - $\xi_3$ is $\bigwedge_i \nu_i \, \text{EQ} \, \lambda_i$, where the $\nu_i$’s and $\lambda_i$’s are all the pairs of unknown literals $\nu_2(x)$ and known literals $\nu_1(x)$ from Case (iii) above. If there are no such pairs, then $\xi_3$ is true.

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

**Example 5.14.** If $\mu_1$ and $\mu_2$ are the conditional mappings of Example 5.11 then:

$$\mu_1 \bowtie \mu_2 = \{ \{ ?F \rightarrow \text{fire1}, ?S \rightarrow \text{region1} \}, \text{true} \land \text{region1} \, \text{EQ} \, \text{"x} \geq 1 \land x \leq 2 \land y \geq 1 \land y \leq 2\text{"} \land \text{region1} \, \text{NTPP} \, \text{"x} \geq 0 \land x \leq 10 \land y \geq 0 \land y \leq 10\text{"} \}$$

For two sets of conditional mappings $\Omega_1$ and $\Omega_2$, 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 one in [PAG06, PAG09]. The new thing in stRDF is that due to the presence of unknown spatial literals, we have to anticipate the possibility that two mappings from $\Omega_1$ and $\Omega_2$ are compatible. We anticipate this case by adding relevant constraints to the constraint part 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 can now define the result of evaluating a graph pattern over an stRDF graph. The definitions are essentially the same as in [KK10] (only the first case is slightly different since the constraint $\text{true}$ is added to the mapping $\nu$ to construct an RDFS rule that results from the evaluation of a triple pattern).

**Definition 5.16.** Let $D = (G, \phi)$ be an stRDF database. Evaluating a graph pattern $P$ over database $D$ is denoted by $\llbracket P \rrbracket_D$ and is defined recursively as follows:
1. If $P$ is the triple pattern $(s, p, o)$ then:
   - if $o$ is a spatial literal then
     \[
     [P]_D = \{ \mu = (\nu, \text{true}) \mid \text{dom}(\mu) = \text{var}(P) \text{ and } \mu(P) \in G \} \cup \\
     \{ \mu = (\nu, J \text{ EQ } o) \mid \text{dom}(\mu) = \text{var}(P), (\nu(s), \nu(p), J) \in G, \text{ and } J \in U \}
     \]
   - else
     \[
     [P]_D = \{ \mu = (\nu, \text{true}) \mid \text{dom}(\mu) = \text{var}(P) \text{ and } \mu(P) \in G \}
     \]

2. If $P$ is $P_1$ AND $P_2$ then $[(P_1 \text{ AND } P_2)]_D = [P_1]_D \sqcap [P_2]_D$

3. If $P$ is $P_1$ UNION $P_2$ then $[(P_1 \text{ UNION } P_2)]_D = [P_1]_D \cup [P_2]_D$

In the first item of the above definition the “else” part is to accommodate the case in which evaluation has to be done as in stSPARQL. The “if” part accommodates the case in which the triple pattern involves a spatial literal (illustrated in Example 5.15 below). Here, there are two alternatives: the graph contains a triple matching with every component of the triple pattern (i.e., triples which have the spatial literal at the object position), or it contains a triple with an unknown literal in the object position. We catch a possible match for the second case by adding in the mapping the constraint that restricts the value of that unknown literal to the spatial literal of the triple pattern (i.e., $J \text{ EQ } o$).

**Example 5.15.** Let us first give two examples for the evaluation of triple patterns over the database $D$ of Example 5.6

\[
[(?F, \text{ occurredIn}, \text{ "x} \geq 1 \land x \leq 2 \land y \geq 1 \land y \leq 2\text{"})]_D = \\
\{\{(?F \rightarrow \text{ fire1}), \_\text{region1} \text{ EQ } \text{"x} \geq 1 \land x \leq 2 \land y \geq 1 \land y \leq 2\text{"}\}\}
\]

\[
[(?F, \text{ occurredIn}, \text{ "x} \geq 24.81 \land x \leq 24.84 \land y \geq 35.30 \land y \leq 35.33\text{"})]_D = \\
\{\{(?F \rightarrow \text{ fire2}), \text{true}\}\} \cup \\
\{\{(?F \rightarrow \text{ fire1}), \_\text{region1} \text{ EQ } \text{"x} \geq 24.81 \land x \leq 24.84 \land y \geq 35.30 \land y \leq 35.33\text{"}\}\}
\]

**Example 5.16.** Let us now give an example of an evaluation of graph pattern $P_1$ AND $P_2$ over the database $D$ of Example 5.5 where $P_1, P_2$ are the triple patterns $(?F, \text{type}, \text{Fire})$ and $(?F, \text{occurredIn}, ?R)$ respectively. According to the above definition, we have:

\[
[(P_1 \text{ AND } P_2)]_D = [P_1]_D \sqcap [P_2]_D = \\
[(?F, \text{type}, \text{Fire})]_D \sqcap [(?F, \text{occurredIn}, ?R)]_D = \\
\{(?(F \rightarrow \text{ fire1}, \text{true})_\text{domain} \sqcap (?(F \rightarrow \text{ fire1}, ?R \rightarrow \_\text{region1}, \text{true}))\} = \\
\{(?(F \rightarrow \text{ fire1}, ?R \rightarrow \_\text{region1}, \text{true})\}
\]

As we have explained in the introduction of this chapter, we will consider a simpler version of stSPARQL. As you can see from the following definition, we have simplified the notion of spatial term and do not consider any of the complex functions such as $MBB$, $AREA$, etc. that stSPARQL has.

**Definition 5.17.** A spatial term is a spatial literal from the set $C$ or a spatial variable from the set $V_x$.

Before proceeding to the formal definition of the evaluation of filter expressions the following definitions are needed.

**Definition 5.18.** Let $R$ be a Boolean combination of atomic $L$-constraints. We denote by $\text{var}(R)$ the set of spatial variables contained in $R$. 

D2.3 Theoretical results on query processing for RDF/SPARQL with time and space 47
The next definition defines the concept of an stSPARQL query.

**Definition 5.20.** An stSPARQL query is a pair \((W, P)\) where \(W\) is a set of spatial and non-spatial variables and \(P\) is a graph pattern.

**Example 5.18.** In the algebraic syntax of stSPARQL we consider in this chapter, the query of Example 5.9 can be expressed as follows:

\[
\{ \text{?F} \},
\text{(?F, type, Fire) AND (?F, occurredIn, ?R) AND}
\text{FILTER (?R NTPP "x \geq 24.82 \land x \leq 24.83 \land y \geq 35.31 \land y \leq 35.32")} \]

**Example 5.19.** Let \(\mu = (\nu, \theta)\) be a conditional mapping. For a boolean combination of atomic \(\mathcal{L}\)-constraints \(R\) we denote by \(\mu(R)\) the \(\mathcal{L}\)-constraint obtained from \(R\) by replacing each variable \(x \in \text{var}(R) \cap \text{dom}(\mu)\) by \(\nu(x)\).

The evaluation of filters involving \(\mathcal{L}\)-constraints can now be defined as follows. Notice that the evaluation imposes extra constraints on the mappings that have already been computed. The evaluation of other kinds of filters involving non-spatial variables is as in standard SPARQL [PAG06].

**Definition 5.20.** Given an stRDF\(^i\) database \(D = (G, \phi)\), a graph pattern \(P\) and an atomic \(\mathcal{L}\)-constraint \(R\), we have:

\[
[P \text{ FILTER } R]_D = \{\mu' = (\nu, \theta') | \mu = (\nu, \theta) \in [P]_D, \theta' \text{ is } \theta \land \mu(R) \text{ and } \text{var}(R) \subseteq \text{dom}(\mu)\}
\]

The extension of FILTER to the case that \(R\) is a Boolean combination of atomic \(\mathcal{L}\)-constraints is now easy to define and is omitted. Similarly, the extension of FILTER to the case that \(R\) contains also the built-in conditions of standard SPARQL [PAG06] is easy to define and is omitted.

The following example illustrates the definition and shows that the purpose of constraint \(\mu(R)\) is to deal in a uniform way with the case that the object of a triple is a constant or an unknown literal.

**Example 5.17.** Based on the evaluation of the graph pattern of Example 5.16, the evaluation of the graph pattern \(((P_1 \land P_2) \text{ FILTER } R)\), where \(R\) is the TRCL-constraint \((?R \text{ NTPP "x } \geq 24.82 \land x \leq 24.83 \land y \geq 35.31 \land y \leq 35.32")\), is the following:

\[
[((P_1 \land P_2) \text{ FILTER } R)]_D = [((?F, \text{type, Fire}) \land (?F, \text{occurredIn, ?R})]
\text{FILTER (?R NTPP "x } \geq 24.82 \land x \leq 24.83 \land y \geq 35.31 \land y \leq 35.32")}_D = 
\text{} \{\{(?F \rightarrow \text{fire1, ?R \rightarrow xregion1}, x\text{region1 NTPP }"x \geq 24.82 \land x \leq 24.83 \land y \geq 35.31 \land y \leq 35.32")\}\}
\]

The next definition defines the notion of answer to an stSPARQL query. Notice that in contrast to stSPARQL queries over stRDF graphs, stSPARQL queries over stRDF\(^i\) databases have answers that consist of conditional mappings so they might be hard to understand.
Definition 5.22. Let \( q = (W, P) \) be an stSPARQL query. The answer to \( q \) over a database \( D = (G, \phi) \) is the set of conditional mappings \( \{ \mu_W \mid \mu \in [P]_D \} \).

Notice that the conditional mappings of the answer to a query might contain unknown literals. These literals are constrained by the constraint store \( \phi \), therefore \( \phi \) can be understood to be implicitly included in the answer (this can also be done formally by considering answers to be pairs).

Example 5.20. The answer to the query from Example 5.19 can be obtained from the evaluation of the respective graph pattern from Example 5.17. The answer is a set that contains only the following mapping:

\[
(\{ ?F \rightarrow \text{fire1} \}, \text{region1 NTTP} \, x \geq 24.82 \land x \leq 24.83 \land y \geq 35.31 \land y \leq 35.32^*)
\]

This answer is conditional. Because the information in the database of Example 5.5 is indefinite (the exact geometry of \( \text{region1} \) is not known), we cannot say for sure whether \( \text{fire1} \) satisfies the requirements of the query. These requirements are satisfied under the condition given in the above mapping.

Remark. It is easy to see how we can simplify the answer to a query \( q \) over database \( D = (G, \phi) \) using the constraints involving unknown literals in the constraint store \( \phi \). There are two things that we can do. The first is to remove from the answer mappings \( \mu = (\nu, \theta) \) where \( \theta \) is a conjunction which contains a constraint \( \sigma \) such that \( \phi \models \neg \sigma \). The second thing that we can do is simplify the conjunction \( \theta \) of a mapping \( \mu = (\nu, \theta) \) by deleting constraints \( \sigma \) such that \( \phi \models \sigma \). Of course, this simplification step can also be performed in some intermediate step of query evaluation and not just at the end, when the final answer has been computed.

5.5 A Representation System for stRDF\(^i\) and stSPARQL

Let us now recall the semantics of stRDF\(^i\) as given through \( \text{Rep} \). The set \( \text{Rep}(D) \) is what an stRDF\(^i\) database \( D \) represents. Clearly, if we were to evaluate a query \( q \) over \( D \), we would use the semantics of stRDF\(^i\) and evaluate \( q \) over any stRDF graph of \( \text{Rep}(D) \) as follows:

\[
q(\text{Rep}(D)) = \{ q(G) \mid G \in \text{Rep}(D) \}
\]

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

\[
\text{Rep}(q(D)) = q(\text{Rep}(D))
\] (5.1)

Seeing formula (5.1) from a different perspective, it allows us to compute the answer to any query over an stRDF\(^i\) database in a consistent way with respect to the semantics of stRDF\(^i\) and without retreating to computing the answer by applying the query on all possible stRDF graphs. \( q(D) \) can be computed using the algebra of Section 5.4 above. But can the algebra of Section 5.4 compute always such a database \( q(D) \) representing \( q(\text{Rep}(D)) \)? The answer is no in general. The following example illustrates this negative fact.
Example 5.21. Consider the stRDF\(^i\) database \(D = (G, \phi)\), where \(G = \{(s, p, o)\}\) and \(\phi = true\), i.e., \(D\) contains the single triple \((s, p, o)\). Consider now a query \(q\) over \(D\) that selects all triples having \(s\) as the subject. Query \(q\) would be specified as the graph pattern \((s, ?p, ?o)\) and evaluated as \([\{(s, ?p, ?o)\}]_D\) using Definition 5.16. Then, the triple \((s, p, o)\) and nothing else is in the resulting database \(q(D)\). However, equation (5.1) 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 \(g \in q(Rep(D))\). Unless the closed world assumption is adopted, which we do not want to do since we are in the realm of RDF, condition (5.1) has to be relaxed.

Given a fixed query language, two incomplete databases are not distinguishable if they give the same certain answer to every query in the language. The next definition formalizes this fact using the concept of \(Q\)-equivalence where \(Q\) is a query language. Originally this concept was defined for incomplete relational databases in [IL84].

Definition 5.23. Let \(Q\) be a query language, and \(\mathcal{G}, \mathcal{H}\) two sets of stRDF graphs. \(\mathcal{G}\) and \(\mathcal{H}\) are called \(Q\)-equivalent (denoted by \(\mathcal{G} \equiv Q \mathcal{H}\)) if they give the same certain answer to every query in the language, that is, \(\cap q(\mathcal{G}) = \cap q(\mathcal{H})\) for all \(q \in Q\).

Given the above discussion, we can now define the notion of a representation system which gives a formal characterization of the correctness (in terms of semantics) of computing the answer to a query directly on an incomplete database (as we did in Section 5.4) instead of using the semantics. This definition introduces the representation system concept of [IL84] to the stRDF model.

Definition 5.24. Let \(\mathcal{D}\) be the set of all stRDF\(^i\) databases, \(\mathcal{G}\) the set of all stRDF graphs, \(Rep : \mathcal{D} \rightarrow \mathcal{G}\) a function determining the set of possible interpretations of an stRDF\(^i\) database, and \(Q\) a query language. The triple \((\mathcal{D}, Rep, Q)\) is a representation system if for all \(D \in \mathcal{D}\) and all \(q \in Q\), there exists an stRDF\(^i\) database \(q(D) \in \mathcal{D}\) such that the following condition is satisfied:

\[
Rep(q(D)) \equiv Q q(Rep(D)).
\]

The next step towards our development of a representation system for stRDF\(^i\) and stSPARQL is to instantiate the query language \(Q\) to consider, and define the notions of monotonicity and coinitality as is done in [IL84].

Notation 5.2. We denote as \(Q_{OP}\) the set of all queries consisting of triple patterns, and graph patterns involving only the operator \(OP\). Consequently, \(Q_{AND}\) is the set of all queries consisting only of AND graph patterns, \(Q_{UNION}\) is the set of all queries consisting only of UNION graph patterns, \(Q_{OPT}\) is the set of all queries consisting only of OPT graph patterns, \(Q_{AND,FILTER}\) is the set of all queries consisting only of AND-FILTER graph patterns, and so on. For readability, when more than one graph patterns are involved (e.g., AND-FILTER) the respective set of queries expressible in these graph patterns is denoted using the initial characters of the graph patterns involved (e.g., \(Q_{AF}\)).

Definition 5.25. A query language \(Q\) is monotone if for every \(q \in Q\) and RDF graphs \(G\) and \(H\) such that \(G \subseteq H\), it is \(q(G) \subseteq q(H)\).

In the following, we prove results concerning monotonicity for RDF and SPARQL. These results (Propositions 5.1 and 5.2) have been presented recently in [AP11]. We give detailed proofs since these are omitted in [AP11]. The results are the same for stRDF and the subset of stSPARQL we study and can be obtained with trivial modifications of the proofs we give below.

Proposition 5.1. The query language \(Q_{AUF}\) is monotone.

Proof. The proof is done by induction on the structure of the graph pattern \(P \in Q_{AUF}\). Let \(G\) and \(H\) be RDF graphs such that \(G \subseteq H\).
(i) $P$ is a triple pattern $p$ (base case):

Let $\mu \in [p]_G$. Then, it is the case that $\mu(p) \in G$. Since $G \subseteq H$, we have that $\mu(p) \in H$, and thus $\mu \in [p]_H$.

(ii) Inductive step:

(a) $P$ is $P_1$ AND $P_2$:

We have $[P_1]_G \subseteq [P_1]_H$ and $[P_2]_G \subseteq [P_2]_H$ from the inductive hypothesis. We shall prove that this is the case for $P$ as well, that is, $[P]_G \subseteq [P]_H$.

Suppose that $\mu \in [P]_G$. Then, there exist compatible mappings $\mu_1 \in [P_1]_G$ and $\mu_2 \in [P_2]_G$, such that $\mu = \mu_1 \cup \mu_2$. Due to the inductive hypothesis, $\mu_1 \in [P_1]_H$ and $\mu_2 \in [P_2]_H$. Because we also have $\mu = \mu_1 \cup \mu_2$, we have that $\mu \in [P]_H$, which proves our claim.

(b) $P$ is $P_1$ UNION $P_2$:

We have $[P_1]_G \subseteq [P_1]_H$ and $[P_2]_G \subseteq [P_2]_H$ from the inductive hypothesis. We shall prove that this is the case for $P$ as well, that is, $[P]_G \subseteq [P]_H$.

Suppose that $\mu \in [P]_G$. Then, $\mu \in [P_1]_G \cup [P_2]_G$ and thus $\mu \in [P_1]_G$ or $\mu \in [P_2]_G$. Due to the inductive hypothesis, whenever $\mu \in [P_1]_G$ or $\mu \in [P_2]_G$, it is the case that $\mu \in [P_1]_H$ or $\mu \in [P_2]_H$, and equivalently that $\mu \in [P_1]_H \cup [P_2]_H$, which proves that $\mu \in [P]_H$, and in turn that $[P]_G \subseteq [P]_H$.

(c) $P$ is $P_1$ FILTER $R$:

We have $[P_1]_G \subseteq [P_1]_H$ from the inductive hypothesis. We shall prove that this is the case for $P$ as well, that is, $[P]_G \subseteq [P]_H$.

Given the semantics for the evaluation of FILTER graph patterns of the form $P_1$ FILTER $R$, we observe that $[P_1$ FILTER $R]_G$ is the restriction of the set $[P_1]_G$ to the set of mappings $\mu$ that satisfy condition $R$, that is, $\{\mu \in [P_1]_G \mid \mu \models R\}$. Thus, if $\mu \in ([P_1$ FILTER $R]_G]$, then we have that $\mu \in [P_1]_G$, or that $([P_1$ FILTER $R]_G \subseteq [P_1]_G$.

Suppose now that $\mu \in [P]_G$. Then, from the above subset relation we have that $\mu \in [P_1]_G$ and also that $\mu \models R$. From the inductive hypothesis that $[P_1]_G \subseteq [P_1]_H$, we get that $\mu \in [P_1]_H$ and because of the fact that $\mu \models R$, we have that $\mu \in [P]_H$.

Proposition 5.2. The query language $Q_{OPT}$ is not monotone.

Proof. We will prove Proposition 5.2 using a counter-example. Let $G$ and $H$ be RDF graphs such that $G = \{(a, b, c)\}$ and $H = \{(a, b, c), (d, b, c)\}$. Clearly, it is that $G \subseteq H$. Consider also the query $q = (P_1$ OPT $P_2)$, where $P_1 = (a, b, ?X)$ and $P_2 = (d, b, ?Y)$. Evaluating each one of the triple patterns $P_1$ and $P_2$ over $G$, we get the following:

$[P_1]_G = \{\mu_1: \frac{?X}{c}\}$

$[P_2]_G = \{\}$

$[P_1 \cap P_2]_G = \{\}$

$[P_1 \setminus P_2]_G = \{\mu_1\}$

$[q]_G = [P_1 \cup P_2]_G = ([P_1]_G \cup [P_2]_G) \cup ([P_1]_G \setminus [P_2]_G) = \{\} \cup \{\mu_1\} = \{\mu_1\}$

Similarly, we get the following for $H$:

$[P_1]_H = \{\mu_1: \frac{?X}{c}\}$

$[P_2]_H = \{\mu_2: \frac{?Y}{e}\}$

$[P_1 \cap P_2]_H = \{\mu_3: \frac{?X}{c}, \frac{?Y}{e}\}$

$[P_1 \setminus P_2]_H = \{\}$

$[q]_H = [P_1 \cup P_2]_H = ([P_1]_H \cup [P_2]_H) \cup ([P_1]_H \setminus [P_2]_H) = \{\} \cup \{\} = \{\}$
The proof is similar to the one given in [IL84, Lemma 4.2]. We have to prove that the following sets are coinitial.

$$G \approx H$$

We observe that while $G \subseteq H$, we have that $q(G) \not\subseteq q(H)$, thus $Q_{OPT}$ is not monotone.

**Lemma 5.1.** Any query language $Q$ involving the $OPT$ operator is not monotone.

**Proof.** Lemma 5.1 is a direct consequence of Proposition 5.2.

We denote by $\text{stSPARQL}_M$ the monotone part of the stSPARQL query language, that is, all expressions composed of the AND, UNION, and FILTER graph patterns.

**Definition 5.26.** Let $G$ and $H$ be sets of RDF graphs. We say that $G$ and $H$ are coinitial, denoted by $G \approx H$, if for any $G \in G$ there exists $H \in H$ such that $H \subseteq G$, and for any $H \in H$ there exists $G \in G$ such that $G \subseteq H$.

**Example 5.22.** The following sets are coinitial.

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

$$H = \{(a,b,c),(a,c,d)\}, \{(a,b,c)\}$$

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

**Proposition 5.3.** Let $Q$ be a monotone query language, and $G$ and $H$ sets of RDF graphs. If $G \approx H$ then for any $q \in Q$ it holds that $q(G) \approx q(H)$.

**Proof.** The proof of Proposition 5.3 is straightforward from the monotonicity property.

**Lemma 5.2.** Let $G$ and $H$ be sets of RDF graphs. If $G$ and $H$ are coinitial, then $G \equiv_{\text{stSPARQL}_M} H$.

**Proof.** The proof is similar to the one given in [IL84] Lemma 4.2. We have to prove that $\bigcap q(G) = \bigcap q(H)$ for every $q \in \text{stSPARQL}_M$. Let $G \approx H$. Then, from Proposition 5.3, we have that $q(G) \approx q(H)$ for every $q \in \text{stSPARQL}_M$. Thus, for every $q(G) \in q(G)$ there exists an $q(H_G) \in q(H)$ such that $q(H_G) \subseteq q(G)$. So, we have that

$$\bigcap q(G) = \bigcap q(G) \supseteq \bigcap q(H_G) \supseteq \bigcap q(H) = \bigcap q(H)$$

To see why $\bigcap q(G) \supseteq \bigcap q(H_G)$, notice that $\bigcap q(G)$ and $\bigcap q(H_G)$ can be written as $q(G_1) \cap q(G_2) \cap \cdots$ and $q(H_{G_1}) \cap q(H_{G_2}) \cap \cdots$ respectively, and that each set $q(H_{G_i}) \subseteq q(G_i)$. Therefore, if an element $x$ is in $\bigcap q(H_G)$, 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 $\bigcap q(H_G) \supseteq \bigcap q(H)$, notice that the relation can be written as $\bigcap q(H_G) \supseteq \bigcap q(H)$, where $H_G \equiv \{H \in H \mid H \subseteq G \text{ for some } G \in G\}$. Thus, $H_G \subseteq H$, and therefore, we have that $\bigcap H_G \supseteq \bigcap H$. Similarly if $q$ is a monotone query, we have $q(H_G) \subseteq q(H)$ and $\bigcap q(H_G) \supseteq \bigcap q(H)$.

We work similarly to prove that $\bigcap q(H) \supseteq \bigcap q(G)$. 

D2.3 Theoretical results on query processing for RDF/SPARQL with time and space 52
We will now present a central theorem which characterizes the evaluation of monotone stSPARQL \( M \) queries (Theorem 5.3). Before we do this, we need a few definitions and preliminary results.

The first definition allows us to apply a valuation to a conditional mapping.

**Definition 5.27.** Let \( v : U \rightarrow C \) be a valuation and \( \mu = (\nu, \theta) \) be a conditional mapping such that \( M_L \models v(\theta) \). Then \( v(\mu) \) denotes the mapping that results from substituting in \( \epsilon \)-mapping \( \nu \) the constant \( v(l) \) for each unknown literal \( l \).

Note that applying a valuation to a conditional mapping, we get an ordinary mapping like in the case of stRDF simply by disregarding the constraint that results since it is equivalent to \( \text{true} \). In a similar way, we can extend a valuation \( v \) to a set of mappings \( \Omega \) as follows.

**Definition 5.28.** Let \( \Omega \) be a set of conditional mappings and \( v : U \rightarrow C \) a valuation. Then \( v(\Omega) = \{ v(\mu) \mid \mu = (\nu, \theta) \in \Omega \text{ and } M_L \models v(\theta) \} \).

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 5.4.** Let \( v : U \rightarrow C \) be a valuation and \( \mu_1 = (\nu_1, \theta_1), \mu_2 = (\nu_2, \theta_2) \) be two possibly 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 \( M_L \models v(\theta_1) \) and therefore \( M_L \models v(\theta_2) \)).

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

The next definition allows us to apply a valuation to an stRDF\(^i\) database.

**Definition 5.29.** Let \( v : U \rightarrow C \) be a valuation and \( D = (G, \phi) \) an stRDF\(^i\) database such that \( M_L \models v(\phi) \). Then \( v(D) \) denotes the stRDF graph that results from the application of \( v \) to \( G \).

The next proposition is very useful for the proof of our Representation Theorem.

**Proposition 5.5.** Let \( D = (G, \phi) \) be an stRDF\(^i\) database, \( q \) an stSPARQL\(_M\) query, and \( v \) a valuation such that \( M_L \models v(\phi) \). Then, the formula
\[
v(q(D)) = q(v(D)) \quad (5.2)
\]
implies the formula
\[
\text{Rep}(q(D)) \approx q(\text{Rep}(D)). \quad (5.3)
\]

**Proof.** From the definition of \( \text{Rep} \) and our formula assumption \( 5.2 \) we have that
\[
\text{Rep}(q(D)) = \{ H \mid \text{there exists a valuation } v \text{ such that } M_L \models v(\phi) \text{ and } H \supseteq v(q(D)) = q(v(D)) \}. \quad (5.4)
\]

Let \( H \in \text{Rep}(q(D)) \). Then \( H \supseteq q(v(D)) \) for some \( v \) such that \( M_L \models v(\phi) \). From the definition of \( \text{Rep}(D) \) there exists an \( H' \in \text{Rep}(D) \) such that \( H' = v(D) \). Therefore \( q(H') = q(v(D)) \) and thus
\[
q(H') \in q(\text{Rep}(D)).
\]
This proves that for each \( H \in \text{Rep}(q(D)) \) there exists an \( q(H') \in \text{q}(\text{Rep}(D)) \) such that \( q(H') \subseteq H \).

To prove \( \text{(5.3)} \), we need to show the same for the other direction.

Let \( H \in \text{q}(\text{Rep}(D)) \), then there exists an \( H' \in \text{Rep}(D) \) such that \( H = q(H') \). Since \( H' \in \text{Rep}(D) \), \( H' \supseteq v(D) \) for some valuation \( v \) such that \( M_C \models v(\phi) \). Therefore,

\[
q(H') \supseteq q(v(D))
\]

which from our assumption \( \text{(5.2)} \) gives

\[
q(H') \supseteq v(D).
\]

Notice now that by the definition of \( \text{Rep}(q(D)) \) as given in \( \text{(5.4)} \) we have

\[
v(q(D)) \in \text{Rep}(q(D)).
\]

Therefore, we showed that there exists an \( H'' \in \text{Rep}(q(D)) \) such that \( H'' \subseteq H \).

Hence

\[
\text{Rep}(q(D)) \approx q(\text{Rep}(D)).
\]

We are now ready to prove our main result.

**Theorem 5.3** (Representation Theorem). The triple \( (\mathcal{D}, \text{Rep}, \text{stSPARQL}_M) \) is a representation system.

**Proof.** To prove Theorem \( \text{(5.3)} \), it is sufficient to show that for any \( D = (G, \phi) \in \mathcal{D} \) and any query \( q \in \text{stSPARQL}_M \) it is possible to define \( q(D) \) in such a way that \( \text{Rep}(q(D)) \equiv_{\text{stSPARQL}_M} \text{q}(\text{Rep}(D)) \).

By Lemma \( \text{(5.2)} \) it is sufficient to prove that

\[
\text{Rep}(q(D)) \approx q(\text{Rep}(D)) \tag{5.5}
\]

From Proposition \( \text{(5.5)} \) it now suffices to prove \( \text{(5.2)} \) for any valuation \( v \) such that \( M_C \models v(\phi) \).

Proof of \( \text{(5.2)} \) is done by induction on the structure of graph patterns \( P \in \text{stSPARQL}_M \).

- \( P \) 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 \( M_C \models v(\theta') \).

  We now distinguish two cases corresponding to the two cases of Definition \( \text{(5.16)} \) (1):

  (i) In this case \( o \in C \) (i.e., \( o \) is a spatial literal). In this case \( \text{dom}(\nu') \) does not contain any spatial variable, hence the application of \( v \) to \( \mu' \) leaves \( \nu' \) 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 obviously \( \mu(P) \in v(D) \) and therefore \( \mu \in [P]_{v(D)} \).

  If \( \mu = \nu' \) is an element of the second set then \( \theta' \) is \( \bigvee \text{EQ} \) \( o \). Since \( M_C \models v(\theta') \), we have \((\mu(s), \mu(p), o) \in v(D) \). Therefore \( \mu \in [P]_{v(D)} \).

  (ii) In this case \( o \in I \cup B \cup L \cup V \). Therefore \( \mu'(o) \in I \cup B \cup L \cup U \cup C \) and \( v(\mu'(P)) \in v(D) \).

  The latter fact together with the fact that \( v(\mu') = \mu \) gives that \( \mu(P) \in v(D) \). Therefore \( \mu \in [P]_{v(D)} \).
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 5.16 (1):

(i) In this case $o \in C$. Then, $dom(\mu)$ does not contain any spatial variable.

Now, we have two cases corresponding to the two sets making up $[P]_D$ in Definition 5.16 (1):

- A conditional mapping $\mu' = (\mu, \theta)$ is an element of the first set, where $\theta$ is true. Since $\mu(P) \in v(D)$ we have $\mu(P) \in G$ or equivalently $\mu'(P) \in G$. Therefore $\mu' \in [P]_D$, and since $\theta$ is true, we have $M_L \models v(\theta)$. Thus we can apply $v$ to $\mu' \in [P]_D$ and get

$$v(\mu') \in v([P]_D)$$

which is equivalent to

$$\mu \in v([P]_D).$$

- A conditional mapping $\mu' = (\mu, J \ E Q o)$ is an element of the second set such that $v(\mu'(s), \mu'(p), J) \in G$. We also have $v(J) = o$ and therefore $M_L \models v(J \ E Q o)$. Thus $\mu \in v([P]_D)$, which is what we want.

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

If $o \in I \cup B \cup L \cup V_{ns}$, then we can form the conditional mapping $\mu' = (\mu, true)$. It is easy to see that $\mu' \in [P]_D$. Applying $v$ to this relation, we get

$$v(\mu') \in v([P]_D)$$

and thus

$$\mu \in v([P]_D)$$

since the application of $v$ leaves $\mu'$ (and $\mu$) unchanged.

Now if $o \in V_s$, there exists a conditional mapping $\mu' = (\nu', true)$ such that $\mu'$ and $\mu$ are possibly compatible and $dom(\mu') = dom(\mu)$. The conditional mapping $\mu'$ 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 $v(\mu'(o)) = \mu(o)$. In either case $\mu(P) \in v(D)$ implies

$$v(\mu'(P)) \in v(D).$$

from which eliminating $v$ we get

$$\mu'(P) \in G$$

or equivalently

$$\mu' \in [P]_D.$$

Applying $v$ to the last 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 $v([P_1 \ AND \ P_2]_D) = [P_1 \ AND \ P_2]_{v(D)}$. 

D2.3 Theoretical results on query processing for RDF/SPARQL with time and space
Let \( \mu \in v([P_1 \ AND \ P_2]_{v(D)}) \). Therefore there exists a conditional mapping \( \mu' = (\nu', \theta') \in [P_1 \ AND \ P_2]_{D} \) such that \( \mu = v(\mu') \) and \( M_{\mathcal{C}} \models v(\theta') \). 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 5.4 and the fact that \( M_{\mathcal{C}} \models v(\theta') \), we have that
\[
\mu = v(\mu') = v(\mu_1' \bowtie \mu_2') = v(\mu_1') \bowtie v(\mu_2').
\]
Also, because \( M_{\mathcal{C}} \models v(\theta') \) it also holds \( M_{\mathcal{C}} \models v(\theta_1') \) and \( M_{\mathcal{C}} \models v(\theta_2') \). Therefore, \( v(\mu_1') \in v([P_1]_{D}) \) and \( v(\mu_2') \in v([P_2]_{D}) \). Notice also that because \( \mu_1' \) and \( \mu_2' \) 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
\[
\mu \in v([P_1 \ AND \ P_2]_{D}).
\]
From the equalities of the inductive hypothesis, we now get
\[
\mu \in ([P_1]_{v(D)} \bowtie [P_2]_{v(D)})
\]
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 \( \mu \) be a mapping such that \( \mu \in [P_1 \ AND \ P_2]_{v(D)} \). Then \( \mu \in ([P_1]_{v(D)} \bowtie [P_2]_{v(D)}) \), 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 \( \mu_1 = v(\mu_1') \), \( \mu_2 = v(\mu_2') \), \( M_{\mathcal{C}} \models v(\theta_1') \) and \( M_{\mathcal{C}} \models v(\theta_2') \). Notice also that \( \mu_1' \) and \( \mu_2' \) are possibly compatible.

From Proposition 5.4 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 \( \mu_1', \mu_2' \) 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 possible compatible mapping such that \( \mu' = \mu_1' \bowtie \mu_2' \). Since \( M_{\mathcal{C}} \models v(\theta_1') \) and \( M_{\mathcal{C}} \models v(\theta_2') \), the definition of join of two compatible mappings gives us \( M_{\mathcal{C}} \models v(\theta') \). Therefore we can apply the valuation \( v \) to \( \mu' \) 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
\[
\mu \in v([P_1 \ AND \ P_2]_{D}).
\]

Thus, \( P \) 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 \( \mu \) 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 \( \mu \in v([P_1 \ UNION \ P_2]_{D}) \), which is equivalent to \( \mu \in v([P_1 \ UNION \ P_2]_{D}) \).
\[ P \text{ is } P_1 \text{ FILTER } R. \]

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

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

Without loss of generality, we give the proof only for the case of filters that are atomic \( \mathcal{L} \)-constraints (Definition 5.20). Let \( \mu \) be in \([P_1 \text{ FILTER } R]_{v(D)}\). By definition, this is equivalent to \( \mu \in [P_1]_{v(D)} \) and \( \mu \models R \). From the inductive hypothesis, we now have

\[
\mu \in v([P_1]_D).
\]

Thus, there exists a conditional mapping \( \mu' = (\nu', \theta') \in [P_1]_D \) such that \( v(\mu') = \mu \) and \( M_\mathcal{C} \models v(\theta') \).

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

The way \( \mu_1 \) and \( \mu' \) have been defined above, together with the definition of the evaluation of FILTER graph patterns give us

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

We can apply the valuation \( v \) to \( \mu_1 \in [P_1 \text{ FILTER } R]_D \) and get

\[
v(\mu_1) \in v([P_1 \text{ FILTER } R]_D)
\]

which is equivalent to \( \mu \in v([P_1 \text{ FILTER } R]_D) \).

This proof establishes that \([P_1 \text{ FILTER } R]_{v(D)} \subseteq v([P_1 \text{ FILTER } R]_D)\). The other direction of the proof is similar and goes as follows.

Let \( \mu \) be a mapping in \( v([P_1 \text{ FILTER } R]_D) \). Then there exists a conditional mapping \( \mu_1 = (\nu_1, \theta_1) \in [P_1 \text{ FILTER } R]_D \) such that \( v(\mu_1) = \mu \) and \( M_\mathcal{C} \models v(\theta_1) \). Therefore, from the definition of FILTER evaluation there exists a conditional mapping \( \mu_2 = (\nu_2, \theta_2) \) such that \( \mu_2 \in [P_1]_D \), where \( \theta_2 = \theta_1 \land \mu_2(R) \). Since \( M_\mathcal{C} \models v(\theta_1) \), then it holds that \( M_\mathcal{C} \models v(\theta_2) \) and \( M_\mathcal{C} \models v(\mu_2(R)) \). Thus

\[
v(\mu_2) = v(\mu_1) = \mu \in [P_1]_D.
\]

Now using the inductive hypothesis, we have

\[
\mu \in [P_1]_{v(D)}.
\]

Because \( M_\mathcal{C} \models v(\mu_1(R)) \) and \( \mu = v(\mu_1) \), we have \( M_\mathcal{C} \models \mu(R) \). Thus we also have \( \mu \models R \). Hence

\[
\mu \in [P_1 \text{ FILTER } R]_{v(D)}.
\]

5.6 Complexity Results

In this section we study the complexity of evaluating stSPARQL queries over stRDF\(^3\) databases. We are concerned with the data complexity of query evaluation. We focus on the stSPARQL\(_M\) subset of the stSPARQL query language, since stSPARQL\(_M\) forms a representation system for stRDF\(^3\) databases as proved in Theorem 5.79. Thus, we are interested in studying the data complexity of what [Gra91] calls the certainty problem, i.e., the problem of deciding whether a given set of mappings is in the certain answer to a query. This is formalized by the following decision problem.
**Definition 5.30.** Let \( q \) be an stSPARQL query. The certainty problem for query \( q \), set of mappings \( M \), and stRDF\(^i\) database \( D \), is to decide whether \( M \subseteq \bigcap q(\text{Rep}(D)) \). We denote this problem by \( \text{CERT}(q, M, D) \).

**Example 5.23.** Let \( D \) be the database of Example 5.5, \( q \) the stSPARQL query of Example 5.18, and \( M = \{\{?F \rightarrow \text{fire}\}\} \). The answer to the certainty problem for \( q \), \( M \) and \( D \) is “yes”. This is because as we have seen in Example 5.9, we have that \( M \subseteq \bigcap q(\text{Rep}(D)) \).

### 5.6.1 Data Complexity of \( \text{CERT}(q, M, D) \)

We now study the data complexity of \( \text{CERT}(q, M, D) \).

**Theorem 5.4.** Let \( D = (G, \phi) \) be an stRDF\(^i\) database, \( q \) an stSPARQL query, and \( M \) a set of mappings. Then, \( \text{CERT}(q, M, D) \) is equivalent to deciding whether the following formula is true in \( M_\mathcal{L} \):

\[
\bigwedge_{\mu \in M} (\forall \mathcal{J})(\phi(\mathcal{J}) \rightarrow \Theta(\mu, q, D, \mathcal{J})) \tag{5.6}
\]

In the above formula:

- \( \mathcal{J} \) is the vector of all the unknown literals in the database \( D \).
- \( \Theta(\mu, q, D, \mathcal{J}) \) is a disjunction \( \theta_1' \lor \cdots \lor \theta_k' \) that is constructed as follows. For each conditional mapping \( \mu_i' = (\nu_i, \theta_i) \) in \( \text{[q]}_D \) such that \( \mu \) and \( \mu_i' \) are possibly compatible, \( \theta_i \) is:
  - \( \theta_i \) if \( \mu \) and \( \mu_i' \) are compatible.
  - \( \theta_i \land \bigwedge_{x} (\nu_i(x) \ \text{EQ} \ \mu(x)) \), otherwise. Here, \( x \) ranges over the variables \( x \in \text{dom}(\mu) \cap V_s \) such that \( \nu_i(x) \) is an unknown literal.

Note that if \( \mu \) is not possibly compatible to any of \( \nu_i \), then \( \Theta(\mu, q, D, \mathcal{J}) \) is taken to be false.

**Proof.** The proof is easy and is omitted.

**Theorem 5.5.** Let \( \mathcal{L} \) be a many sorted first-order constraint language. If the satisfiability problem for atomic \( \mathcal{L} \)-constraints can be solved in \( C \subseteq \text{EXPTIME} \), then the data complexity of \( \text{CERT}(q, M, D) \) problem for databases \( D \) and queries \( q \) with \( \mathcal{L} \)-constraints is \text{EXPTIME}.

**Proof.** Each conjunct of formula (5.6) can be written as follows.

\[
(\forall \mathcal{J})(\phi(\mathcal{J}) \rightarrow \Theta(\mu, q, D, \mathcal{J})) \equiv \\
\neg(\exists \mathcal{J})(\phi(\mathcal{J}) \land \neg \Theta(\mu, q, D, \mathcal{J})) \equiv \\
\neg(\exists \mathcal{J})(\phi(\mathcal{J}) \land \neg \theta_1' \land \cdots \land \neg \theta_k') \tag{5.7}
\]

Each \( \neg \theta_j' \) is \( s_1 \lor \cdots \lor s_n \), where each \( s_j \) is an \( \mathcal{L} \)-constraint given the properties of \( \mathcal{L} \) we assumed in Section 5.2.

To decide (5.7) we transform it into DNF, solve the conjunctions of \( \mathcal{L} \)-constraints that appear as disjuncts (i.e., using constraint network algorithms), and then negate the truth value.
Then, it is simple to decide (5.6) which is conjunctive.

Provided that satisfiability of conjunctions of $L$-constraints can be decided in $\mathcal{L} \subseteq EXPTIME$, the complexity of the decision problem is in EXPTIME due to the DNF transformation since we have data complexity. Even when $\phi$ is a conjunction of $L$-constraints, this is still EXPTIME because of the disjunctions $\theta'_i$.

A language $\mathcal{L}$ with atomic constraints that have the property wanted by Theorem 5.5 is TCL defined in Section 2.2.1. Thus, the theorem holds for TCL. If $\mathcal{L}$ is RCL, then notice that the formula (5.6) of Theorem 5.4 has one alternation of quantifiers and the atomic constraints involved are only RCL-constraints (i.e., order constraints on terms representing vertices of boxes). Then, from [Kou94a], which studies similar first-order languages for temporal constraints, we have that the complexity of the relevant decision problem is coNP-complete, and so is the certainty problem.

The recent paper [LWL11] shows that conjunctions of atomic RCC-5 constraints involving constants that are polygons in $V$-representation (called landmarks in [LWL11]) can be decided in PTIME. This result allows us to apply Theorem 5.5 to a language like PCL, where only RCC-5 constraints are allowed and constants are given in $V$-representation.

It is open whether the theorem holds for TRCL as well since no results are known regarding the complexity of checking conjunctions of TRCL-constraints (naturally, the techniques of [LWL11] should apply; we have not investigated this in any detail).

The following theorem demonstrates cases when the certainty problem is tractable.

### 5.7 Conclusions and Open Problems

In this chapter, motivated by the fact that often geospatial data are imprecise, incomplete or qualitative in nature, we proposed the stRDF$^3$ data model, a model that extends stRDF with incomplete spatial information. After we formally defined our data model (Section 5.2), we gave its formal semantics (Section 5.3) and used it to define formally how stSPARQL queries can be evaluated on stRDF$^3$ databases (Section 5.4). Following the work on incomplete relational databases [IL84, Gra91], we presented our new results concerning the semantics of stRDF$^3$ and stSPARQL query evaluation in this context (Section 5.5). In particular, we identified the subset of stSPARQL operators that make stSPARQL a representation system over the stRDF$^3$ data model. Queries in this subset language, stSPARQL$^M$, can be correctly evaluated according to the semantics of stRDF$^3$. Last, we studied the computational complexity of deciding the certainty problem (Definition 5.30) for stSPARQL$^M$.

The study of the present chapter leaves open the following questions:

- What are tractable cases of the certainty problem?
- Are there any other representation systems from the ones defined in [IL84] that can be profitably introduced to stRDF$^3$?
- What is the exact correspondence of the results of this chapter with query answering in the ABox of DL reasoners like PelletSpatial that have a qualitative spatial reasoning component?

We will deal with the above open problems in the second year of TELEIOS and report our findings in relevant deliverables of WP4.
6. Conclusions

In this deliverable we extended the theoretical foundations of the data models stRDF and stRDF³ and query language stSPARQL presented in Deliverable D2.1 “A data model and query language for an extension of RDF with time and space” \[KKN^{+}11\]. We studied the semantics and the computational complexity of evaluating stSPARQL queries over stRDF graphs and stRDF³ databases.

The results of this deliverable are original and extend the state of the art in theory of geospatial extensions of W3C standards RDF and SPARQL. In fact, to the best of our knowledge, they are the only foundational results in this area; the rest of the published work in this area concentrates on more practical issues.

The technical development of this deliverable forms the theoretical basis of the work on stRDF and stSPARQL foreseen in WP2 of TELEIOS. The implementation of stSPARQL is now the next step of the relevant work in the project and it is undertaken in WP4 “Query processing - Image annotations”.

As we also explained in the introduction, the theoretical work of this deliverable will guide the implementation work we undertake in WP4 in the context of system Strabon. The semantics and computational complexity analysis of Chapters 3 and 4 have already been guiding us to implement stRDF and stSPARQL in Strabon. The material in Chapter 5 of this deliverable will guide our implementation of stRDF³.

Finally, let us point out that although WP2 “Data Models and Query Languages” of TELEIOS ends in M12, we expect our relevant work in this area to continue. We will continue to improve the foundational analysis of stRDF, stRDF³, and stSPARQL as implementation challenges faced in WP4 might give rise to new fundamental problems that need to be solved. Any interesting results of this work will be reported in relevant deliverables of WP4.
Bibliography


D2.3 Theoretical results on query processing for RDF/SPARQL with time and space


