



Transactions on  
**Graph Data and Knowledge**

**Volume 4 | Issue 2 | September, 2026**

**Special Issue: Data Management for (Knowledge) Graphs**

Edited by

Stefania Dumbrava

George Fletcher

Matteo Lissandrini

Riccardo Tommasini



**ISSN 2942-7517**

*TGDK Special Issue Editors*

**Stefania Dumbrava** 

ENSIIE, SAMOVAR, Télécom SudParis, France  
IRIF, INRIA Paris, France  
stefania.dumbrava@ensiie.fr

**George Fletcher** 

Eindhoven University of Technology, The Netherlands  
g.h.l.fletcher@tue.nl

**Matteo Lissandrini** 

University of Verona, Italy  
matteo.lissandrini@univr.it

**Riccardo Tommasini** 

INSA Lyon, France  
CNRS LIRIS, Lyon, France  
riccardo.tommasini@insa-lyon.fr

*ACM Classification 2012*

Information systems → Graph-based database models;  
Information systems → Resource Description Framework (RDF);  
Information systems → Query languages;  
Computing methodologies → Knowledge representation and reasoning

*Published online and open access by*

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany.

Online available at

<https://www.dagstuhl.de/dagpub/2942-7517>.

*Publication date*

September, 2026

*License*

This work is licensed under a Creative Commons Attribution 4.0 International license (CC BY 4.0): <https://creativecommons.org/licenses/by/4.0>.



In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

*Digital Object Identifier*

10.4230/TGDK.4.2.0

*Bibliographic information published by the Deutsche Nationalbibliothek*

The Deutsche Nationalbibliothek lists all publications of this issue in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at <https://dnb.d-nb.de>.

*Aims and Scope*

Transactions on Graph Data and Knowledge (TGDK) is an Open Access journal that publishes original research articles and survey articles on graph-based abstractions for data and knowledge, and the techniques that such abstractions enable with respect to integration, querying, reasoning and learning. The scope of the journal thus intersects with areas such as Graph Algorithms, Graph Databases, Graph Representation Learning, Knowledge Graphs, Knowledge Representation, Linked Data and the Semantic Web. Also in-scope for the journal is research investigating graph-based abstractions of data and knowledge in the context of Data Integration, Data Science, Information Extraction, Information Retrieval, Machine Learning, Natural Language Processing, and the Web.

The journal is Open Access without fees for readers or for authors (also known as Diamond Open Access).

*Editors in Chief*

- Aidan Hogan
- Andreas Hotho
- Uli Sattler
- Kerry Taylor

*Editorial Office*

Schloss Dagstuhl – Leibniz-Zentrum für Informatik  
TGDK, Editorial Office  
Oktavie-Allee, 66687 Wadern, Germany  
[tgdk@dagstuhl.de](mailto:tgdk@dagstuhl.de)

<https://www.dagstuhl.de/tgdk>





## ■ Contents

|                 |              |
|-----------------|--------------|
| List of Authors | 0:vii–0:viii |
|-----------------|--------------|

### Preface

|                                                                                            |         |
|--------------------------------------------------------------------------------------------|---------|
| Data Management for (Knowledge) Graphs                                                     |         |
| <i>Stefania Dumbrava, George Fletcher, Matteo Lissandrini, and Riccardo Tommasini</i> .... | 1:1–1:2 |

### Data Management for (Knowledge) Graphs

|                                                                                                                                                                                                                                                    |          |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| How Human-Centric Are Our Graph Data Abstractions?                                                                                                                                                                                                 |          |
| <i>Angela Bonifati, Anastasia Dimou, Stefania Dumbrava, George Fletcher, Katja Hose, George Konstantinidis, Jose Emilio Labra Gayo, Wim Martens, Nina Pardal, Liat Peterfreund, Katherine Thornton, Maria-Esther Vidal, and Hannes Voigt</i> ..... | 2:1–2:31 |
| Transforming Shape Schemas with Composable Property-Graph Queries                                                                                                                                                                                  |          |
| <i>Philipp Seifer, Daniel Hernández, Ralf Lämmel, and Steffen Staab</i> .....                                                                                                                                                                      | 3:1–3:29 |
| Multi-Layered Legislative Knowledge Management with Property Graphs                                                                                                                                                                                |          |
| <i>Andrea Colombo, Francesco Cambria, and Francesco Invernici</i> .....                                                                                                                                                                            | 4:1–4:23 |
| Towards a Theory of Façade-X Data Access: Satisfiability of SPARQL Basic Graph Patterns                                                                                                                                                            |          |
| <i>Luigi Asprino and Enrico Daga</i> .....                                                                                                                                                                                                         | 5:1–5:46 |
| OM4OV: Leveraging Ontology Matching for Ontology Versioning                                                                                                                                                                                        |          |
| <i>Zhangcheng Qiang, Kerry Taylor, and Weiqing Wang</i> .....                                                                                                                                                                                      | 6:1–6:19 |
| Logica-TGD: Transforming Graph Databases Logically                                                                                                                                                                                                 |          |
| <i>Evgeny Skvortsov, Yilin Xia, Bertram Ludäscher, and Shawn Bowers</i> .....                                                                                                                                                                      | 7:1–7:27 |
| GeoSPARQL and SPARQL Benchmarking with GeoRDFBench Framework                                                                                                                                                                                       |          |
| <i>Theofilos Ioannidis, Nikos Mamoulis, and Manolis Koubarakis</i> .....                                                                                                                                                                           | 8:1–8:50 |
| BLINKG: A Benchmark for LLM-Integrated Knowledge Graph Generation                                                                                                                                                                                  |          |
| <i>Carla Castedo, Enrique Iglesias, Manuel Lama, Alberto Bugari n-Diz, Maria-Esther Vidal, and David Chaves-Fraga</i> .....                                                                                                                        | 9:1–9:35 |





## ■ List of Authors

Luigi Asprino  (5)  
Università Telematica San Raffaele, Rome, Italy

Angela Bonifati  (2)  
Lyon 1 University, CNRS LIRIS, France;  
IUF, Lyon, France

Shawn Bowers  (7)  
Department of Computer Science, Gonzaga  
University, Spokane, WA, USA

Alberto Bugarín-Diz  (9)  
Departamento de Electrónica e Computación,  
Universidade de Santiago de Compostela, Spain;  
Centro Singular de Investigación en Tecnoloxías  
Intelixentes (CiTIUS), Universidade de Santiago de  
Compostela, Spain

Francesco Cambria  (4)  
Dipartimento di Elettronica, Informazione e  
Bioingegneria, Politecnico di Milano, Italy

Carla Castedo  (9)  
Centro Singular de Investigación en Tecnoloxías  
Intelixentes (CiTIUS), Universidade de Santiago de  
Compostela, Spain

David Chaves-Fraga  (9)  
Departamento de Electrónica e Computación,  
Universidade de Santiago de Compostela, Spain;  
Centro Singular de Investigación en Tecnoloxías  
Intelixentes (CiTIUS), Universidade de Santiago de  
Compostela, Spain

Andrea Colombo  (4)  
Dipartimento di Elettronica, Informazione e  
Bioingegneria, Politecnico di Milano, Italy

Enrico Daga  (5)  
Knowledge Media Institute (KMI), The Open  
University, Milton Keynes, UK

Anastasia Dimou  (2)  
KU Leuven, Flanders Make@KULeuven,  
Leuven.AI, Belgium

Stefania Dumbrava  (1, 2)  
ENSIIE, SAMOVAR, Télécom SudParis, France;  
IRIF, INRIA Paris, France

George Fletcher  (1, 2)  
Eindhoven University of Technology, The  
Netherlands

Jose Emilio Labra Gayo  (2)  
WESO Lab, University of Oviedo, Spain

Daniel Hernández  (3)  
Institute for Artificial Intelligence,  
University of Stuttgart, Germany

Katja Hose  (2)  
TU Wien, Austria

Enrique Iglesias  (9)  
L3S Research Center Germany,  
Hannover, Germany

Francesco Invernici  (4)  
Dipartimento di Elettronica, Informazione e  
Bioingegneria, Politecnico di Milano, Italy

Theofilos Ioannidis  (8)  
Dpt. of Informatics and Telecommunications,  
National and Kapodistrian University of Athens,  
Greece

George Konstantinidis  (2)  
University of Southampton, UK

Manolis Koubarakis  (8)  
Dpt. of Informatics and Telecommunications,  
National and Kapodistrian University of Athens,  
Greece

Manuel Lama  (9)  
Departamento de Electrónica e Computación,  
Universidade de Santiago de Compostela, Spain;  
Centro Singular de Investigación en Tecnoloxías  
Intelixentes (CiTIUS), Universidade de Santiago de  
Compostela, Spain

Matteo Lissandrini  (1)  
University of Verona, Italy

Bertram Ludäscher  (7)  
School of Information Sciences, University of  
Illinois Urbana-Champaign, IL, USA

Ralf Lämmel  (3)  
The Software Languages Team,  
University of Koblenz, Germany

Nikos Mamoulis  (8)  
Dpt. of Computer Science and Engineering,  
University of Ioannina, Greece

Wim Martens  (2)  
University of Bayreuth, Germany

Nina Pardal  (2)  
University of Edinburgh, UK

Liat Peterfreund  (2)  
Hebrew University of Jerusalem, Israel

*Transactions on Graph Data and Knowledge*, Vol. 4, Issue 2, Article No. 0, pp. 0:vii–0:viii  
Special Issue: Data Management for (Knowledge) Graphs.

Editors: Stefania Dumbrava, George Fletcher, Matteo Lissandrini, and Riccardo Tommasini

 TGDK Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Zhangcheng Qiang  (6)  
Australian National University, Canberra, Australia

Philipp Seifer  (3)  
The Software Languages Team,  
University of Koblenz, Germany

Evgeny Skvortsov  (7)  
Google LLC, Kirkland, WA, USA

Steffen Staab  (3)  
Institute for Artificial Intelligence,  
University of Stuttgart, Germany;  
Web and Internet Science Research Group,  
University of Southampton, UK

Kerry Taylor  (6)  
Australian National University, Canberra, Australia

Katherine Thornton  (2)  
Yale University Library, New Haven, CT, USA

Riccardo Tommasini  (1)  
INSA Lyon, France;  
CNRS LIRIS, Lyon, France

Maria-Esther Vidal  (2, 9)  
Data Science Institute, Leibniz University  
Hannover, Germany;  
TIB-Leibniz Institute for Science and Technology,  
Hannover, Germany;  
L3S-Research Center, Hannover, Germany

Hannes Voigt  (2)  
Neo4j, Leipzig, Germany

Weiqing Wang  (6)  
Monash University, Melbourne, Australia

Yilin Xia  (7)  
School of Information Sciences, University of  
Illinois Urbana-Champaign, IL, USA

# Data Management for (Knowledge) Graphs

Stefania Dumbrava   

ENSIIE, SAMOVAR, Télécom SudParis, France  
IRIF, INRIA Paris, France

George Fletcher   

Eindhoven University of Technology, The Netherlands

Matteo Lissandrini   

University of Verona, Italy

Riccardo Tommasini   

INSA Lyon, France  
CNRS LIRIS, Lyon, France

---

## Abstract

In this Special Issue of Transactions on Graph Data and Knowledge, entitled “Data Management for (Knowledge) Graphs”, we present eight articles: one position paper, four research papers, and three resource papers. These span graph data query-

ing, transformation, integration, evolution, and evaluation. Their applications include legislative knowledge management, geospatial data access, and knowledge graph construction from heterogeneous sources with large language models.

**2012 ACM Subject Classification** Information systems → Graph-based database models; Information systems → Resource Description Framework (RDF); Information systems → Query languages; Computing methodologies → Knowledge representation and reasoning

**Keywords and phrases** graph data, property graphs, knowledge graphs, RDF, data management

**Digital Object Identifier** 10.4230/TGDK.4.2.1

**Category** Preface

**Acknowledgements** We thank Dagstuhl Publishing and the TGDK Advisory and Editorial Boards for their continued support, our reviewers for their timely and constructive feedback, and Aidan Hogan for his guidance in preparing this special issue.

**Published** 2026-09-03

**Editors** Stefania Dumbrava, George Fletcher, Matteo Lissandrini, and Riccardo Tommasini

**Special Issue** Data Management for (Knowledge) Graphs

## 1 Special Issue Articles

We are pleased to present the TGDK Special Issue “Data Management for (Knowledge) Graphs”, which collects eight articles: a position paper, four research papers, and three resource papers. We discuss them briefly below.

### 1.1 Position Paper

The issue opens with a position paper. In “How Human-Centric Are Our Graph Data Abstractions?”, A. Bonifati et al. argue that expressive power alone does not make an abstraction usable, and that human accessibility should be a first-class design goal alongside expressiveness. Examining graph abstractions through four dimensions and use cases from Wikidata to clinical and biological data, they set out a research roadmap toward making graph abstractions easier to study, use, and learn.



© Stefania Dumbrava, George Fletcher, Matteo Lissandrini, and Riccardo Tommasini;  
licensed under Creative Commons License CC-BY 4.0

Transactions on Graph Data and Knowledge, Vol. 4, Issue 2, Article No. 1, pp. 1:1–1:2



Transactions on Graph Data and Knowledge

TGDK Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1.2 Research Papers

“Transforming Shape Schemas with Composable Property-Graph Queries”, by P. Seifer et al., addresses schema inference for composable property graph queries, deriving the schema of a query’s output from that of its input without reference to any concrete instance. A family of mappings translates property graphs, their schemas, and their queries into RDF-based counterparts, isolating reification in a dedicated layer, which lets the authors reuse the correspondence between SHACL and description logics. They establish soundness and semantic equivalence.

“Multi-Layered Legislative Knowledge Management with Property Graphs” by A. Colombo et al. models legislative acts and their references as a property graph. Instead of storing redundant versions, a query reconstructs the text of a law as it applied on any given date, and Cypher triggers fire when a repeal removes the legal basis of active laws. An association rule operator exposes recurring overhauls of predecessor procurement legislation.

L. Asprino and E. Daga consolidate Façade-X, a uniform SPARQL access method for heterogeneous formats such as CSV, XML, and relational data. Because its meta-model captures only a fraction of RDF, not every basic graph pattern is satisfiable over a Façade-X source. “Towards a Theory of Façade-X Data Access: Satisfiability of SPARQL Basic Graph Patterns” addresses this by providing a sound, complete algorithm to statically detect unsatisfiable patterns.

In “OM4OV: Leveraging Ontology Matching for Ontology Versioning”, Z. Qiang et al. formalise a framework repurposing ontology matching systems for versioning. They implement it as Agent-OV, extending their own Agent-OM, and evaluate it on a synthetic testbed from OAEI datasets and on Brick Schema version changes. The evaluation exposes weaknesses in naive reuse, which a cross-reference mechanism addresses by narrowing candidates via prior alignments.

## 1.3 Resource Papers

“Logica-TGD: Transforming Graph Databases Logically”, by E. Skvortsov et al., expresses graph transformation as logic programming, compiling declarative rules to run over standard engines such as PostgreSQL, DuckDB, and BigQuery. It handles demanding scenarios including time-varying graphs and Win-Move games under the three-valued well-founded semantics, and is benchmarked against the Soufflé Datalog engine and SQL/PGQ on DuckPGQ.

“GeoSPARQL and SPARQL Benchmarking with GeoRDFBench Framework”, by T. Ioannidis et al., contributes reusable infrastructure for benchmarking geospatial semantic stores. They specify benchmark components declaratively as JSON, served over a configurable REST interface, and manage store lifecycles for cold-cache measurements. Stores including RDF4J, GraphDB, Stardog, Strabon, Virtuoso, and Jena GeoSPARQL are provided as reusable modules.

Closing the issue, “BLINKG: A Benchmark for LLM-Integrated Knowledge Graph Generation” by C. Castedo et al. targets the mapping step of knowledge graph construction, whose manual effort can exceed six person months. Rather than assessing opaque end-to-end systems, they evaluate large language models on generating declarative, and therefore traceable, mappings across three scenarios of graded difficulty with gold standards, reporting promising results on simpler cases and clear limitations as complexity grows.

# 2 Data Management for (Knowledge) Graphs

This Special Issue spans the querying, transformation, integration, evolution, and evaluation of graph-structured data and knowledge, from foundational perspectives to reusable artifacts. We thank the authors and reviewers whose work made it possible, and we invite the reader working with graph data to consider TGDK as a venue for their own contributions.

# How Human-Centric Are Our Graph Data Abstractions?

**Angela Bonifati** ✉ 🏠 

Lyon 1 University, CNRS LIRIS, France  
IUF, Lyon, France

**Anastasia Dimou** ✉ 🏠 

KU Leuven, Flanders Make@KULeuven, Leuven.AI, Belgium

**Stefania Dumbrava** ✉ 🏠 

ENSIIE, SAMOVAR, Télécom SudParis, France  
IRIF, INRIA Paris, France

**George Fletcher** ✉ 🏠 

Eindhoven University of Technology, The Netherlands

**Katja Hose** ✉ 🏠 

TU Wien, Austria

**George Konstantinidis** ✉ 🏠 

University of Southampton, UK

**Jose Emilio Labra Gayo** ✉ 🏠 

WESO Lab, University of Oviedo, Spain

**Wim Martens** ✉ 🏠 

University of Bayreuth, Germany

**Nina Pardal** ✉ 🏠 

University of Edinburgh, UK

**Liat Peterfreund** ✉ 🏠 

Hebrew University of Jerusalem, Israel

**Katherine Thornton** ✉ 🏠 

Yale University Library, New Haven, CT, USA

**Maria-Esther Vidal** ✉ 🏠 

Data Science Institute, Leibniz University Hannover, Germany  
TIB-Leibniz Institute for Science and Technology, Hannover, Germany  
L3S-Research Center, Hannover, Germany

**Hannes Voigt** ✉ 🏠 

Neo4j, Leipzig, Germany

---

## Abstract

Graph data abstractions are often assumed to be intuitive, but experience shows that they are not equally understandable or usable in practice. In this vision and challenges paper, we examine the human-centricity of contemporary graph data abstractions through four lenses: researchability, usability, teachability, and societal impact. Drawing on diverse real-world use cases, ranging from clinical data and

collaborative knowledge bases to biological and pan-genomic graphs, we distill insights from database research, human-computer interaction, and education. Based on this analysis, we identify open research challenges that must be addressed to make graph abstractions easier to study, use, learn, and reason about.



© Angela Bonifati, Anastasia Dimou, Stefania Dumbrava, George Fletcher, Katja Hose, George Konstantinidis, Jose Emilio Labra Gayo, Wim Martens, Nina Pardal, Liat Peterfreund, Katherine Thornton, Maria-Esther Vidal, and Hannes Voigt;  
licensed under Creative Commons License CC-BY 4.0

*Transactions on Graph Data and Knowledge*, Vol. 4, Issue 2, Article No. 2, pp. 2:1–2:31



Transactions on Graph Data and Knowledge

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

**2012 ACM Subject Classification** Information systems → Graph-based database models; Information systems → Resource Description Framework (RDF); Human-centered computing → Human computer interaction (HCI); Social and professional topics → Computing education; Information systems → Query languages; Computing methodologies → Knowledge representation and reasoning

**Keywords and phrases** graph data abstractions, property graphs, RDF, human-centric data management, usability, computing education

**Digital Object Identifier** 10.4230/TGDK.4.2.2

**Category** Position

**Funding** *Angela Bonifati*: Partially supported by the ANR Verigraph (ANR-21-CE48-0015) project.

*Stefania Dumbrava*: Partially supported by the ANR JCJC VERDI (ANR-24-CE25-1109) project.

*Katja Hose*: Partially supported by the Vienna Science and Technology Fund (WWTF) [DORSET, grant id: 10.47379/ICT25032].

*Jose Emilio Labra Gajo*: Partially funded by projects SHAKIGRA: Shaping Knowledge and Interoperable Graphs, code: PID2024-157010OBI00, and project SEK25-GRU-GIC-24-089.

*Nina Pardal*: Partially funded by DFG grant VI 1045-1/1, and Huawei Strategic Talent Program.

*Liat Peterfreund*: Supported by Israel Science Foundation 2355/24.

*Maria-Esther Vidal*: Partially funded by Leibniz Association, program “Leibniz Best Minds: Programme for Women Professors”, project TrustKG-Transforming Data in Trustable Insights (Grant P99/2020), and by the Lower Saxony Ministry of Science and Culture (MWK) with funds from the Volkswagen Foundation’s zukunft.niedersachsen program (CAIMed – Lower Saxony Center for AI and Causal Methods in Medicine; GA No. ZN4257).

**Acknowledgements** This work builds on discussions and outcomes from the Dagstuhl Seminar 24102 *Shapes in Graph Data: Theory and Implementation* [2].

**Received** 2026-02-27 **Accepted** 2026-06-16 **Published** 2026-09-03

**Editors** Stefania Dumbrava, George Fletcher, Matteo Lissandrini, and Riccardo Tommasini

**Special Issue** Data Management for (Knowledge) Graphs

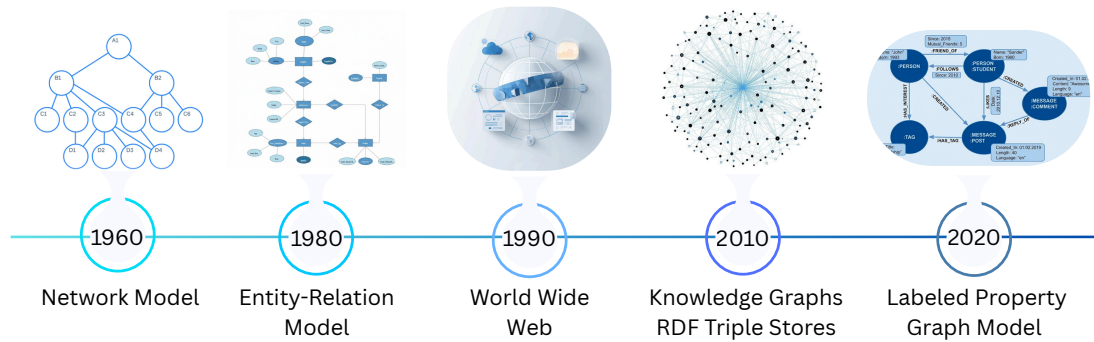
**Generative AI Declaration** During the preparation of this work, the authors only used generative AI assistants for language and style improvement, for support in structuring parts of the text, and for polishing visual elements (Canva AI). No references were generated by these tools. All AI-assisted output was reviewed and edited by the authors, who take full responsibility for the content of this article.

## 1 Introduction

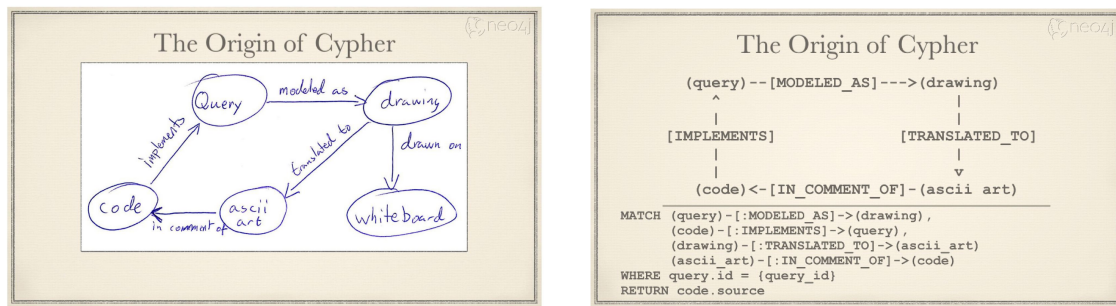
Graph data abstractions have emerged as a central paradigm for representing and querying interconnected data. From ER diagrams [34] and Sowa’s conceptual graphs [92] to data models such as XML [28], RDF [57] and property graphs [6, 82], and their associated query languages, graph abstractions are now widely used across domains, including data integration, collaborative knowledge bases, and scientific data management [7, 83]. Their appeal is often attributed to the explicit representation of relationships, which supports flexible schema evolution and arguably offers more intuition and easier constructs than relational or hierarchical models.

Figure 1 provides a high-level historical perspective on the evolution of graph data abstractions. Successive abstractions have accumulated in this historical progression, resulting in a complex and diverse ecosystem of coexisting graph data models and languages. Practical examples of graph abstractions have often evolved in close interaction with human needs, practices, and communities. Wikidata [98] is such a bottom-up initiative that has emerged while inherently accounting for human-centric aspects of graph data representations such as usability, learnability, and explainability. Cypher offers a particularly instructive example.





**Figure 1** Historical evolution of graph data abstractions: from the CODASYL Network Data Model (NDL), introducing explicit navigational, pointer-based record graphs; through Entity-Relationship (ER) modeling, elevating graphs to a high-level conceptual design formalism; to the World Wide Web, which operationalized a global, large-scale hyperlink graph over semi-structured documents; followed by Knowledge Graphs, emphasizing semantic integration and ontology-driven interoperability; and culminating in the Labeled Property Graph model and ongoing standardization efforts (e.g., GQL), marking the consolidation of expressive, interoperable, and queryable graph data systems.



**(a)** Conceptual origin whiteboard drawing.

**(b)** 2D ASCII-art comment (top) and Cypher ASCII-art patterns syntax (bottom).

**Figure 2** Evolution of Cypher from informal sketches to declarative pattern matching (from [60]).

► **Example 1** (The Human-Centric Design of Cypher). The origins of the Cypher graph query language illustrate how graph data abstractions can emerge in a human-centric, bottom-up manner. As documented in its historical evolution [60], Cypher was shaped not by a pre-existing formal model, but by observing how practitioners interacted with graph data in early applications. Users reasoned about traversing chains of relationships and interesting patterns of connected entities on whiteboard or scratchpad drawings (cf. Figure 2a) and documented them in 2D ASCII-art comments next to their imperative traversal code (cf. top of Figure 2b). Cypher ported this practice to a query language. It introduced pattern-matching constructs with an ASCII-art syntax [69], as well as labeled nodes and relationships that are closely aligned with the graph mental models of users (cf. bottom of Figure 2b). These features abstracted recurring usage patterns into language constructs, reducing the gap between the intended query formulation and its execution. From a human-centric perspective, Cypher’s design prioritizes interpretability and learnability alongside expressiveness, enabling users with limited formal training to interact with graph data [66, 69].

Early approaches in the neighboring discipline of relational databases had explicitly treated usability as a first-class property of a data abstraction, even at a system-design level, alongside formal semantics and query evaluation. In this area, human-centric aspects have been studied in a principled manner. Seminal empirical studies of query languages such as SQUARE, SEQUEL, and Query-by-Example [30, 112], and more recently of Relational Diagrams [42], systematically examined how users understood, learned, and applied these abstractions in practice, grounding language design in human factors rather than purely formal considerations [80, 95]. These studies established a tradition in which abstraction quality was evaluated not only by expressiveness or efficiency but also by accessibility.

In graph data management approaches, these aspects have not always been first-class design principles nor have they been scrutinized by the research community in the same principled way.

Thus, human-facing properties of graph abstractions, e.g., their interpretability, usability, learnability, and the way they differentiate user roles, remain unevenly understood. Empirical studies in graph data visualization and exploration show substantial variation in how users interpret graph structure, semantics, and query results, particularly across different levels of technical expertise [4, 11, 19, 20, 59, 61, 110]. Research in education and human-computer interaction further suggests that graph-based representations pose distinct cognitive challenges compared to tabular or tree-based abstractions, affecting users' ability to form accurate mental models [21, 48, 108]. Crucially, empirical evidence shows that the ability to successfully execute graph queries or complete interaction tasks does not necessarily imply a sound conceptual understanding of the underlying data abstraction [21, 71].

These limitations increasingly matter in practice. Community-maintained knowledge bases such as Wikidata rely on graph schemas, constraints, and validation mechanisms to coordinate large and heterogeneous contributor populations [51, 63, 78, 98]. Scientific applications in domains such as biology, ecology, and genomics use graph abstractions to integrate heterogeneous datasets and support complex analytical workflows [29, 36]. Recently, graph-structured data has been used with large language models (LLMs) and AI systems to support downstream reasoning and interaction, widening the audience of graph abstractions to include non-expert users and automated agents [72]. Given the increasing synergy between graph abstractions and the booming AI landscape, for tasks ranging from data annotation and model training to supporting human-in-the-loop explainability and decision-making, aligning the human mental model with graph abstractions becomes crucial. In these settings, shortcomings in the learnability, interpretability, or usability of graph abstractions directly affect data quality, system reliability, and downstream behavior.

## Vision and Contributions

We use the term *graph data abstraction* to refer to the conceptualization of data as a graph, where nodes represent entities and edges represent relationships. In such abstractions, information emerges not only from individual data elements, but also from the structure of their interconnections, enabling the explicit representation of patterns, dependencies, and contextual relationships within the data. We draw the boundary deliberately: an abstraction is *graph-based* in our sense when relationships are first-class, independently addressable modeling primitives that may themselves carry identity, type, or attributes. This includes RDF and property graphs and excludes models in which inter-entity links are implicit or value-derived, e.g., the relational model, where a foreign key is a value-based join rather than a first-class edge.

Graph abstractions are increasingly adopted because certain data and tasks, e.g., densely interlinked entities, multi-hop reasoning over relationships, and schema-flexible integration of heterogeneous sources, are represented more faithfully and queried more naturally as graphs.

Yet, the expressive power of graph abstractions does not automatically translate into human accessibility or understanding. Because such abstractions are *already* widely deployed across collaborative, scientific, and AI-mediated settings, the question we pursue is not whether to adopt them, but how to make them human-accessible to the diverse audiences that now encounter them.

Human-centricity is not the only lens through which graph abstractions can be evaluated. Such abstractions also serve machine-oriented purposes, including automated reasoning, recommendation, and learning, where considerations such as expressiveness and performance are central. We view human-facing properties as complementary to these concerns rather than as a replacement for them. Our focus on human accessibility reflects the observation that graph abstractions are increasingly authored, queried, taught, and relied upon by people, and that these human-facing aspects have received comparatively less systematic attention.

We argue that graph abstractions should be designed not only to represent interconnected data, but also to treat human-accessible aspects as first-class design principles, supporting how users observe, navigate, interpret, manipulate, and reason about such interconnections.

We use the term *human-centric graph data abstractions* for graph abstractions that treat human-facing properties as first-class design and evaluation concerns. This specializes to graphs a broader human-data interaction agenda, which has produced consolidated design recommendations [97]. We identify these as *interpretability* (users can understand what the abstraction represents and why a result was produced), *usability* (users can perform intended tasks efficiently and with few errors), *learnability* (newcomers can acquire competence with the abstraction), *role differentiation* (distinct roles, such as schema designers, query writers, analysts, consumers, and indirectly affected parties, interact with the same abstraction with different goals, expertise, and responsibilities, and the abstraction accommodates them), and *societal impact* (consequences for people who are not direct users are made visible and accountable). We examine these properties through the four analysis lenses used consistently throughout the paper: researchability, usability, teachability, and societal impact. In these abstractions, the relational and topological properties of graph representations are not merely expressive features of the data model, but cognitively accessible structures that users can effectively exploit for reasoning, interaction, and decision-making.

We examine real-world graph-based use cases from clinical data management, collaborative knowledge bases, and biological and pangenomic analysis to surface recurring human-facing tensions, particularly around usability and interpretability, that persist across domains and abstraction paradigms (Section 2). As our main contribution, we distill these tensions into research challenges, spanning institutional, methodological, operational, and applicative concerns. These challenges are grounded in prior results from database theory, human-computer interaction, and education (Section 3) and we refine them into a concrete roadmap of theory- and application-driven research questions (Section 4).

## Methodology

This paper is a *vision and challenges contribution*. We synthesize observations from diverse use cases and prior empirical work to delineate an agenda for human-centric graph abstraction research. Our approach proceeds in three stages.

First, we selected use cases covering two distinct paradigms of graph data abstractions: open, collaborative, and community-driven graph construction (Wikidata, Section 2.1), and expert-curated, integration-heavy scientific graph data management from clinical and biological domains (Section 2.2). Several co-authors are directly involved in these projects, providing first-hand knowledge of their human-facing challenges.

Second, for each use case, we synthesized published empirical findings, e.g., controlled user studies [71], system evaluations [76], and community surveys [78, 81, 85], along our four human-centric dimensions: researchability, usability, teachability, and societal impact (Section 4).

## 2:6 How Human-Centric Are Our Graph Data Abstractions?

Third, recurring tensions observed across two or more use cases were elevated to research challenges (Section 3), organized by institutional, methodological, operational, and applicative categories; these are subsequently refined into actionable research questions in Section 4.

To support traceability, each challenge in Section 3 is annotated with the specific use case(s) that motivated it, and Table 1 consolidates these into the connective tissue of the paper. Reading it left to right: the institutional, methodological, operational, and applicative *challenges* of this section (IQ/MQ/OQ/AQ) are refined into the researchability, usability, teachability, and societal *roadmap* dimensions of Section 4; each dimension concerns a specific human *subject*, i.e., researcher, direct user, learner, or indirect user, and is pursued through theory-driven and application-driven research questions (RQ/UQ/TQ/SQ).

■ **Table 1** Overview of research challenges, roadmap, concerned human subjects, and the main underlying theory-driven and application-driven research questions.

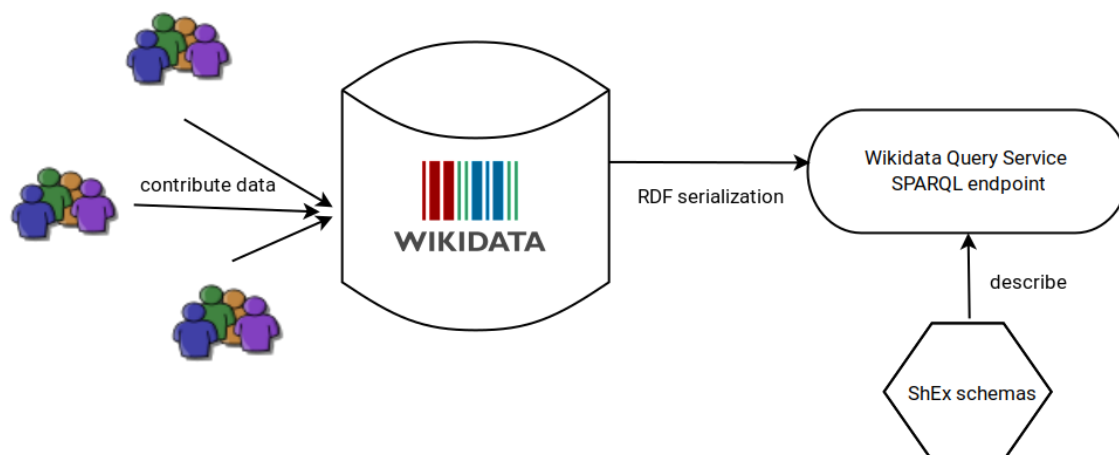
|                               | Institutional<br>(Sec. 3.1)          | Methodological<br>(Sec. 3.2)                                | Operational<br>(Sec. 3.3)                                                           | Applicative<br>(Sec. 3.4)                                                     |
|-------------------------------|--------------------------------------|-------------------------------------------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------|
| <b>Challenges</b><br>(Sec. 3) | IQ1, IQ2                             | MQ1, MQ2, MQ3                                               | OQ1, OQ2, OQ3                                                                       | AQ1, AQ2                                                                      |
| <b>Roadmap</b><br>(Sec. 4)    | <b>Researchability</b><br>(Sec. 4.1) | <b>Usability</b><br>(Sec. 4.2)                              | <b>Teachability</b><br>(Sec. 4.3)                                                   | <b>Societal</b><br>(Sec. 4.4)                                                 |
| <b>Subject</b>                | <b>Researcher</b>                    | <b>Direct User</b><br>(data modeler, query<br>writer, etc.) | <b>Learner</b><br>(person learning to<br>become a<br>practitioner or<br>researcher) | <b>Indirect User</b><br>(person affected by<br>use of database<br>technology) |
| <b>Theory</b>                 | RQ1                                  | UQ1, UQ2, UQ3                                               | TQ1, TQ2                                                                            | SQ1, SQ2                                                                      |
| <b>Application</b>            | RQ2                                  | UQ4, UQ5                                                    | TQ3                                                                                 | SQ3, SQ4                                                                      |

## 2 Motivational Use Cases

We present a set of use cases that illustrate how graph data abstractions are designed, learned, and used across different domains, including collaborative knowledge bases, as well as clinical and biological data management. These applications show that graph abstractions can successfully support complex tasks and serve diverse communities. At the same time, they reveal recurring difficulties related to abstraction understanding, modeling choices, and role differentiation. In several settings, users rely on documentation, peer support, or specialized tools to work effectively with these abstractions. While such practices demonstrate that graph abstractions can be adopted successfully in practice, they do not yet amount to a systematic understanding of how these abstractions align with users' mental models across roles and domains.

### 2.1 Collaborative Graph Abstractions in Wikidata

Wikidata is a collaboratively curated knowledge base of structured data with an active editor community [98]. Data can be accessed through the Wikidata Query Service [107] as well as through public API endpoints [105]. As a cross-domain knowledge graph, Wikidata enables unrestricted reuse of its data under the Creative Commons 0 (CC0) license [106].



■ **Figure 3** Wikidata editors contribute data to the knowledge base and contribute ShEx schemas to encode data models and support validation of entity data.

We focus on Shape Expressions (ShEx) schemas here deliberately: they are the explicit, human-authored abstraction layer that the Wikidata community *co-creates and consults*, which makes them the natural object of our human-centric analysis.

In May 2019, the Wikidata development team announced a new namespace in Wikidata, called EntitySchema, whose entries carry the prefix E (analogous to Q for items and P for properties) and store schemas encoded in the Shape Expressions (ShEx) language. The ShEx language is a formal language for data modeling and validation of RDF graphs [24, 93]. ShEx syntax adopts tokens from Turtle and SPARQL, and its semantics are inspired by XML Schema and RELAX NG [55]. ShEx has been successfully used for RDF validation in the FHIR community for exchanging medical records [91]. Wikidata editors encode data models as schemas, exemplifying community co-creation of practical graph abstractions. Other editors consult these schemas to understand how data is modeled by editors, thereby using these abstractions for clarity about sub-sections of the Wikidata graph. In addition to encoding data models, Wikidata editors use ShEx schemas to validate entity data from the knowledge base. Community members use Wikidata’s schema namespace to publish schemas for validating data conformance [24, 67]. The successful use of ShEx schemas for building consensus around the alignment of biodiversity data with Wikidata is discussed in [100]. ShEx validation has shown promise in increasing data quality by addressing low-level modeling inconsistencies in Wikidata [78].

Researchers have also developed several different applications to extract schemas from Wikidata [23, 101]. These tools suggest a schema based on an analysis of how data for similar items have been modeled. Editors may choose to further refine the suggested schema or reuse it.

Considering the aspect of **usability**, the fact that several contributors have authored schemas after consulting documentation or example schemas suggests that ShEx authoring is usable enough for task completion within the community context. No systematic usability study has yet evaluated whether it is also efficient and low-error.

Considering the aspect of **teachability**, educators use Wikidata as a learning platform [90]. The Wikidata community has created resources for learning how to write schemas and how to use schemas to validate data. Researchers observed that members of the Wikidata community without previous experience with the SPARQL query language taught themselves SPARQL through interacting with Wikidata’s SPARQL endpoint [63]. Members of the Wikidata community learn ShEx to author schemas. Community organizers have hosted interactive sessions where people

who are already familiar with ShEx demonstrate how to use ShEx schemas within Wikidata, such as Data Quality Days 2021 [102], Data Quality Days 2022 [103], and Data Modeling Days 2023 [104]. Following these events, contributors who had not previously engaged with the schema namespace began editing existing schemas and creating new ones, demonstrating how the Wikidata community facilitates peer-to-peer learning and skill development.

While these developments demonstrate the dynamism of the Wikidata community, they also point to a barrier to entry for newcomers. The very evidence that ShEx schemas are learnable here also shows how that competence is acquired: through documentation, example schemas, onboarding events, mentorship, and tooling, rather than because the schemas are self-explanatory. Familiarity with the schema language and the community’s modeling conventions thus remains a hurdle. Since no systematic usability study has yet been conducted, the success of schema adoption is better read as a sign of strong community support.

Due to the fact that members of the Wikidata community publish their schemas on-wiki using the CC0 license, researchers have access to the data in the knowledge base as well as the related human-defined schemas. This combination of openly licensed data and community-authored schemas provides a rare opportunity to empirically study how human-defined abstractions evolve in large-scale collaborative environments.

People from diverse backgrounds contribute data to Wikidata [53, 74, 81]. Editors range from highly active, technically skilled power editors to occasional contributors with limited technical background [85]. This blend of contributors makes Wikidata a useful community for investigating differences in user affinity toward existing graph abstractions. Wikidata is also a cross-domain knowledge base [46]. The fact that such a wide range of conceptual areas are represented in Wikidata allows researchers to examine how well existing graph abstractions support mainstream use cases across domains by comparing data with community-authored schemas.

While Wikidata illustrates that graph abstractions can be adopted and extended in large-scale collaborative settings, this success leverages sustained community coordination, self-directed learning, and the efforts of technically skilled contributors. Schema design, validation logic, and modeling conventions are not uniformly accessible to all participants, and abstraction complexity remains unevenly distributed across roles. These observations raise recurring questions about user affinity, roles, and accountability in collaborative graph systems.

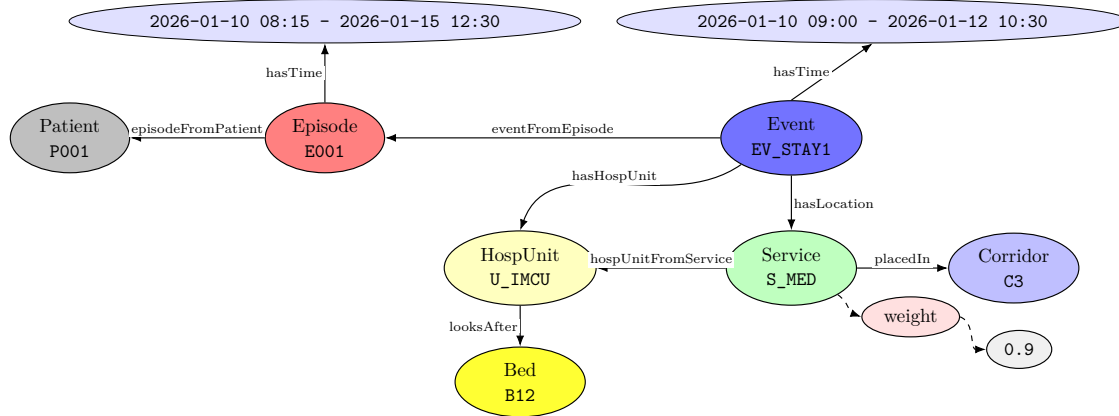
Beyond direct contributors, Wikidata increasingly serves as a backend for AI-mediated systems, introducing a new category of *indirect users* who interact with graph abstractions not through query interfaces but through outputs generated by LLMs. This is relevant to human-centricity because indirect users inherit whatever biases, coverage gaps, or modeling limitations the graph abstraction imposes, without visibility into those constraints.

## 2.2 Graph Abstractions for Clinical and Biological Data

Clinical and biological domains provide a compelling context for studying the human-centric properties of graph abstractions due to their highly interconnected, heterogeneous, and evolving nature, as well as diverse user roles. The following use-cases highlight how graph abstractions support analytical tasks, while exposing recurrent challenges in usability and learnability.

### 2.2.1 MIMIC-III and Epidemiological Nosocomial Infections

Nosocomial infections represent a major challenge for epidemiological analysis, as their emergence and propagation depend on complex interactions between patients, hospital spaces, and time-dependent clinical events. Tracing potential transmission pathways requires reconstructing who was where, when, and under which clinical conditions, often across heterogeneous data sources



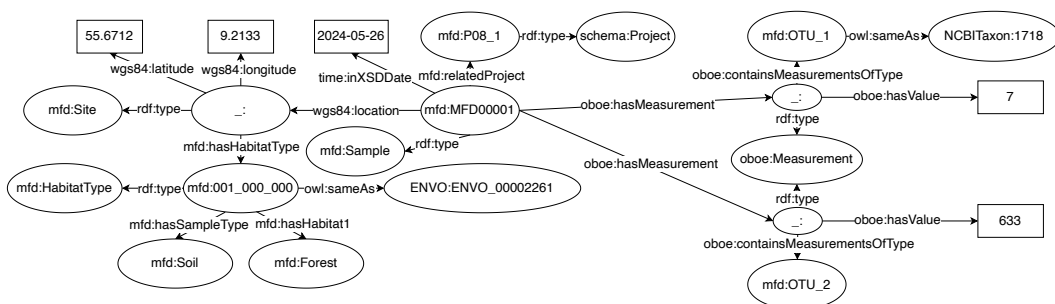
■ **Figure 4** Instance graph (single patient, single episode, single event) illustrating the integration of temporal and spatial dimensions in the hospitalization model. Node colors distinguish different entity types (patient, episode, event, spatial locations, temporal literals, relation attributes, and numeric values). To enhance readability, the corresponding class-level representations are intentionally omitted, and only instance-level elements are depicted in this visualization.

and fine-grained spatial layouts. This setting demands expressive modeling approaches capable of jointly capturing temporal dynamics, spatial hierarchies, and clinical context, while remaining amenable to analysis, querying, and visualization.

Graph abstractions were employed to model and analyze patients' movements during their hospital stays in the context of spatio-temporal epidemiological research [76]. The use case captures the interplay between temporal evolution and hospital space to support the detection and analysis of nosocomial infection patterns. The model integrates two interconnected dimensions. The *spatial dimension* describes the hospital environment as a structured hierarchy of locations, ranging from beds to rooms, corridors, areas, floors, and buildings. In addition to hierarchical containment, spatial adjacency relations (e.g., *next to*, *opposite*) capture proximity between locations at the same level, enabling fine-grained reasoning about potential infection transmission paths. The spatial model further distinguishes between the *physical space* (locations) and the *logical space* (hospitalization units and clinical services responsible for patient care). The *temporal dimension* models each patient's hospitalization as a sequence of episodes, where each episode corresponds to a hospital stay composed of time-stamped events. Events represent movements (e.g., room transfers), medical procedures, or diagnostic outcomes such as microbiological test results. Each episode and event is associated with a start and end time, enabling the representation of both instantaneous and interval-based occurrences. Events linked to locations constitute the integration point between temporal and spatial dimensions. Figure 4 illustrates a fragment of this model as a directed edge-labeled graph at the instance level.

Various graph data abstractions, including RDF, property graphs, and RDF-star, were evaluated for clinical epidemiological tasks [76]. Data from MIMIC-III [52] encompassed inpatient movements, clinical tests, and epidemiological outcomes, including patient stays, ward transitions, services, and microbiological test results. To complete the domain model, synthetic data were generated to replicate hospital layouts, categorizing beds into surgical, medical, mixed, and neonatal units. The resulting graphs exhibited increasing event sets and varying structural densities while preserving consistent layouts and service configurations. Three epidemiological tasks were modeled: (i) detecting infection outbreaks, (ii) tracking outbreak propagation, and (iii) monitoring potentially infectious locations. All tasks were successfully supported across the considered graph abstractions.





■ **Figure 5** Microflora Danica RDF Graph.

Following the user taxonomy proposed by Li et al. [59], stakeholders were categorized into three groups: *knowledge builders* (experts in graph construction, data modeling, and query optimization), *analysts* (proficient in graph data models and visualization techniques), and *consumers* (infectious disease researchers and healthcare professionals). Knowledge builders required efficient query processing mechanisms for traversing graphs and computing adjacency, reachability, and shortest paths. Analysts needed expressive data models that support visualization of queries and results, while consumers primarily sought accurate outputs and interpretable visual representations for clinical decision-making.

From a **usability** perspective, recent work has reported [76] that Cypher queries over property graphs were more concise and easier to interpret visually than the equivalent SPARQL formulations over the RDF encodings. Consumers understood high-level graphical representations and the notion of traversing relationships to answer epidemiological questions; however, the underlying graph abstractions (schemas, encodings, and formal query formulations) remained inaccessible without technical mediation.

From a **teachability** perspective, modeling statements about relationships (particularly when edges required attributes or when n-ary relations were involved) has proven challenging. RDF’s lack of native support for such constructs requires reification, i.e., transforming relationships into first-class nodes with associated properties. While cognitively demanding, reification enables expressive modeling of annotated relations. RDF-star provides a more direct mechanism for expressing statements about statements, but its nested syntax is still difficult for newcomers [7, 55].

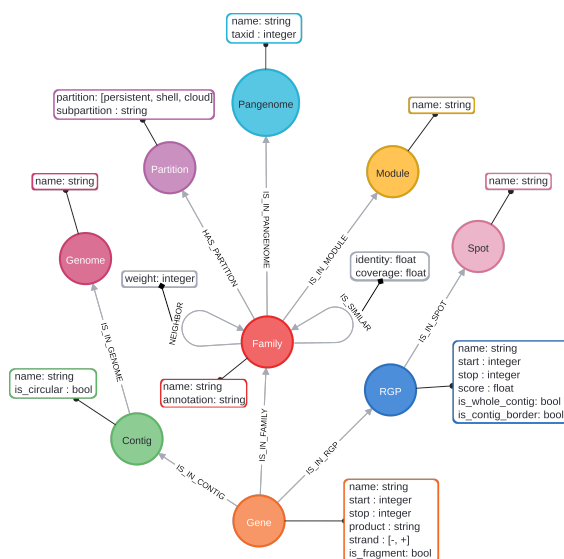
From the perspective of **knowledge builders**, the evaluation revealed that graph management systems supporting different abstractions exhibited distinct memory management strategies and indexing mechanisms. Query complexity affected execution performance differently across implementations, highlighting that abstraction choices have both conceptual and measurable system-level implications.

## 2.2.2 Microflora Danica

Life sciences constitute an active research area in which large volumes of data are shared with the community. While some initiatives publish data natively as ontologies or RDF, such as Bio2RDF [29], the majority of biological datasets originate from empirical studies and are shared in tabular formats, such as spreadsheets or CSV files.

Microflora Danica integrates large-scale microbiological data with external environmental datasets [36]. The core dataset results from a nationwide field study in Denmark and includes DNA sequences and associated metadata such as location, time, and sampling processes. Identified microorganisms are linked to established ontologies and taxonomies, and the data is integrated with external datasets describing soil properties, such as EcoDes-DK15 [13].





■ **Figure 6** Graph schema structure of the pangenome dataset.

This use case illustrates a recurring human-centric friction rather than a **usability** success. Although the data is published as an RDF graph (Figure 5), most bioscience workflows, including modern machine-learning pipelines, remain optimized for tabular data. As a result, researchers outside computer science often struggle to adopt the graph abstraction directly: despite its integration benefits, the representation raises rather than lowers the barrier to entry for its intended scientific users.

From the perspective of **teachability**, the dataset offers opportunities for teaching SPARQL, spatio-temporal graph querying, and graph-based machine learning. The data supports OLAP-style analyses over graphs, such as identifying geographic regions with high microbial diversity. Although these opportunities are not yet fully exploited, the use case highlights the pedagogical potential of graph abstractions in the life sciences.

### 2.2.3 Pangenomic Graphs

In genomics, the rapid growth of sequencing projects has resulted in hundreds of thousands of available genomes. Comparative genomics increasingly relies on large-scale analyses of pangenomes to study gene presence, absence, and variation across species. Graphs provide a natural abstraction for capturing the complex relationships between multiple pangenomes and for analyzing evolutionary mechanisms, such as the transmission of antimicrobial resistance.

Recent work [12] has proposed representing multi-pangenomic data as property graphs. One such dataset, PanGraphDB, integrates data produced using the panRGP method and the PPanG-GOLiN framework into a unified property graph stored in Neo4j, whose schema is given in Figure 6. This representation allows domain experts to efficiently express and execute complex genomic analyses in Cypher, including cross-species queries that were previously difficult.

Regarding **teachability**, this use case was the subject of a controlled student study [71]. The study aimed to assess how effectively students with varying backgrounds could discover the schema of a complex graph dataset and formulate graph queries of increasing complexity. After a six-hour session consisting of an intensive introduction to graph databases followed by hands-on tasks, participants were able to write basic queries. However, errors primarily occurred due to misunderstandings of semantic language constructs and difficulties in distinguishing between schema and data, highlighting challenges in abstraction comprehension rather than syntax.

### 3 Research Challenges

Across domains, graph abstractions succeed technically yet reveal recurring human-centric frictions. As Section 2 shows, these frictions are not isolated usability issues but structural consequences of how abstractions are formalized, taught, governed, and deployed. That is, graph data abstractions are socio-technical constructs shaped by institutional contexts, methodological choices, user roles, and application scenarios. In Wikidata (Section 2.1), abstraction design and validation are embedded in community governance and peer learning. In clinical epidemiology (Section 2.2.1), abstraction choices affect interpretability and role differentiation. In biological and pangenomic settings (Sections 2.2.2 and 2.2.3), abstraction expressiveness interacts with tooling ecosystems and learners’ conceptual understanding. These tensions motivate the research challenges below.

Each challenge below follows a uniform *Why/What/How* structure: *Why* states the human-centric tension and which use case(s) from Section 2 motivate it; *What* surveys the relevant state of the art; and *How* outlines potential *research directions*. Concrete, prioritized research questions derived from these challenges are deferred to the roadmap in Section 4.

#### 3.1 Institutional Challenges

Institutional challenges relate to educational structures, curriculum guidelines, incentive systems, and evaluation criteria that shape which research questions are prioritized and which skills are cultivated in future practitioners. We discuss these as follows.

##### IQ1: Incorporating human-centric studies into our curricula

**Why.** Advancing human-centric graph abstractions requires early and sustained exposure to human-centered thinking within educational pathways. However, current curricula, particularly in the context of the rapid expansion of AI and data-driven technologies, tend to emphasize technical proficiency, algorithmic performance, and system scalability, often at the expense of human factors such as usability, interpretability, and social impact. Without curricular redesign, future practitioners may lack the tools to reason about graph abstractions as socio-technical rather than purely technical artifacts.

**What.** To address this gap, human-centric methodologies must be systematically embedded into graph-related curricula across educational levels, from introductory courses to advanced graduate programs. This includes exposing students to user-centered design principles, qualitative and mixed-methods evaluation, and interdisciplinary perspectives that complement formal modeling and algorithmic techniques. Existing work in education and data abstraction has shown that challenge-based and visually grounded learning approaches can improve conceptual understanding and transferability across domains [5, 50, 62].

**How.** Implementing these changes requires coordinated curricular and institutional efforts. Courses and programs should be co-designed and, where possible, co-taught with experts from socio-humanistic disciplines, education research, and HCI to ensure meaningful integration rather than superficial coverage. Project-based and challenge-based learning should be adopted at multiple stages of training, allowing students to engage with graph abstractions through real-world, interdisciplinary problems. At the institutional level, sustained collaboration between data researchers, mathematics educators, and computational pedagogy experts is needed to develop reusable teaching materials, assessment strategies, and learning outcomes. Finally, articulating clear educational value propositions and business cases can help engage academic leadership and accreditation bodies, ensuring that human-centric competencies are supported.

## **IQ2: Institutional and disciplinary barriers to centering human perspectives**

**Why.** Institutional and disciplinary structures can make it difficult to systematically integrate human perspectives into socio-technical research, including user-centric data management. These barriers arise from entrenched funding priorities, evaluation criteria, and disciplinary cultures, particularly in computer science, where scalability, performance, and formal correctness are traditionally emphasized, while user studies and qualitative methods receive comparatively less recognition. As a result, researchers adopting human-centered approaches may face limited visibility, fewer publication opportunities, and constrained career incentives.

**What.** Prior socio-technical discussions in the data management and computing communities have highlighted the lack of widely recognized venues and evaluation frameworks that explicitly value multidisciplinary contributions. Metrics such as ergonomics, learnability, interpretability, and user satisfaction are rarely treated as first-class research outcomes, despite their central role in the design and adoption of data systems. This imbalance reinforces disciplinary silos and discourages sustained engagement with human-centric research questions.

**How.** Addressing these barriers requires coordinated institutional efforts. These include broadening review and evaluation criteria to explicitly recognize human-centered contributions, fostering publication venues and tracks that welcome mixed qualitative–quantitative methodologies, and encouraging collaboration across data management, HCI, social sciences, and education. By aligning incentives with the goals of human-centric data management, institutions can better support research that integrates technical rigor with meaningful engagement with human needs.

## **3.2 Methodological Challenges**

Methodological challenges address how graph abstractions are studied and evaluated. They highlight the need to borrow, adapt, and integrate methods from disciplines such as HCI, cognitive science, and visualization in order to better understand users’ mental models, interaction strategies, and sources of error.

### **MQ1: Learning from how other disciplines use graph abstractions**

**Why.** Disciplines such as biology, the social sciences, network analysis, and data science rely heavily on graph abstractions for problem solving and data representation [37, 83]. Studying how practitioners in these fields actually build, query, and reason over graphs reveals the cognitive demands graph abstractions place on real users, grounding design in established practice rather than the assumptions of a single community.

**What.** These communities have developed mature, domain-specific conventions for modeling and analyzing graph data [37]. Examining them clarifies which abstraction features support or hinder domain reasoning and which conventions recur across fields, providing an externally validated basis for graph-abstraction design.

**How.** Future work should systematically survey graph practices across data-intensive disciplines and distill the recurring modeling patterns, tool choices, and points of friction into transferable design guidance, identifying common human-facing requirements.

### **MQ2: Adapting empirical methods from HCI and cognitive science**

**Why.** Understanding how users perceive, interpret, and struggle with graph abstractions requires empirical study, yet graph-data research rarely employs the methods that HCI and cognitive science have established for exactly this purpose; bringing cognitive-science methods into data management has been argued to be a productive and underused direction [17, 21, 108].

## 2:14 How Human-Centric Are Our Graph Data Abstractions?

**What.** These fields offer a mature toolkit, e.g., surveys, structured and semi-structured interviews, Grounded Theory, sketching, think-aloud protocols, eye-tracking, and mixed-methods usability evaluation, for eliciting users' mental models and locating sources of friction. Bigelow et al. [21] and Williams et al. [108] use Grounded Theory and sketching to show how personal and professional context shapes the data abstractions users form; Li et al. [59] and Lissandrini et al. [61] document domain-specific usability and visualization challenges for knowledge-graph practitioners; mixed-methods usability evaluations have been applied to knowledge-graph-based systems [58]; eye-movement studies of visual versus textual representations [5] quantify differences in how users read each; and abstraction ability itself can be treated as a learnable, assessable competence rather than an assumed one [54].

**How.** Adapting these methods to graph abstractions means using surveys and interviews to surface mental models, expectations, and pain points; applying Grounded Theory to interaction traces to expose the social and cognitive factors behind usability problems; pairing think-aloud and eye-tracking with statistical analysis to evaluate behavior across user groups; and conducting mixed-methods usability evaluations of graph systems in the spirit of [58].

### MQ3: Applying HCI interaction- and visualization-design principles

**Why.** Beyond methods for *studying* users, HCI and visualization research offer principles for *building* abstractions that match how people think. Iterative testing, support for exploration, attention to the diversity of users' mental models, and theory-grounded visual design can make graph abstractions and their interfaces more comprehensible and usable [19, 20, 62, 68, 109].

**What.** Evidence points to concrete design levers. Ma et al. [62] show that grounding the visual abstraction of property graph queries in established principles produces more effective and appealing representations. Chanthatrojwong et al. [32] extend this principle to property graph schemas, showing that composite, theory-grounded visual encodings can make schema features such as inheritance, constraints, and label expressions more comprehensible. Nouraei et al. [68] stress supporting different user types and balancing productivity with exploration. Williams et al. [108] show that divergent mental models call for multiple, customizable representations of the same data. Xie et al. [109] quantify the gap between users' and designers' mental models, establishing that it directly affects task performance.

**How.** These principles translate into adaptive, customizable visualizations, schema-visualization interfaces such as PASCAL [33], role- and goal-aware knowledge-graph exploration interfaces [96], and iterative, user-tested design cycles. Mental model elicitation techniques such as path diagrams [109] can align abstractions with users' expectations and narrow the user-designer gap.

## 3.3 Operational Challenges

Graph abstractions are designed, used, and maintained by a heterogeneous set of human and non-human actors. These challenges focus on identifying relevant user roles, understanding how their goals and expertise differ, and examining how transitions between roles affect abstraction needs, learning processes, and accountability.

### OQ1: Which types of users rely on graph abstractions?

**Why.** As illustrated in Section 2, graph abstractions are used by heterogeneous actors. In Wikidata (Section 2.1), contributors, schema authors, tool developers, and data consumers interact with the same abstraction at different levels. In the MIMIC-III epidemiological case (Section 2.2.1),

knowledge builders, analysts, and healthcare professionals engage with graph representations in distinct ways. In the pangenomic study (Section 2.2.3), students transition from learners to query writers. These examples show that graph abstractions are no longer confined to expert database designers but are appropriated by users with varying goals, expertise, and responsibilities.

**What.** Prior work distinguishes roles such as knowledge builders, analysts, and consumers, particularly in the context of knowledge graphs and visualization systems [59]. Similar role differentiation has been observed in collaborative knowledge bases such as Wikidata, where contributors, curators, tool developers, and data consumers interact with the same abstractions in different ways [63, 74, 98]. However, these roles are often treated implicitly, and most graph abstractions expose a uniform interface that assumes homogeneous expertise and goals.

**How.** Future research should develop explicit, empirically grounded taxonomies of user roles for graph abstractions, linking abstraction features to user goals, capabilities, and responsibilities. Such taxonomies should inform the design of role-aware abstractions, interfaces, and documentation that adapt to different user needs without fragmenting the underlying data model.

## **OQ2: How do transitions between roles shape abstraction needs over time?**

**Why.** Users frequently transition between roles, for example, from learner to analyst, from analyst to schema designer, or from data consumer to contributor. These transitions often involve abrupt increases in abstraction complexity, which can hinder learning, reduce confidence, and lead to misuse or avoidance of graph-based tools. The controlled study on pangenomic graphs (Section 2.2.3) illustrates how learners struggle when transitioning from schema discovery to query formulation, highlighting the cognitive discontinuities that accompany role shifts.

**What.** Empirical studies in database education and graph querying show that novices struggle with abstraction boundaries, schema discovery, and the semantics of query constructs [71]. Similar challenges have been observed in knowledge graph exploration systems, where users report difficulty transitioning from exploratory interaction to precise analytical tasks [61]. Most systems expose the full complexity of abstraction from the outset, offering limited support for progressive learning.

**How.** Supporting role transitions requires graph abstractions and tools that enable the progressive disclosure of complexity. This includes scaffolded interaction modes, learning-oriented query assistance, and interfaces that evolve as users gain expertise. Longitudinal studies are needed to understand how users' mental models of graph abstractions change over time and how abstraction design can facilitate these transitions.

## **OQ3: How should responsibility and accountability be distributed across roles?**

**Why.** Different user roles carry different levels of responsibility for data quality, modeling decisions, and downstream effects. For example, schema designers and constraint authors influence which data are considered valid, while analysts and automated systems rely on these decisions for interpretation and inference.

**What.** Current graph abstractions often obscure responsibility boundaries. Collaborative environments, such as Wikidata, show that modeling decisions are unevenly distributed among contributors, with limited visibility into who is responsible for specific abstraction choices [74, 85]. Similarly, constraint languages such as SHACL and ShEx introduce implicit assumptions that may not be apparent to downstream users [51, 55, 78].

## 2:16 How Human-Centric Are Our Graph Data Abstractions?

**How.** Role-aware graph abstractions should make responsibility explicit by linking modeling and validation actions to provenance, feedback mechanisms, and governance processes. This includes supporting traceability from abstraction decisions to their effects on data quality and analysis outcomes, as well as clarifying accountability in collaborative and automated settings.

### 3.4 Applicative Challenges

Use case challenges examine whether the scenarios commonly used to motivate graph abstractions are representative of mainstream practice, and how evolving construction, refinement, and maintenance workflows, particularly those involving automation and AI, reshape human involvement.

#### AQ1: Do we need several types of mainstream graph construction scenarios?

**Why.** It is important to understand the landscape of existing graph construction methods and use cases to clarify the interplay between human effort and automation in these scenarios [49]. With the surge of LLMs, the problem arises regarding which parts of a graph can be constructed by pre-trained models and which need to be constructed by humans.

**What.** In the past, the construction of knowledge graphs (KG) relied heavily on manual effort and crowd-sourcing methods, such as in highly curated common-sense KGs, e.g., Wikidata [98] and Freebase [22]. As discussed in Section 2.1, Wikidata exemplifies a highly curated, community-driven construction paradigm, whereas the clinical and biological use cases in Section 2.2 illustrate integration-heavy, domain-specific construction processes. These contrasting scenarios suggest that multiple construction paradigms coexist and should be analyzed explicitly.

More recently, deep learning techniques in NLP have led to substantial progress in automatic KG construction, enabling tasks such as entity recognition, entity linking, coreference resolution, and relation extraction [111]. The emergence of pre-trained models, spanning from BERT [38] to the recent LLMs, has allowed us to study the problem of automating knowledge graph construction from large text-based corpora [72]. Knowledge acquisition from other sources, such as images and structured data, has also been considered in Retrieval Augmented Generation (RAG) models.

**How.** Combining automatic knowledge graph construction with manually curated approaches is still an open problem that requires the attention of researchers in the coming years. While automation in KG extraction is desirable to cope with large-scale data, the quality of these approaches and adherence to fairness, equity, and regulatable AI results should be checked [87].

#### AQ2: Which graph refinement and maintenance scenarios are most relevant?

**Why.** This question about the quality of graph-based use cases and their evolution is crucial, as graph data needs to be checked for quality and maintained upon changes, similarly to data in other formats. What is distinctive about graph data use cases is that the underlying relations are *first-class, explicit, and directly traversable edges*; cleaning and refinement operations therefore propagate along these edges across the graph rather than being mediated by value-based joins. The user also plays a central role in graph refinement and cleaning tasks, as fully automated approaches might not reflect what the user has in mind. The Wikidata case (Section 2.1) demonstrates community-driven validation and schema-based quality control, while the epidemiological and biological settings (Section 2.2) highlight domain-driven refinement and analytical correctness requirements. This suggests that refinement and maintenance tasks vary greatly across contexts.

**What.** Recent approaches, such as knowledge graph completion (or link prediction) [64], rule-based graph repairing [35], user-centric repairs [70], and graph data normalization techniques [86] have been proposed to address use cases in which graph data needs to be sanitized. All these approaches assume that rules are either formulated interactively with the help of users or learned from examples on the knowledge graph through Inductive Logic Programming (ILP) approaches. In both cases, the role of users is central to driving the graph completion and repairing tasks. Beyond repairing data, recent work also tests the reliability of graph database applications, for instance by fuzzing them with constraint-violating graphs [39].

**How.** Scenarios that aim to combine symbolic approaches based on graph sampling with interactive graph rule repairing approaches are needed in order to design use cases that reduce human effort in graph data curation. For instance, link prediction can be applied to obtain links that can repair a graph violation, as opposed to edge deletion and node/edge relabeling in current graph repairing use cases.

## 4 Research Roadmap

While the previous sections focus on overarching and cross-cutting challenges, this section refines them into concrete research questions organized by researchability, usability, teachability, and societal impact. These questions distinguish between fundamental investigations and application-driven concerns.

### 4.1 Researchability Challenges

Researchability challenges concern how graph abstractions can be formally defined, compared, and reasoned about by researchers. They address the tension between expressive power, theoretical tractability, interoperability, and accessibility across research communities.

Here the human subject is the researcher. An abstraction becomes a shared object of formal study only once it is specified precisely enough that communities with different backgrounds, e.g., database theory, knowledge representation, systems, can agree on what is being reasoned about and reproduce results over it. Expressiveness, tractability, and interoperability are thus not purely technical desiderata: a property established once over a commonly accepted formalization transfers across communities, whereas under-specified models oblige each group to re-derive its own foundations. Researchability, in this sense, concerns the researcher who studies and extends an abstraction rather than the end user who queries it.

#### 4.1.1 Theory-driven questions

We discuss what makes a graph abstraction suitable for formal study, focusing on how expressiveness, conceptualization, interoperability, and extensibility affect theoretical research on graph data models and query languages.

##### RQ1: What influences the suitability of graph abstractions for formal study?

**Why.** Good graph abstractions allow us to systematically study different aspects of graph data models and query languages, such as complexity and expressiveness, compatibility with the standards, and interoperability and extensibility issues. Depending on our research question, we need to make the appropriate abstractions and understand the important features that we must include in our modeling. Graph abstractions are unifying paradigms that are also necessary for understanding how humans conceptualize graphs.

**What.** The space of graph abstractions is broad. At the data-model level, it forms a lattice over graph-shaped data (the so-called “abstraction soup”), spanning RDF and several property-graph variants [84]. At the query-language level, it comprises a similarly rich landscape of formalisms [7]. From a theoretical viewpoint, recent work has characterized the expressive power of the standardized graph query languages GQL and SQL/PGQ [44], introduced unified formalisms for property graphs schemas and constraints such as PG-Schema [8] and PG-Keys [9], designed declarative formalisms for graph pattern matching and transformation, such as the Graph Pattern Calculus (GPC) [41] and the transformation-oriented GPC+ formalism [27], and proposed a path-based algebra for property graphs [10].

**How.** Despite the above efforts to study graph abstractions suitable for formal studies, the latter mainly focused on theoretical models to delineate the expressive boundaries of graph data models and graph query languages. Further research aspects, such as the conceptualization of graph abstractions, standards interoperability, and their extensibility, need further attention. The focus should be on creating theoretically robust abstractions that are accessible to researchers from different communities.

#### 4.1.2 Application-driven question

We discuss the practical limitations of existing graph abstractions, highlighting the scalability, performance, and expressiveness challenges that arise when these are deployed in large-scale, heterogeneous, and evolving applications.

##### RQ2: What are the practical limitations of existing graph abstractions?

**Why.** In large-scale, highly connected datasets and distributed computing environments, graph abstractions encounter scalability, storage, and communication bottlenecks. For example, large graphs, such as those representing social networks or web-scale data, require massive storage capacities while maintaining fast access and processing times. At the same time, distributed and GPU systems need extensive inter-node communication, which can become a bottleneck for highly connected graphs. In terms of expressiveness, graph abstractions often simplify relationships and structures to fit into node-edge models, while many real-world problems require richer semantics, such as multi-layer or heterogeneous graphs and streaming, dynamic, or temporal structures.

**What.** Beyond scalability and expressiveness, graph abstractions face practical limitations in evolution, validation, and cross-system portability in long-lived applications. Knowledge graphs evolve continuously [75], requiring constraint management and schema validation mechanisms such as SHACL and ShEx, whose complexity and adoption challenges are well documented [24, 51, 55, 78, 79, 93]. Collaborative ecosystems like Wikidata further show that schema design and quality control depend on sustained community coordination and tooling support [63, 98]. Recent work also explores local-first, replicated property graphs that can be edited collaboratively, with concurrent edits converging without central coordination [73]. Meanwhile, heterogeneity across RDF, property graphs, and emerging standards complicates interoperability and reproducibility [1, 60, 84]. Recently, approaches for cross-model graph transformations have been proposed, e.g., from relational to property graphs [94], from RDF to property graphs [77], and from property graphs to RDF [89]. However, more work is needed to cover all possible interoperability issues. These frictions directly impact analytical reliability, lifecycle management, and long-term governance in domains such as healthcare and life sciences [43, 83], so addressing RQ2 requires controlled evolution mechanisms, portable constraints, and abstraction-aware lifecycles.



**How.** Framed through our lens, these are not only performance questions: the practical limits of an abstraction govern whether researchers can study realistic, large-scale instances at all, and therefore which human-facing questions about graph data even become investigable. We need to maintain and enhance the research community’s focus on developing efficient graph operations, such as traversals or centrality computations, and efficient mechanisms to partition or compress graphs without losing essential structural information. Graph databases can handle moderately complex queries, but they struggle with large, deep traversals or global queries (e.g., finding shortest paths in continent-scale graphs), and we need to address such problems. We need to manage irregular graph structures (e.g., power-law distributions in social networks) and the uneven workload distribution across computing nodes. Problems such as community detection, pattern finding, or graph neural network training on large graphs require substantial resources and new graph abstractions. For distributed and GPU computing, communication and load-balancing overhead deserve more focus. Future research should extend graph abstractions to heterogeneous, dynamic, and multi-layer structures while preserving tractable query evaluation and performance.

## 4.2 Usability Challenges

Usability challenges focus on how different categories of users interact with graph abstractions in practice. They examine cognitive load, task alignment, abstraction fit, and the limits of existing mechanisms such as views and interoperability layers.

### 4.2.1 Theory-driven questions

We explore how different categories of users perceive and appropriate graph abstractions, questioning assumptions on intuitiveness and analyzing whether mechanisms such as views suffice to mitigate cognitive load and abstraction complexity.

#### UQ1: User affinity of existing graph abstractions

**Why.** Graph abstractions are often perceived as intuitive and closely aligned with human ways of reasoning about relationships. However, their actual usability varies significantly across user groups. It remains insufficiently understood who benefits from graph abstractions, for which tasks, and under which conditions. In particular, non-technical domain experts may face substantial barriers that limit their ability to effectively engage with graph-based representations. Developing a clearer understanding of user affinity is essential to identify exclusion patterns and to ensure that graph abstractions do not inadvertently privilege certain users or forms of expertise over others. Such insights are also critical for tailoring abstractions to the distinct needs of graph data builders, analysts, consumers, and other user roles discussed later in this section.

**What.** Graph abstractions are well established in technical contexts, such as software engineering (e.g., UML) and conceptual data modeling (e.g., ER diagrams), where their usability is primarily assessed in terms of expressiveness, correctness, and formal properties. By contrast, broader cognitive, accessibility, and learnability aspects have received comparatively less attention. At the same time, graph abstractions are increasingly encountered by non-technical users as data becomes more widely available. This is visible in data-journalism platforms such as the ICIJ Data Explorer<sup>1</sup> and graph-centric investigative tools<sup>2</sup>. A recurring lesson in this setting is that raw graph visualizations (e.g., force-directed layouts) are hard to use, motivating systems that *hide*

<sup>1</sup> <https://offshoreleaks.icij.org/power-players/tony-blair>

<sup>2</sup> <https://team.inria.fr/cedar/connectionlens/>

the underlying graph behind global overviews and tabular exports while retaining graph-based integration [15]. Yet such research is largely qualitative and typically focuses on improving tools for specific, narrowly defined user groups. There is still a lack of systematic, cross-domain studies examining how different populations, particularly users without formal training in data systems, perceive, interpret, and appropriate graph abstractions.

**How.** Addressing these gaps requires combining qualitative and quantitative research methods. In-depth qualitative studies can uncover users' mental models, expectations, and sources of friction when interacting with graph abstractions. At the same time, large-scale surveys are needed to map user demographics, expertise levels, goals, and usage patterns across technical and non-technical communities. Such studies should explicitly investigate factors influencing engagement or disengagement, including perceived complexity, cognitive alignment, task suitability, and tool support. Accessibility concerns must also be considered, including visualization challenges for differently abled users and language barriers for non-English speakers. Collecting feedback from underserved or excluded groups is essential for informing the redesign of graph abstractions. Finally, measuring cognitive load differences across user groups<sup>3</sup> and analyzing learning curves and long-term usage patterns can provide actionable guidance for designing graph abstractions that better align with diverse mental models and workflows.

## UQ2: Limitations of the view-based approach for usability

**Why.** The view-based approach simplifies the representation of complex graph data by focusing only on specific portions that are relevant for a given context. This helps reduce the cognitive load [17] of users, making the data more accessible, especially to non-experts who need to explore or interact with large-scale, densely interconnected, and highly annotated graph data.

**What.** The view-based approach creates dynamic perspectives over the input graph that correspond to particular tasks or user needs [25, 27, 47]. These are often implemented through queries that can be further refined, depending on the functionalities provided by each system. While such methods simplify user interaction, they have several limitations. First, the support for view-based mechanisms is limited and fragmented. Second, given the evolution of the underlying graph data, one needs to ensure that the views remain consistent and that modifications to the views are reflected in the initial graph.

**How.** First, to overcome the fragmented support for view-based mechanisms, it is essential to develop standardized methods and tools for view creation, maintenance, and refinement, leveraging existing standard graph query languages such as GQL and SQL/PGQ. It is important that these are designed in an intuitive, user-friendly manner, allowing users to engage with the graph dynamically in ways that minimize complexity. For example, visual graph view builders or drag-and-drop interfaces can simplify their formulation, making view generation accessible to non-experts [18].

Second, to ensure that the input graph data and the views defined over it remain consistent with one another upon modifications, existing techniques for view maintenance [45] and updates [16] can be adapted from the relational setting. Furthermore, providing users with tools to visualize the effects of view changes, as well as integrating their feedback, can help create more reliable abstractions. Interactive dashboards can help visualize the effects of updates on both the graph and its views, enabling users to understand and validate changes in real time. Integrating user feedback loops ensures that the generated views remain aligned with their evolving needs. Also, adaptive guidance mechanisms, such as recommendations for relevant subgraphs or query templates based on user behavior and historical interactions, can empower users to refine views effectively.

---

<sup>3</sup> [https://en.wikipedia.org/wiki/Cognitive\\_load#Measurement](https://en.wikipedia.org/wiki/Cognitive_load#Measurement)

### UQ3: Suitability of existing graph abstractions for mainstream use cases

**Why.** Graph abstractions [84] are foundational for representing and processing interconnected data across various domains. Their suitability for specific scenarios directly determines their effectiveness in addressing real-world challenges. These scenarios span a wide spectrum, including data augmentation, integration, semantic reasoning, recommendation systems, and advanced machine learning applications such as link prediction and graph representation learning.

**What.** Different types of graph abstractions are needed to effectively address the requirements of these popular tasks. For data augmentation, abstractions that support dynamic and extensible schemas, such as RDF and property graphs, are needed, as they enable the incorporation of diverse data sources and the enrichment of the graph with new relationships and attributes. For data integration, multi-modal graphs that combine heterogeneous data types and support schema alignment are essential, as they facilitate merging datasets into a unified representation. In semantic reasoning, RDF graphs support inference tasks, leveraging ontologies that capture domain-specific semantics. For recommendation systems, multi-relational graph abstractions, such as edge-labeled and property graphs, are well-suited, as they capture user-item interactions, contextual metadata, and dynamic preferences that drive recommendation algorithms. Finally, advanced machine learning applications, such as link prediction and graph representation learning, require abstractions that can efficiently model high-dimensional attributes, dynamic graphs, and hierarchical structures. Abstractions like attributed property graphs or hypergraphs are particularly effective in these scenarios, as they facilitate encoding rich structural and semantic information, enabling the generation of expressive embeddings and robust predictive models.

**How.** To ensure that graph abstractions are well suited for the scenarios in which they are deployed, user-centric approaches should be developed. For example, users can be empowered to combine features from different graph abstractions depending on their specific application requirements. Tools and interfaces should prioritize accessibility for non-expert users through features such as interactive schema creation, guided workflows, and embedded recommendations tailored to common use cases. Task-specific optimizations, such as native support for efficient embeddings in machine learning applications or mechanisms for handling dynamic edge weights in recommendation systems, are crucial for performance. Finally, user feedback loops can help refine abstractions dynamically, allowing systems to adapt to evolving patterns and needs.

## 4.2.2 Application-driven questions

We focus on how well graph abstractions support real-world analytical workflows, examining how they handle iterative analysis, workflow composition, interoperability, and long-term usability.

### UQ4: How do graph abstractions support real-world analytical workflows?

**Why.** In practice, users rarely interact with graph abstractions through isolated queries or single operations. Instead, graph-based analysis is embedded in broader analytical workflows that involve data exploration, hypothesis refinement, iterative querying, validation of intermediate results, and collaboration among multiple users. If graph abstractions fail to support these end-to-end workflows, users may experience friction, increased cognitive load, or be forced to rely on ad-hoc workarounds that undermine both usability and correctness.

**What.** Most usability studies of graph systems focus on localized interactions, such as the expressiveness of a query language, the correctness of query results, or the efficiency of individual tasks. While valuable, these evaluations often abstract away from how users combine multiple

interactions over time to achieve higher-level analytical goals. Hence, important workflow-related aspects, such as how users refine queries incrementally, manage intermediate results, recover from errors, or coordinate analysis across tools and collaborators, remain underexplored. Furthermore, current graph abstractions typically provide limited explicit support for workflow structuring, provenance tracking, or the reuse and adaptation of analysis fragments across tasks and users.

**How.** Addressing this challenge requires moving beyond interaction-level evaluation toward workflow-oriented and longitudinal studies of graph usage. These should examine how different abstractions support iterative exploration, sense-making, and decision-making over extended periods, as well as how they facilitate error diagnosis, correction, and rollback. From a system design perspective, graph abstractions should expose mechanisms for structuring analytical workflows, such as composable queries, reusable views, and explicit representations of intermediate results and dependencies. Support for collaboration, including shared abstractions, annotation, and provenance-aware analysis, is also essential. By aligning graph abstractions more closely with real-world analytical workflows, systems can improve their usability and reliability.

#### **UQ5: Limitations of existing graph abstractions for usability**

**Why.** Existing graph abstractions tend to lock users and applications into a specific ecosystem centered around those abstractions. This is often determined by the particular use case and the needs regarding how the data will be used. For example, RDF graphs are designed to work well in scenarios where ontologies are available and are thus well suited for reasoning. Property graphs, however, tend to support network analyses and connectivity queries more efficiently.

**What.** While use cases often determine which graph abstraction is better suited, the same graph might be used in different use cases in different ways. Hence, there has been some work on interoperability, e.g., by transforming one abstraction into another or developing a data model that serves as multiple graph abstractions that can be used to access the data [56, 99]. However, this so far only considers translation between query languages and data models.

**How.** Since existing graph abstractions differ in their limitations and strengths, we need to find a way to combine the strengths of the existing ones to overcome their limitations. This can either lead to the development of a new abstraction that encompasses existing ones or to the development of a framework that achieves full interoperability at all levels, including query languages, schema languages, data models, etc.

### **4.3 Education/Teachability Challenges**

Teachability challenges address how graph abstractions are learned and taught, and how their cognitive demands compare to other abstractions. They emphasize the need for strategies that explicitly support abstraction formation, mental model alignment, and progressive learning.

#### **4.3.1 Theory-driven questions**

We discuss the pedagogical aspects and the fundamental cognitive difficulties learners face when reasoning about graph-based representations.

##### **TQ1: Criteria for selecting a suitable graph abstraction for teaching**

**Why.** Abstractions are central to computational thinking [3, 54, 88], alongside decomposition, design, debugging, iteration, and generalization [88], as they allow one to ignore non-essential features, extract similarities between concepts, and generate further knowledge [31].

Given their educational role, it is crucial to choose appropriate graph abstractions to enhance learner understanding and make materials comprehensible, relatable, and applicable to real-world problems.

**What.** The criteria should emphasize clarity, context, visual intuition, and computational relevance. A suitable abstraction presents graph objects and the additional features and/or constraints attached to these in ways that students can quickly comprehend and manipulate, ideally with real-world examples that underscore practical applications.

**How.** To achieve this, instructors can prioritize abstractions that offer clear visualizations and gradually introduce more advanced topics to ensure that learners build a strong conceptual foundation before exploring more complex concepts. Practical exercises, real-world analogies, and intuitive visual aids are key strategies that foster engagement, reinforce comprehension, and make abstract graph concepts more tangible and meaningful.

## **TQ2: Inherent difficulties in learning and understanding graph abstractions**

**Why.** Unlike tabular or hierarchical models, which align with natural cognitive tendencies for sequential or categorical reasoning, graph-based representations emphasize interdependencies and structural flexibility. This shift demands a different cognitive approach, making graphs less intuitive to grasp [48, 65]. Additionally, the diversity of mental models among learners further complicates comprehension, requiring tailored instructional methods [14].

**What.** The challenge stems from the way graph abstractions prioritize relationships over individual data points, requiring learners to understand connectivity rather than isolated entities. Research shows that humans often struggle with networked representations due to their reliance on linear or hierarchical thinking [48, 65, 109]. Additionally, as Williams et al. [108] highlight, learners approach graph-based models with varying conceptual frameworks, making it difficult to standardize effective teaching strategies.

**How.** To improve learning, educators should employ progressive abstraction, beginning with concrete, real-world graph representations before transitioning to abstract models [48]. Utilizing visualization-driven learning, such as interactive exploration and multiple representation formats, can accommodate diverse cognitive styles [14]. Additionally, integrating path diagrams and mental model elicitation techniques [109] helps align instructional methods with how learners conceptualize graphs. Project-based and challenge-based learning approaches provide hands-on engagement, reinforcing practical understanding and making abstract graph reasoning more accessible.

### **4.3.2 Application-driven questions**

We analyze the limitations of existing methods for teaching graph abstractions, focusing on mismatches between pedagogical practices, learners' mental models, and the cognitive demands imposed by graph abstractions in educational settings.

## **TQ3: Limitations of current methods for teaching graph abstractions**

**Why.** Teaching graph abstractions is essential for computational thinking, as they model complex systems and relationships. However, current methods often fail to address the cognitive complexity of graph abstractions and are further limited by a weak focus on user-centered approaches and inadequate support for learners' diverse mental models [48, 65]. Moreover, Bach et al. [14] emphasize that visualization education must cater to a diverse audience while addressing

methodological and contextual challenges. Additionally, Hazzan [48] highlights that teaching abstraction requires overcoming uncertainty and cognitive complexity, especially for concepts like graphs, which demand conscious navigation across abstraction levels.

**What.** These limitations stem from a lack of systematic frameworks for teaching graph abstractions and from instructional methods poorly aligned with learners’ cognitive processes. Moore et al. [65] extend Piaget’s theories of abstraction to show how instructional materials shape the development of stable knowledge structures, while Bach et al. [14] identify challenges such as fostering visual representation skills, addressing learner diversity, and assessing project-based learning. This misalignment between instructors’ and learners’ mental models significantly impacts comprehension and performance [108, 109].

**How.** Addressing these limitations requires integrating human-centric, evidence-based strategies for teaching graph abstractions. Educators must adopt methods such as project-based learning and challenge-based learning, as suggested by Bach et al. [14], to foster engagement and contextual understanding. Hazzan [48] advocates for raising awareness of abstraction levels and teaching learners to navigate these levels consciously to mitigate cognitive complexity. Drawing on Moore et al. [65], instructional materials should be designed to support abstraction processes, focusing on the gradual development of knowledge structures. Incorporating tools like mental model elicitation, such as path diagrams [109], and iterative user testing [108] can align teaching methods with learners’ understanding. These approaches help educators bridge abstract graph concepts and learner comprehension, making graph abstractions more accessible.

## 4.4 Societal Challenges

Societal challenges consider the broader consequences of graph abstractions beyond direct users. They include fairness, accountability, transparency, and the indirect impact of graph-based systems on individuals and communities affected by their deployment.

### 4.4.1 Theory-driven questions

We examine how graph abstraction choices shape fairness, bias, and transparency, analyzing how modeling decisions influence downstream reasoning and affect stakeholders who are not direct users of graph-centric data systems.

#### **SQ1: Fairness and bias induced by graph abstractions**

**Why.** Graph abstractions play a central role in shaping how data is structured, connected, and interpreted. Choices about which entities, relationships, attributes, and constraints are modeled influence downstream reasoning, analytics, and automated decisions. When graph abstractions are used in sensitive domains or as part of AI pipelines, such design choices can amplify existing biases or introduce new ones, with consequences that extend beyond technical users to affected individuals and groups.

**What.** Bias may be introduced at multiple stages of abstraction design, including schema modeling, data integration, constraint definition, and inference mechanisms. For example, large shared knowledge graphs such as Wikidata or Freebase embed collective modeling decisions that reflect the perspectives, priorities, and coverage of their contributing communities [22, 74, 98]. Similarly, choices in validation languages and constraints (e.g., SHACL or ShEx) can privilege certain data patterns while marginalizing others [55]. Despite their impact, these biases are often opaque to end users and difficult to trace back to specific abstraction decisions.

**How.** Addressing fairness in graph abstractions requires systematic methods for identifying how abstraction choices propagate bias across graph-based workflows. This includes developing auditing techniques that relate schema and constraint design to observed disparities in query results or automated outcomes, as well as supporting comparative analyses across alternative abstractions. Human-centric approaches should also expose abstraction assumptions to users, enabling critical revisions of modeling decisions, particularly in high-stakes applications.

## **SQ2: Transparency and interpretability for affected parties**

**Why.** Graph-based systems increasingly influence decisions that affect people who are not the direct users of graph abstractions, such as patients, citizens, or content consumers. In these settings, the lack of transparency regarding how graph abstractions shape results can undermine trust, accountability, and the ability to contest or understand outcomes.

**What.** Existing graph abstractions and query languages are primarily designed to be interpretable by technical experts, emphasizing formal semantics and expressive power [7]. However, non-expert stakeholders often lack access to meaningful explanations of how data is structured, how inferences are derived, and how uncertainty and incompleteness are handled. This gap is particularly evident when graph abstractions are combined with machine learning or language models, where abstraction layers and automated reasoning further obscure causal links between data and outcomes [72, 87].

**How.** Human-centric graph abstractions should support explanation mechanisms that translate technical representations into accessible accounts of modeling assumptions, limitations, and uncertainty. This may include abstraction-aware explanations of query results, provenance tracking across abstraction layers, and role-specific views that communicate relevant information to affected parties without requiring technical expertise. Such mechanisms are essential for enabling informed oversight and meaningful engagement with graph-based systems.

### **4.4.2 Application-driven questions**

We overview societal and governance challenges associated with graph abstractions, identifying their role in automated decision making and in managing shared abstractions within collaborative and institutional settings.

## **SQ3: Societal impact of graph abstractions in automated decision systems**

**Why.** Graph abstractions are increasingly embedded in automated decision-making pipelines, including those involving machine learning, recommender systems, and large language models. In these contexts, abstraction choices can influence not only system performance but also real-world outcomes: access to resources, prioritization of information, and clinical and policy decisions.

**What.** While there is growing awareness of the societal risks associated with automated systems, the specific role played by graph abstractions within these pipelines remains poorly understood. Studies often focus on algorithmic fairness or model behavior in isolation, overlooking how abstraction design, e.g., schema structure, integration of heterogeneous data, or constraint enforcement, shapes the inputs and outputs of automated processes. Case studies in domains such as healthcare and epidemiology illustrate both the potential and the risks of graph-based decision support when abstraction choices are insufficiently scrutinized [52, 76].

**How.** Application-level research should examine the societal impact of graph abstractions through empirical studies of real deployments. This includes analyzing how different abstraction choices affect decision outcomes, error propagation, and disparate impacts across populations. Such studies should be complemented by design guidelines that help practitioners assess the societal implications of abstraction choices when integrating graphs into automated systems.

#### SQ4: Governance of shared graph abstractions

**Why.** Many large-scale graph systems rely on shared abstractions maintained by communities or institutions, such as collaboratively curated knowledge graphs or domain-wide schemas. The long-term sustainability and societal trustworthiness of these systems depend on how changes to abstractions are governed, negotiated, and communicated.

**What.** Existing governance mechanisms for shared graph abstractions are often informal, uneven, or implicit. Studies of collaborative platforms such as Wikidata show that a relatively small subset of contributors exerts disproportionate influence over modeling decisions, schema evolution, and editorial norms [74, 85]. While this enables scalability, it also raises concerns about representation, accountability, and inclusiveness, particularly as these abstractions are reused across domains and applications.

**How.** Designing graph abstractions with explicit governance and stewardship mechanisms is essential for responsible and sustainable use. This includes support for versioning, documented design rationales, community feedback, and mechanisms to assess the impact of abstraction changes on downstream users and applications. Embedding governance considerations into abstraction design can help align technical evolution with societal values and long-term accountability.

## 5 Conclusion

We have examined the human-centricity of graph abstractions, analyzing several current paradigms through use cases ranging from collaborative knowledge graphs to clinical and biological applications, and we have distilled the research challenges they raise from a data management perspective.

Making graph abstractions more user-centric opens several directions for future work. The most immediate is empirical: targeted user studies, each focused on a single dataset and processing paradigm, should assess how well current graph paradigms serve particular applications and, in aggregate, characterize what users expect from a given data or query model. A second concerns expressiveness, which we argue should be measured jointly with users' perception of that expressive power. The analysis of Wikidata query logs [26] is illustrative in this respect: queries are posed by both humans and bots, and although recursive operators are common, the regular path queries that actually appear are structurally simple. Extending such analyses beyond RDF would help close the gap between formal power and practical use. A final direction is to study machine-centric abstractions alongside human-centric ones, including how well generative models reason over graph models beyond simple ones [40].

We believe the next generation of graph abstractions must treat expressiveness and human accessibility equally as first-class design goals.

---

## References

- |                                                                                                                                                                           |                                                                                                                                                                                              |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>1 Shqiponja Ahmetaj, Iovka Boneva, Jan Hidders, Katja Hose, Maxime Jakubowski, José Emilio Labra Gayo, Wim Martens, Fabio Mogavero, Filip Murlak, Cem Okumus, Axel</p> | <p>Polleres, Ognjen Savkovic, Mantas Simkus, and Dominik Tomaszuk. Common foundations for SHACL, ShEx, and PG-Schema. In <i>WWW</i>, pages 8–21. ACM, 2025. doi:10.1145/3696410.3714694.</p> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|



- 2 Shqiponja Ahmetaj, Slawomir Staworko, Jan Van den Bussche, and Maxime Jakubowski. Shapes in graph data: Theory and implementation (Dagstuhl Seminar 24102). *Dagstuhl Reports*, 14(3):9–30, 2024. doi:10.4230/DAGREP.14.3.9.
- 3 Alfred V. Aho and Jeffrey D. Ullman. Abstractions, their algorithms, and their compilers. *Commun. ACM*, 65(2):76–91, 2022. doi:10.1145/3490685.
- 4 Sihem Amer-Yahia, Jasmina Bogojeska, Roberta Facchinetti, Valeria Franceschi, Aristides Gionis, Katja Hose, Georgia Koutrika, Roger D. Kouyos, Matteo Lissandrini, Silviu Maniu, Katsiaryna Mirylenka, Davide Mottin, Themis Palpanas, Mattia Rigotti, and Yannis Velegrakis. Towards reliable conversational data analytics. In *EDBT*, pages 962–969. OpenProceedings.org, 2025. doi:10.48786/EDBT.2025.78.
- 5 Magdalena Andrzejewska and Anna Stolinska. Do structured flowcharts outperform pseudocode? evidence from eye movements. *IEEE Access*, 10:132965–132975, 2022. doi:10.1109/ACCESS.2022.3230981.
- 6 Renzo Angles. The property graph database model. In *AMW, CEUR Workshop Proceedings*. CEUR-WS.org, 2018. URL: <https://ceur-ws.org/Vol-2100/paper26.pdf>.
- 7 Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoc. Foundations of modern query languages for graph databases. *ACM Comput. Surv.*, 50(5):68:1–68:40, 2017. doi:10.1145/3104031.
- 8 Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Alastair Green, Jan Hidders, Bei Li, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Stefan Plantikow, Ognjen Savkovic, Michael Schmidt, Juan Sequeda, Slawek Staworko, Dominik Tomaszuk, Hannes Voigt, Domagoj Vrgoc, Mingxi Wu, and Dusan Zivkovic. Pg-schema: Schemas for property graphs. *Proc. ACM Manag. Data*, 1(2):198:1–198:25, 2023. doi:10.1145/3589778.
- 9 Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Keith W. Hare, Jan Hidders, Victor E. Lee, Bei Li, Leonid Libkin, Wim Martens, Filip Murlak, Josh Perryman, Ognjen Savkovic, Michael Schmidt, Juan F. Sequeda, Slawek Staworko, and Dominik Tomaszuk. Pg-keys: Keys for property graphs. In *SIGMOD Conference*, pages 2423–2436. ACM, 2021. doi:10.1145/3448016.3457561.
- 10 Renzo Angles, Angela Bonifati, Roberto García, and Domagoj Vrgoc. Path-based algebraic foundations of graph query languages. In *EDBT*, pages 783–795. OpenProceedings.org, 2025. doi:10.48786/EDBT.2025.63.
- 11 Marcelo Arenas, Enrico Franconi, Janik Hammerer, Olaf Hartig, Katja Hose, Laura Koesten, George Konstantinidis, Leonid Libkin, Wim Martens, Yuya Sasaki, Stefanie Scherzinger, Katherine Thornton, and Hsiang-Yun Wu. Exploring exploratory querying. *Proc. VLDB Endow.*, 18(13):5731–5739, 2025. URL: <https://www.vldb.org/pvldb/vol18/p5731-konstantinidis.pdf>.
- 12 Jérôme Arnoux, Angela Bonifati, Alexandra Calteau, Stefania Dumbrava, and Guillaume Gautreau. Integrating complex pangenome graphs. In *ICDEW*, pages 350–354. IEEE, 2024. doi:10.1109/ICDEW61823.2024.00052.
- 13 Jakob J Assmann, Jesper E Moeslund, Urs A Treier, and Signe Normand. EcoDes-DK15: High-resolution ecological descriptors of vegetation and terrain derived from Denmark’s national airborne laser scanning data set. *Earth System Science Data*, 2022.
- 14 Benjamin Bach, Mandy Keck, Fateme Rajabiyazdi, Tatiana Losev, Isabel Meirelles, Jason Dykes, Robert S. Laramée, Mashael AlKadi, Christina Stoiber, Samuel Huron, Charles Perin, Luiz Morais, Wolfgang Aigner, Doris Kosminsky, Magdalena Boucher, Søren Knudsen, Areti Manataki, Jan Aerts, Uta Hinrichs, Jonathan C. Roberts, and Sheelagh Carpendale. Challenges and opportunities in data visualization education: A call to action. *IEEE Trans. Vis. Comput. Graph.*, 30(1):649–660, 2024. doi:10.1109/TVCG.2023.3327378.
- 15 Oana Balalau, Nelly Barret, Simon Ebel, Théo Galizzi, Ioana Manolescu, and Madhulika Mohanty. Graph lenses over any data: the ConnectionLens experience. In *ICDEW*, pages 370–374. IEEE, 2024. doi:10.1109/ICDEW61823.2024.00056.
- 16 François Bancilhon and Nicolas Spyrtatos. Update semantics of relational views. *ACM Trans. Database Syst.*, 6(4):557–575, 1981. doi:10.1145/319628.319634.
- 17 Sourav S. Bhowmick, S. H. Annabel Chen, and Divesh Srivastava. Cognitive psychology meets data management: State of the art and future directions. In *SIGMOD Conference Companion*, pages 590–596. ACM, 2024. doi:10.1145/3626246.3654682.
- 18 Sourav S. Bhowmick and Byron Choi. Data-driven visual query interfaces for graphs: Past, present, and (near) future. In *SIGMOD Conference*, pages 2441–2447. ACM, 2022. doi:10.1145/3514221.3522562.
- 19 Sourav S. Bhowmick, Byron Choi, and Chengkai Li. Graph querying meets HCI: state of the art and future directions. In *SIGMOD Conference*, pages 1731–1736. ACM, 2017. doi:10.1145/3035918.3054774.
- 20 Sourav S. Bhowmick, Byron Choi, and Chengkai Li. *Human Interaction with Graphs: A Visual Querying Perspective*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2018. doi:10.2200/S00855ED1V01Y201805DTM047.
- 21 Alex Bigelow, Katy Williams, and Katherine E. Isaacs. Guidelines for pursuing and revealing data abstractions. *IEEE Trans. Vis. Comput. Graph.*, 27(2):1503–1513, 2021. doi:10.1109/TVCG.2020.3030355.
- 22 Kurt Bollacker, Robert P. Cook, and Patrick Tufts. Freebase: A shared database of structured general human knowledge. In *AAAI*, pages 1962–1963. AAAI Press, 2007. URL: <http://www.aaai.org/Library/AAAI/2007/aaai07-355.php>.
- 23 Iovka Boneva, Jérémie Dusart, Daniel Fernández-Álvarez, and José Emilio Labra Gayo. Shape designer for ShEx and SHACL constraints. In *ISWC (Satellites)*, CEUR Workshop Proceedings, pages 269–272. CEUR-WS.org, 2019. URL: <https://ceur-ws.org/Vol-2456/paper70.pdf>.

- 24 Iovka Boneva, José Emilio Labra Gayo, Samuel Hym, Eric G. Prud'hommeaux, Harold R. Solbrig, and Slawek Staworko. Validating RDF with shape expressions. *CoRR*, abs/1404.1270, 2014. [arXiv:1404.1270](https://arxiv.org/abs/1404.1270).
- 25 Angela Bonifati, Stefania Dumbrava, and Emilio Jesús Gallego Arias. Certified Graph View Maintenance with Regular Datalog. *Theory Pract. Log. Program.*, 18(3-4):372–389, 2018. doi:10.1017/S1471068418000224.
- 26 Angela Bonifati, Wim Martens, and Thomas Timm. Navigating the Maze of Wikidata Query Logs. In *WWW*, pages 127–138. ACM, 2019. doi:10.1145/3308558.3313472.
- 27 Angela Bonifati, Filip Murlak, and Yann Ramusat. Transforming property graphs. *Proc. VLDB Endow.*, 17(11):2906–2918, 2024. doi:10.14778/3681954.3681972.
- 28 Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible Markup Language (XML) 1.0, W3C. <http://www.w3.org/TR/REC-xml/>, 1998.
- 29 Alison Callahan, Jose Cruz-Toledo, Peter Ansell, and Michel Dumontier. Bio2RDF release 2: Improved coverage, interoperability and provenance of life science linked data. In *ESWC*, LNCS, pages 200–212. Springer, 2013. doi:10.1007/978-3-642-38288-8\_14.
- 30 Donald D. Chamberlin and Raymond F. Boyce. SEQUEL: A structured english query language. In *SIGMOD Workshop, Vol. 1*, pages 249–264. ACM, 1974. doi:10.1145/800296.811515.
- 31 John H. Chambers. The difference between the abstract concepts of science and the general concepts of empirical educational research. *Journal of Educational Thought / Revue de la Pensée Educative*, 2018. URL: <https://api.semanticscholar.org/CorpusID:151011055>.
- 32 Kasidis Chanthatrojwong, Sourav S. Bhowmick, and Byron Choi. LICS: towards theory-informed effective visual abstraction of property graph schemas. *Proc. ACM Manag. Data*, 3(3):180:1–180:26, 2025. doi:10.1145/3725410.
- 33 Kasidis Chanthatrojwong, Sourav S. Bhowmick, and Byron Choi. PASCAL: A theory-informed visual interface for property graph schema visualization. In *SIGMOD Conference Companion*, pages 67–70. ACM, 2025. doi:10.1145/3722212.3725123.
- 34 Peter P. Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976. doi:10.1145/320434.320440.
- 35 Yurong Cheng, Lei Chen, Ye Yuan, and Guoren Wang. Rule-Based Graph Repairing: Semantic and Efficient Repairing Methods. In *ICDE*, pages 773–784. IEEE Computer Society, 2018. doi:10.1109/ICDE.2018.00075.
- 36 Mads Corfixen, Thomas Heede, Tomer Sagi, Mads Albertsen, Thomas D. Nielsen, and Katja Hose. Integrating Multi-Modal Spatial Data using Knowledge Graphs - a Case Study of Microflora Danica. In *SeWeBMeDA@ESWC*, CEUR Workshop Proceedings. CEUR-WS.org, 2024. URL: <https://ceur-ws.org/Vol-3726/paper1.pdf>.
- 37 Resul Das and Mucabit Soyulu. A key review on graph data science: The power of graphs in scientific studies. *Chemometrics and Intelligent Laboratory Systems*, 240:104896, September 2023. doi:10.1016/j.chemolab.2023.104896.
- 38 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*, pages 4171–4186. Association for Computational Linguistics, 2019. doi:10.18653/V1/N19-1423.
- 39 Stefania Dumbrava, Melchior W. M. Oudemans, and Burcu Kulahcioglu Ozkan. Fuzzing graph database applications with graph transformations. In *ICGT*, volume 15720 of *Lecture Notes in Computer Science*, pages 135–156. Springer, 2025. doi:10.1007/978-3-031-94706-3\_7.
- 40 Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. Talk like a Graph: Encoding Graphs for Large Language Models. In *ICLR*. OpenReview.net, 2024. URL: <https://openreview.net/forum?id=TuXR1CCrSi>.
- 41 Nadime Francis, Amélie Gheerbrant, Paolo Guagliardo, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Liat Peterfreund, Alexandra Rogova, and Domagoj Vrgoc. GPC: A pattern calculus for property graphs. In *PODS*, pages 241–250. ACM, 2023. doi:10.1145/3584372.3588662.
- 42 Wolfgang Gatterbauer and Cody Dunne. Relational diagrams and the pattern expressiveness of relational languages. *SIGMOD Rec.*, 54(1):80–88, April 2025. doi:10.1145/3733620.3733637.
- 43 Sandra Geisler, Cinzia Cappiello, Irene Celino, David Chaves-Fraga, Anastasia Dimou, Ana Iglesias-Molina, Maurizio Lenzerini, Anisa Rula, Dylan Van Assche, Sascha Welten, and Maria-Esther Vidal. From genesis to maturity: Managing knowledge graph ecosystems through life cycles. *Proc. VLDB Endow.*, 18(5):1390–1397, 2025. doi:10.14778/3718057.3718067.
- 44 Amélie Gheerbrant, Leonid Libkin, Liat Peterfreund, and Alexandra Rogova. GQL and SQL/PGQ: theoretical models and expressive power. *Proc. VLDB Endow.*, 18(6):1798–1810, 2025. doi:10.14778/3725688.3725707.
- 45 Ashish Gupta and Inderpal Singh Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Eng. Bull.*, 18(2):3–18, 1995. URL: <http://sites.computer.org/debull/95JUN-CD.pdf>.
- 46 Armin Haller and Axel Polleres. Are we better off with just one ontology on the Web? *Semantic Web*, 11(1):87–99, 2020. doi:10.3233/SW-190379.
- 47 Soonbo Han and Zachary G. Ives. Implementation strategies for views over property graphs. *Proc. ACM Manag. Data*, 2(3):146, 2024. doi:10.1145/3654949.
- 48 Orit Hazzan. Reflections on teaching abstraction and other soft ideas. *ACM SIGCSE Bull.*, 40(2):40–43, 2008. doi:10.1145/1383602.1383631.
- 49 Marvin Hofer, Daniel Obraczka, Alieh Saeedi, Hanna Köpcke, and Erhard Rahm. Construction of Knowledge Graphs: Current State and Challenges. *Inf.*, 15(8):509, 2024. doi:10.3390/INF015080509.
- 50 Gwo-Jen Hwang, Haoran Xie, Benjamin W. Wah, and Dragan Gašević. Vision, challenges, roles and

- research issues of artificial intelligence in education. *Computers and Education: Artificial Intelligence*, 1:100001, 2020. doi:10.1016/j.caeai.2020.100001.
- 51 Maxime Jakubowski, Dominik Tomaszuk, and Katja Hose. Bridging Gaps in RDF Validation—Insights and Innovation Opportunities in RDF Validation Practices. In *Linking Meaning: Semantic Technologies Shaping the Future of AI*, pages 70–84. IOS Press, 2025. doi:10.3233/SSW250011.
  - 52 Alistair E.W. Johnson, Tom J. Pollard, Lu Shen, Li-wei H. Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G. Mark. Mimic-III, a freely accessible critical care database. *Scientific Data*, 3(1), May 2016. doi:10.1038/sdata.2016.35.
  - 53 Lucie-Aimée Kaffee, Alessandro Piscopo, Pavlos Vougiouklis, Elena Simperl, Leslie Carr, and Lydia Pintscher. A Glimpse into Babel: An Analysis of Multilinguality in Wikidata. In *OpenSym*, pages 14:1–14:5. ACM, 2017. doi:10.1145/3125433.3125465.
  - 54 Jeff Kramer. Is abstraction the key to computing? *Commun. ACM*, 50(4):36–42, 2007. doi:10.1145/1232743.1232745.
  - 55 Jose Emilio Labra-Gayo, Herminio García-González, Daniel Fernández-Alvarez, and Eric Prud’hommeaux. Challenges in RDF validation. In *Current Trends in Semantic Web Technologies: Theory and Practice*, pages 121–151. Springer, 2019. doi:10.1007/978-3-030-06149-4\_6.
  - 56 Ora Lassila, Michael Schmidt, Olaf Hartig, Brad Bebee, Dave Bechberger, Willem Broekema, Ankesh Khandelwal, Kelvin Lawrence, Carlos Manuel López Enríquez, Ronak Sharda, and Bryan B. Thompson. The OneGraph vision: Challenges of breaking the graph model lock-in. *Semantic Web*, 14(1):125–134, 2023. doi:10.3233/SW-223273.
  - 57 Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. <https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/>, 1999.
  - 58 Minmin Leng, Yue Sun, Ce Li, Shuyu Han, and Zhiwen Wang. Usability evaluation of a knowledge graph-based dementia care intelligent recommender system: Mixed methods study. *Journal of Medical Internet Research*, 25:e45788, September 2023. doi:10.2196/45788.
  - 59 Harry X. Li, Gabriel Appleby, Camelia Daniela Brumar, Remco Chang, and Ashley Suh. Knowledge graphs in practice: Characterizing their users, challenges, and visualization opportunities. *IEEE Trans. Vis. Comput. Graph.*, 30(1):584–594, 2024. doi:10.1109/TVCG.2023.3326904.
  - 60 Tobias Lindaaker. An overview of the recent history of graph query languages. Technical report dm32.2-2018-00085r1, ISO/IEC JTC 1 Database Languages WG3, May 2018. Prepared for ISO/IEC JTC 1 WG3. URL: [https://s3.amazonaws.com/artifacts.opencypher.org/website/materials/DM32.2-2018-00085R1-recent\\_history\\_of\\_property\\_graph\\_query\\_languages.pdf](https://s3.amazonaws.com/artifacts.opencypher.org/website/materials/DM32.2-2018-00085R1-recent_history_of_property_graph_query_languages.pdf).
  - 61 Matteo Lissandrini, Davide Mottin, Katja Hose, and Torben Bach Pedersen. Knowledge graph exploration systems: are we lost? In *CIDR*. [www.cidrdb.org](http://www.cidrdb.org), 2022. URL: <https://vldb.org/cidrdb/2022/knowledge-graph-exploration-systems-are-we-lost.html>.
  - 62 Jiebing Ma, Sourav S. Bhowmick, Byron Choi, and Lester Tay. Theories and principles matter: Towards visually appealing and effective abstraction of property graph queries. *Proc. ACM Manag. Data*, 1(2):132:1–132:26, 2023. doi:10.1145/3589277.
  - 63 Stanislav Malyshev, Markus Krötzsch, Larry González, Julius Gonsior, and Adrian Bielefeldt. Getting the Most Out of Wikidata: Semantic Technology Usage in Wikipedia’s Knowledge Graph. In *ISWC (2)*, LNCS, pages 376–394. Springer, 2018. doi:10.1007/978-3-030-00668-6\_23.
  - 64 Christian Meilicke, Melisachew Wudage Chekol, Patrick Betz, Manuel Fink, and Heiner Stuckenschmidt. Anytime bottom-up rule learning for large-scale knowledge graph completion. *VLDB J.*, 33(1):131–161, 2024. doi:10.1007/S00778-023-00800-5.
  - 65 Kevin C. Moore, Erin Wood, Shaffiq Welji, Mike Hamilton, Anne Waswa, Amy B. Ellis, and Halil I. Tasova. Using abstraction to analyze instructional tasks and their implementation. *The Journal of Mathematical Behavior*, 74:101153, 2024. doi:10.1016/j.jmathb.2024.101153.
  - 66 Nebula Graph Documentation. An introduction to graph databases. <https://docs.nebula-graph.io/3.4.3/1.introduction/0-1-graph-database/>, 2023. Accessed: 2026-02.
  - 67 Finn Årup Nielsen, Katherine Thornton, and José Emilio Labra Gayo. Validating Danish Wikidata lexemes. In *SEMANTiCS (Posters & Demos)*, CEUR Workshop Proceedings. CEUR-WS.org, 2019. URL: <https://ceur-ws.org/Vol-2451/paper-20.pdf>.
  - 68 Farnaz Nouraei, Alexa F. Siu, Ryan A. Rossi, and Nedim Lipka. Thinking outside the box: Non-designer perspectives and recommendations for template-based graphic design tools. In *CHI Extended Abstracts*, pages 326:1–326:9. ACM, 2024. doi:10.1145/3613905.3650967.
  - 69 openCypher Implementers Group. openCypher. <https://s3.amazonaws.com/artifacts.opencypher.org/website/>, 2015. Accessed: 2026-02.
  - 70 Amedeo Pachera, Angela Bonifati, and Andrea Mauri. User-centric property graph repairs. *Proc. ACM Manag. Data*, 3(1):85:1–85:27, 2025. doi:10.1145/3709735.
  - 71 Amedeo Pachera, Stefania Dumbrava, Angela Bonifati, and Andrea Mauri. Understanding student errors in graph query formulation. *ACM Trans. Comput. Educ.*, 25(3):1–35, 2025. doi:10.1145/3743687.
  - 72 Jeff Z. Pan, Simon Razniewski, Jan-Christoph Kalo, Sneha Singhania, Jiaoyan Chen, Stefan Dietze, Hajira Jabeen, Janna Omeliyanenko, Wen Zhang, Matteo Lissandrini, Russa Biswas, Gerard de Melo, Angela Bonifati, Edlira Vakaj, Mauro Dragoni, and Damien Graux. Large language models and knowledge graphs: Opportunities and challenges. *TGDK*, 1(1):2:1–2:38, 2023. doi:10.4230/TGDK.1.1.2.

- 73 Ayush Pandey, Stefania Dumbrava, Marc Shapiro, Carla Ferreira, Mário Pereira, and Nuno M. Prego. Towards local-first distributed property graphs. In *PaPoC@EuroSys*, pages 22–29. ACM, 2025. doi:10.1145/3721473.3722139.
- 74 Alessandro Piscopo and Elena Simperl. Who Models the World?: Collaborative Ontology Creation and User Roles in Wikidata. *Proc. ACM Hum. Comput. Interact.*, 2(CSCW):141:1–141:18, 2018. doi:10.1145/3274410.
- 75 Axel Polleres, Romana Pernisch, Angela Bonifati, Daniele Dell’Aglia, Daniil Dobriy, Stefania Dumbrava, Lorena Etcheverry, Nicolas Ferranti, Katja Hose, Ernesto Jiménez-Ruiz, Matteo Lissandrini, Ansgar Scherp, Riccardo Tommasini, and Johannes Wachs. How Does Knowledge Evolve in Open Knowledge Graphs? *TGDK*, 1(1):11:1–11:59, 2023. doi:10.4230/TGDK.1.1.11.
- 76 Lorena Pujante-Otalora, Manuel Campos, Jose M. Juarez, and Maria-Esther Vidal. Comparison between graph databases and RDF engines for modelling epidemiological investigation of nosocomial infections. In *BIOSTEC (2)*, pages 23–36. SCITEPRESS, 2024. doi:10.5220/0012319900003657.
- 77 Kashif Rabbani, Matteo Lissandrini, Angela Bonifati, and Katja Hose. Transforming RDF Graphs to Property Graphs using Standardized Schemas. *Proc. ACM Manag. Data*, 2(6), December 2024. doi:10.1145/3698817.
- 78 Kashif Rabbani, Matteo Lissandrini, and Katja Hose. SHACL and ShEx in the wild: A community survey on validating shapes generation and adoption. In *WWW (Companion Volume)*, pages 260–263. ACM, 2022. doi:10.1145/3487553.3524253.
- 79 Kashif Rabbani, Matteo Lissandrini, and Katja Hose. Extraction of Validating Shapes from very large Knowledge Graphs. *Proc. VLDB Endow.*, 16(5):1023–1032, 2023. doi:10.14778/3579075.3579078.
- 80 Phyllis Reisner, Raymond F. Boyce, and Donald D. Chamberlin. Human factors evaluation of two data base query languages: SQUARE and SEQUEL. In *AFIPS National Computer Conference*, AFIPS Conference Proceedings, pages 447–452. AFIPS Press, 1975. doi:10.1145/1499949.1500036.
- 81 WMDE research. Questions, Methods and Results for the Wikidata Community Survey, 2021. URL: [https://commons.wikimedia.org/wiki/File:Wikidata\\_Community\\_Survey\\_2021.pdf](https://commons.wikimedia.org/wiki/File:Wikidata_Community_Survey_2021.pdf).
- 82 Ian Robinson, Jim Webber, and Emil Eifrem. *Graph databases: new opportunities for connected data*. O’Reilly Media, Inc., 2015.
- 83 Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M. Tamer Özsu. The ubiquity of large graphs and surprising challenges of graph processing: extended survey. *VLDB J.*, 29(2-3):595–618, 2020. doi:10.1007/s00778-019-00548-X.
- 84 Sherif Sakr, Angela Bonifati, Hannes Voigt, Alexandru Iosup, Khaled Ammar, Renzo Angles, Walid G. Aref, Marcelo Arenas, Maciej Besta, Peter A. Boncz, Khuzaima Daudjee, Emanuele Della Valle, Stefania Dumbrava, Olaf Hartig, Bernhard Haslhofer, Tim Hegeman, Jan Hidders, Katja Hose, Adriana Iamnitchi, Vasiliki Kalavri, Hugo Kapp, Wim Martens, M. Tamer Özsu, Eric Peukert, Stefan Plantikow, Mohamed Ragab, Matei Ripeanu, Semih Salihoglu, Christian Schulz, Petra Selmer, Juan F. Sequeda, Joshua Shinavier, Gábor Szárnyas, Riccardo Tommasini, Antonino Tumeo, Alexandru Uta, Ana Lucia Varbanescu, Hsiang-Yun Wu, Nikolay Yakovets, Da Yan, and Eiko Yoneki. The future is big graphs: a community view on graph processing systems. *Commun. ACM*, 64(9):62–71, 2021. doi:10.1145/3434642.
- 85 Cristina Sarasua, Alessandro Checco, Gianluca Demartini, Djellel Eddine Difallah, Michael Feldman, and Lydia Pintscher. The Evolution of Power and Standard Wikidata Editors: Comparing Editing Behavior over Time to Predict Lifespan and Volume of Edits. *Comput. Support. Cooperative Work.*, 28(5):843–882, 2019. doi:10.1007/S10606-018-9344-Y.
- 86 Johannes Schrott, Maxime Jakubowski, and Katja Hose. A Graph-Native Approach to Normalization. *CoRR*, abs/2603.02995, 2026. doi:10.48550/arXiv.2603.02995.
- 87 Xudong Shen, Hannah Brown, Jiashu Tao, Martin Strobel, Yao Tong, Akshay Narayan, Harold Soh, and Finale Doshi-Velez. Directions of technical innovation for regulatable AI systems. *Commun. ACM*, 67(11):82–89, 2024. doi:10.1145/3653670.
- 88 Valerie J. Shute, Chen Sun, and Jodi Asbell-Clarke. Demystifying computational thinking. *Educational Research Review*, 22:142–158, 2017. doi:10.1016/j.edurev.2017.09.003.
- 89 Sophia Sideri, Vasilis Efthymiou, Dimitris Plexousakis, and Haridimos Kondylakis. PG2RDF: schema-guided transformation of property graphs to RDF. In *IJCKG*, Lecture Notes in Computer Science, pages 358–373. Springer, 2025. doi:10.1007/978-981-95-5009-8\_24.
- 90 Shani Evenstein Sigalov and Rafi Nachmias. Investigating the potential of the semantic web for education: Exploring Wikidata as a learning platform. *Educ. Inf. Technol.*, 28(10):12565–12614, 2023. doi:10.1007/S10639-023-11664-1.
- 91 Harold R. Solbrig, Eric Prud’hommeaux, Graham Grieve, Lloyd McKenzie, Joshua C. Mandel, Deepak K. Sharma, and Guoqian Jiang. Modeling and validating HL7 FHIR profiles using semantic web shape expressions (ShEx). *J. Biomed. Informatics*, 67:90–100, 2017. doi:10.1016/J.JBI.2017.02.009.
- 92 John F. Sowa. Conceptual graphs for a data base interface. *IBM J. Res. Dev.*, 20(4):336–357, 1976. doi:10.1147/RD.204.0336.
- 93 Slawek Staworko, Iovka Boneva, José Emilio Labra Gayo, Samuel Hym, Eric G. Prud’hommeaux, and Harold R. Solbrig. Complexity and expressiveness of ShEx for RDF. In *ICDT, LIPIcs*, pages 195–211. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.ICDT.2015.195.
- 94 Radu Stoica, George H. L. Fletcher, and Juan F. Sequeda. On directly mapping relational databases to property graphs. In *AMW, CEUR Workshop Proceedings*. CEUR-WS.org, 2019. URL: <https://ceur-ws.org/Vol-2369/short06.pdf>.
- 95 John C. Thomas and John D. Gould. A psychological study of query by example. In *AFIPS Na-*



- tional Computer Conference*, AFIPS Conference Proceedings, pages 439–445. AFIPS Press, 1975. doi:10.1145/1499949.1500035.
- 96 Hernán Vargas, Carlos Buil-Aranda, Aidan Hogan, and Claudia López. A user interface for exploring and querying knowledge graphs (extended abstract). In *IJCAI*, pages 4785–4789. ijcai.org, 2020. doi:10.24963/IJCAI.2020/666.
  - 97 Eliane Zambon Victorelli, Júlio Cesar dos Reis, Heiko Horst Hornung, and Alysson Bolognesi Prado. Understanding human-data interaction: Literature review and recommendations for design. *Int. J. Hum. Comput. Stud.*, 134:13–32, 2020. doi:10.1016/J.IJHCS.2019.09.004.
  - 98 Denny Vrandečić. Wikidata: a new platform for collaborative data collection. In *WWW (Companion Volume)*, pages 1063–1064. ACM, 2012. doi:10.1145/2187980.2188242.
  - 99 Domagoj Vrgoc, Carlos Rojas, Renzo Angles, Marcelo Arenas, Diego Arroyuelo, Carlos Buil-Aranda, Aidan Hogan, Gonzalo Navarro, Cristian Riveros, and Juan Romero. MillenniumDB: An open-source graph database system. *Data Intell.*, 5(3):560–610, 2023. doi:10.1162/DINT\_A\_00229.
  - 100 Andra Waagmeester, Lynn Schriml, and Andrew Su. Wikidata as a linked-data hub for biodiversity data. *Biodiversity Information Science and Standards*, 3, 2019. doi:10.3897/biss.3.35206.
  - 101 Lucas Werkmeister. Schema inference on Wikidata. Master’s thesis, Karlsruhe Institute of Technology, 2018. URL: <https://github.com/lucaswerkmeister/master-thesis/releases/download/final/master-thesis-Lucas-Werkmeister.pdf>.
  - 102 Wikidata Community. Data quality days 2021. [https://www.wikidata.org/wiki/Wikidata:Events/Data\\_Quality\\_Days\\_2021](https://www.wikidata.org/wiki/Wikidata:Events/Data_Quality_Days_2021), 2021. Accessed: 2026-02-16.
  - 103 Wikidata Community. Data quality days 2022. [https://www.wikidata.org/wiki/Wikidata:Events/Data\\_Quality\\_Days\\_2022](https://www.wikidata.org/wiki/Wikidata:Events/Data_Quality_Days_2022), 2022. Accessed: 2026-02-16.
  - 104 Wikidata Community. Data modelling days 2023. [https://www.wikidata.org/wiki/Wikidata\\_talk:Events/Data\\_Modelling\\_Days\\_2023](https://www.wikidata.org/wiki/Wikidata_talk:Events/Data_Modelling_Days_2023), 2023. Accessed: 2026-02-16.
  - 105 Wikimedia Foundation. Wikidata: Data access. [https://www.wikidata.org/wiki/Wikidata:Data\\_access](https://www.wikidata.org/wiki/Wikidata:Data_access), 2026. Accessed: 2026-02-16.
  - 106 Wikimedia Foundation. Wikidata: Licensing. <https://www.wikidata.org/wiki/Wikidata:Licensing>, 2026. Accessed: 2026-02-16.
  - 107 Wikimedia Foundation. Wikidata query service. <https://query.wikidata.org/>, 2026. Accessed: 2026-02-16.
  - 108 Katy Williams, Alex Bigelow, and Katherine E. Isaacs. Data abstraction elephants: The initial diversity of data representations and mental models. In *CHI*, pages 803:1–803:24. ACM, 2023. doi:10.1145/3544548.3580669.
  - 109 Bingjun Xie, Jia Zhou, and Huilin Wang. How influential are mental models on interaction performance? exploring the gap between users’ and designers’ mental models through a new quantitative method. *Adv. Hum. Comput. Interact.*, 2017:3683546:1–3683546:14, 2017. doi:10.1155/2017/3683546.
  - 110 Vahan Yeghouchian, Daniel Archambault, Stephan Diehl, Tim Dwyer, Karsten Klein, Helen C. Purchase, and Hsiang-Yun Wu. Exploring the limits of complexity: A survey of empirical studies on graph visualisation. *Vis. Informatics*, 2(4):264–282, 2018. doi:10.1016/J.VISINF.2018.12.006.
  - 111 Lingfeng Zhong, Jia Wu, Qian Li, Hao Peng, and Xindong Wu. A comprehensive survey on automatic knowledge graph construction. *ACM Comput. Surv.*, 56(4):94:1–94:62, 2024. doi:10.1145/3618295.
  - 112 Moshé M. Zloof. Query-by-Example: the Invocation and Definition of Tables and Forms. In *VLDB*, pages 1–24. ACM, 1975. doi:10.1145/1282480.1282482.



# Transforming Shape Schemas with Composable Property-Graph Queries

Philipp Seifer 

The Software Languages Team, University of Koblenz, Germany

Daniel Hernández 

Institute for Artificial Intelligence, University of Stuttgart, Germany

Ralf Lämmel 

The Software Languages Team, University of Koblenz, Germany

Steffen Staab 

Institute for Artificial Intelligence, University of Stuttgart, Germany

Web and Internet Science Research Group, University of Southampton, UK

---

## Abstract

Property graphs may be constrained by schemas that inform both query engines and human users about the shape of valid data, enforcing a contract between data provider and consumer. Composable property-graph queries transform input graphs into output graphs. Then, the question arises of which schema can be expected after one (or several) transformation steps. We investigate how schema constraints can be inferred given an input schema and a transforming query. Specifically, we propose a reasoning procedure that, given an input schema in ProGS and a query in G-CORE infers an output schema. Since graph updates will happen frequently, our inference procedure does not rely on graph instances, such that the computed output schema applies to all graphs originating from any input graph complying with the input schema. Related work has addressed this problem for SPARQL CONSTRUCT queries, encoding it in De-

scription Logics (DLs) so that the output schema is entailed by axioms inferred from input schema and queries. Property graphs and their queries, however, complicate the matter, as property graphs feature label and property annotations as well as first-class edges. Thus, reification has to be used in one way or another, though available DLs lack the means to encode such features directly. We approach this novel challenge via a family of mappings for i) property graphs reified in RDF, aligned with ii) a mapping from ProGS to SHACL and iii) a mapping from G-CORE to SPARQL CONSTRUCT queries. In this manner, schema inference for property graphs becomes manageable, as we break apart the problem through the extra mapping layer and utilize efficient DL reasoners. We develop the metatheory regarding the soundness of inferred schema constraints and the semantic equivalence of mapped schemas and queries.

**2012 ACM Subject Classification** Information systems → Graph-based database models; Information systems → Resource Description Framework (RDF); Information systems → Query languages; Information systems → Extraction, transformation and loading

**Keywords and phrases** Property Graphs, Validation, Schema Inference, Reification, G-CORE, SPARQL

**Digital Object Identifier** 10.4230/TGDK.4.2.3

**Category** Research

**Related Version** *Extended Version*: <http://arxiv.org/abs/2606.14309>

**Supplementary Material** *Software (Source Code)*: <https://github.com/softlang/s2s> [40]  
archived at `swb:1:dir:a38354c2cab3efb8abdc9e80f611d19aa3531d82`

**Funding** *Daniel Hernández*: Deutsche Forschungsgemeinschaft (DFG) – SFB 1574 – 471687386.

**Received** 2025-07-01 **Accepted** 2026-04-03 **Published** 2026-09-03

**Editors** Stefania Dumbrava, George Fletcher, Matteo Lissandrini, and Riccardo Tommasini

**Special Issue** Data Management for (Knowledge) Graphs

**Generative AI Declaration** Generative AI was not used in the preparation of this article.



© Philipp Seifer, Daniel Hernández, Ralf Lämmel, and Steffen Staab;  
licensed under Creative Commons License CC-BY 4.0

*Transactions on Graph Data and Knowledge*, Vol. 4, Issue 2, Article No. 3, pp. 3:1–3:29



Transactions on Graph Data and Knowledge

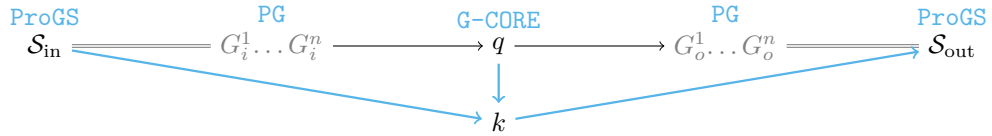
TGDK Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Introduction

Composable property-graph queries transform graphs into graphs. In this paper, we are interested in schema inference for the results of such composable queries. For RDF graphs, the W3C proposes a composable query language, namely SPARQL [18] with its **CONSTRUCT** [27] clause. For *property graphs* (PGs), which we aim at in this paper, the recent GQL [21] standard is not composable (as yet [23]). Accordingly, we leverage G-CORE [3], an earlier *composable* query language for property graphs that inspired GQL.

Property-graph schemas are used to constrain PGs; the constraints can be enforced by validation. There are schema languages, sometimes also called *shape* languages, for PGs, such as the *Property Graph Shapes Language* (ProGS) [44] or PG-Schema [4]. As far as schema constraints of the inference approach in this paper are concerned, both ProGS and PG-Schema are viable candidates. Pragmatically, we commit to ProGS, as it is more directly aligned with the established schema language for RDF, i.e. SHACL [26], which is readily covered by the foundation we rely on. In particular, the correspondence between SHACL and description logics (DL) is well understood [41, 8]. Thus, we can leverage efficient reasoner implementations for inference. We follow the terminology of ProGS and SHACL and refer to schemas as *shapes*.

We refer to shapes validating all possible result graphs of a query as *output shapes*. Similarly, we use the term *input shapes* to refer to the set of shapes over the input graphs of a query. The inference of output shapes is a challenging aspect of the composable query notion: input shapes do not carry over directly to the output but do indeed impact it. Additionally, to obtain sound shapes for all possible query results, inference must rely on the *input shapes* and the *query* as inputs rather than individual result graphs. Access to these statically inferred shapes facilitates numerous use cases, even when considering compositions of multiple queries: they enable tools such as type systems for language-embedded queries (compare [30]) among others (e.g., [50, 35]) that rely on shapes. For query pipelines that are executed repeatedly, such shapes need only be inferred once, which is particularly useful in cases where some level of manual review or selection of shapes is desired. Finally, even in data transformation or integration use cases, where result graphs are stored in databases, they complement shapes inferred from graph instances and can provide insights to developers without query execution.

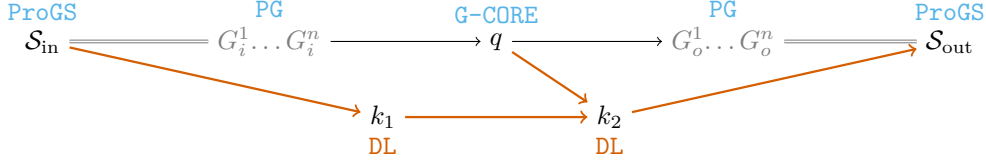


■ **Figure 1** Schematic overview of the problem: Input shapes  $\mathcal{S}_{\text{in}}$  validate a (possibly infinite) set of input graphs  $G_i^1 \dots G_i^n$  which are valid inputs for the query  $q$ . The query produces, as its result, a set of corresponding result graphs  $G_o^1 \dots G_o^n$ . Just as  $\mathcal{S}_{\text{in}}$  validates all input graphs,  $\mathcal{S}_{\text{out}}$  validates all output graphs. The set of  $\mathcal{S}_{\text{out}}$  is constructed from some formalization  $k$ , which is derived from  $\mathcal{S}_{\text{in}}$  and  $q$ .

Figure 1 shows the input (shapes  $\mathcal{S}_{\text{in}}$  and query  $q$ ) and the output (shapes  $\mathcal{S}_{\text{out}}$ ) of shape inference, with the respective languages annotated in blue. For these languages, we will later define the relevant subsets we consider in this work. The figure also hints in gray at the set of input graphs (conforming to shapes  $\mathcal{S}_{\text{in}}$ , indicated by gray double lines) and corresponding output graphs (conforming to  $\mathcal{S}_{\text{out}}$ ), with black arrows representing input and output of query execution. These graphs are included in the figure only for the sake of clarity; our approach does not depend on any concrete graphs. Our approach is rendered using blue arrows: we encode in  $k$  the information that allows us to deduce which shapes are in  $\mathcal{S}_{\text{out}}$ .



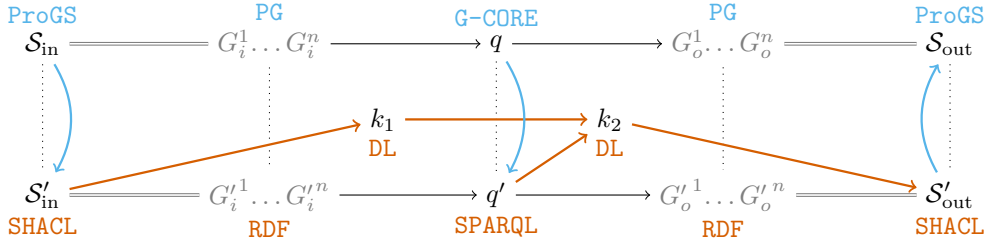
To introduce  $k$ , we establish a common language that can express the constraints of the input shapes  $\mathcal{S}_{\text{in}}$  and how inputs and outputs relate to each other over the query  $q$ . Given such a language, we then define an algorithm for constructing  $k$  from both  $\mathcal{S}_{\text{in}}$  and the query  $q$  and for inferring the new shapes in  $\mathcal{S}_{\text{out}}$  from  $k$ .



■ **Figure 2** Schematic overview of a possible solution where constraints are encoded in description logics.

One possibility for encoding these constraints is *description logics* (DL). Figure 2 indicates where DL is used with red arrows: we first encode the input shapes  $\mathcal{S}_{\text{in}}$  as a set of DL axioms and then combine them with a second set of axioms inferred from the query  $q$ . This approach benefits from relying on an existing formalism and, in practical applications, from established and efficient reasoners for implementing the required decision procedures. In prior work [41], we also relied on a DL to solve a similar inference problem for RDF graphs and SPARQL CONSTRUCT queries. However, there is a drawback: Common DLs do not support properties with key-value annotations. Thus, we would have to either define a new variant – for which no efficient reasoners exist – or we have to use some form of reification to encode properties in an existing DL.

Since the first option is not desirable, we choose the second. However, instead of encoding reification directly in DL, we introduce an intermediate RDF layer. This provides the following benefits: foremost, it clearly separates the *reification* from the *DL encoding and inference* steps of our solution, making each part easier to formalize, prove, and implement. Secondly, we can build on the aforementioned prior work for the encoding and inference step; we still need to modify and extend the algorithm. Finally, a mapping between the involved shape and query languages has not been formally explored, which we achieve in this work.



■ **Figure 3** Schematic overview of the involved languages, mappings, and algorithms. The upper PG layer is mapped to the bottom RDF layer, where individual mappings are indicated using dotted lines. Inference and reasoning exclusively operate on RDF. The blue arrows show how mappings are utilized.

Figure 3 shows the full solution we propose. Utilizing reification (indicated with dotted lines) at all levels allows us to step down from a PG-based language family (upper half of Figure 3) to an RDF-based one (lower half of Figure 3). We use RDF for representing the data by reifying first-class edges with identities and properties in PG as ordinary RDF triples. We use SHACL as the shape language, accounting for this reification. This allows us to rely on DL for the formalization and implementation of shape inference, since round tripping between DL axioms and SHACL shapes is possible (cf. [41, 8]). Finally, we use SPARQL CONSTRUCT as the composable query language to which we map. Here, the definition of a sound mapping is more complex, since in addition to the reification, we must also account for the differences in query language semantics:

where labels and properties are copied by default in G-CORE and explicitly removed, SPARQL relies on explicit triple patterns and filters. The data reification, shape-to-DL mapping, and shape inference approaches have been studied [24, 4, 41], whereas the query mapping has not.

We build on and extend our existing algorithm [41] that infers SHACL shapes for results of SPARQL `CONSTRUCT` queries on RDF graphs by constructing a DL knowledge base entailing valid output shapes. To define our query mapping, we must extend the SPARQL fragment considered in [41] with more *generic* triple patterns where variables extend over concept or role names to capture the semantic properties of G-CORE. This leads to a number of required adaptations of the method presented in this work, since the knowledge base must encode the potential presence of previously unexpected concept or role names. The dataflow of the extension to [41] is indicated by red arrows in Figure 3. It is important to note that our mapping approach serves the formalization and implementation of schema inference; whether the queries are executed on a PG database or an RDF store is a separate question.

## 1.1 Contributions

In summary, the contributions of this paper are as follows:

1. We determine the impact of PG-specific expressiveness when it comes to shape inference for composable graph queries. Thus, we handle edges that have identities, as well as nodes and edges annotated with labels and key-value properties that persist, implicitly, in results.
2. Despite common DLs lacking the corresponding language constructs, we resolve the mismatch through reification, which carries over to structures of shapes and queries.
3. Instead of reification at the DL level, we define an intermediate layer, introducing mappings from ProGS to SHACL and G-CORE to SPARQL `CONSTRUCT` over PGs reified in RDF.
4. We develop the metatheory regarding the soundness of inferred shapes and the semantic equivalence of mapped shapes and queries.

## 1.2 Roadmap of the Paper

Section 2 introduces the underlying formalisms, as well as our running examples. In Section 3, we formalize the essential problem addressed in this work. Section 4 and Section 5 solve the problem in two steps: first, we develop a family of mappings from property graph abstractions to RDF graph abstractions (Section 4). On these grounds, we develop an inference method, which is an extension of [41] (Section 5). Section 6 develops the corresponding metatheory. In Section 7, we briefly outline our prototype implementation of this approach. Finally, Section 8 discusses related work while Section 9 concludes the paper.

In several places we refer to an extended version of the paper, referenced in the information section on the first page, which includes additional examples as well as full proofs.

# 2 Foundations

We first introduce the graph data models, query languages, and shape-based validation languages relevant to our work; they can be separated into two groups, namely *PG-related* languages and *RDF-related* languages pertaining to the upper and lower layers of Figure 3.

## 2.1 RDF Graphs

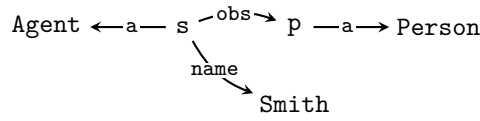
We define RDF graphs based on the W3C specification [13], excluding blank nodes and literals; we consider the latter as ordinary IRIs to simplify the formalization of the data model. We assume four pairwise disjoint, finite<sup>1</sup> but arbitrary, and sufficiently large sets of IRIs, namely concept

<sup>1</sup> This is not a restriction to the problem, since the sets are arbitrary, but it simplifies a few definitions.

names **C**, individual names **I**, role names **R**, and the special role name  $\{\text{rdf:type}\}$ . For the sake of consistency, we use description-logic terminology throughout all abstractions (e.g., *concept name* instead of *class*). Commonly, RDF uses so-called prefixes to abbreviate (parts of) IRIs. In formal definitions, we express most IRIs through ordinary symbols, with a few exceptions, such as `rdf:type`. In examples, we also omit prefixes for the example domain; that is, we simply write `Agent` instead of `ex:Agent`. Similarly, we sometimes write `a` instead of `rdf:type`.

► **Definition 1** (Simple RDF Graph). *An RDF graph  $G$  is a finite set of triples of the form  $(a, p, b)$  or  $(a, \text{rdf:type}, A)$  where  $a, b \in \mathbf{I}$ ,  $A \in \mathbf{C}$ , and  $p \in \mathbf{R}$ .*

► **Example 2.** Consider the following set of triples as an example RDF graph  $G_1^{\text{rdf}} = \{(s, a, \text{Agent}), (s, \text{name}, \text{Smith}), (p, a, \text{Person}), (s, \text{obs}, p)\}$ .



### 2.1.1 SPARQL Queries

We define a subset of SPARQL queries referred to as *Extended Conjunctive CONSTRUCT Queries* (ECCQ) in analogy to *Simple Conjunctive CONSTRUCT Queries* (as defined in [41]) in Definition 3. We choose this conjunctive fragment, as is commonly done, to keep the scope of our algorithms manageable; we extend<sup>2</sup> the fragment of SPARQL chosen in [41] with the language features required for our intended mappings.

To this end, we introduce *generic* patterns such as  $x:y$ , written  $?x \text{ a } ?y$  in concrete SPARQL syntax, where both  $x$  and  $y$  are variables. Here, we need to introduce an additional syntactic restriction for generic variables, as we will discuss in more depth later: if the query includes the atomic query pattern  $x:y$ , then  $y$  must not occur again in another atomic query pattern. That is, we can copy all concept names of  $x$  generically but cannot introduce restrictions on this meta level; for example, a query with patterns  $\{x:y, z:y\}$  is not allowed, since it would constrain the bindings of  $x$  and  $z$  through  $y$ . Thus, we write  $x:\bar{x}^C$  to indicate that the generic variable  $\bar{x}^C$  is syntactically dependent on  $x$  in this way. We also introduce certain filter expressions for only these generic variables written  $\bar{x}^C$ , filtering certain concept or role names from their bindings.

► **Definition 3** (Syntax of ECCQ). *We define atomic patterns  $t$  and filter expressions  $f$  as follows:*

$$t ::= x:A \mid (x,y):p \mid (x,a):p \mid x:\bar{x}^C \mid (x,\bar{x}^I):\bar{x}^R \quad f ::= \bar{x}^C \neq A \mid \bar{x}^R \neq p$$

with  $A \in \mathbf{C}$ ,  $a \in \mathbf{I}$ ,  $p \in \mathbf{R}$ , and variables  $x, y, \bar{x}^C, \bar{x}^R, \bar{x}^I \in \mathbf{V}$ . Let the function `vars` denote the set of all variables that occur in a set of atomic patterns or filter expressions. An ECCQ  $q$  is defined as  $H \leftarrow (P, F)$  where template  $H$  and pattern  $P$  consist of finite sets of atomic patterns  $t$  and the filters  $F$  of a finite set of filter expressions  $f$  such that  $\text{vars}(F) \subseteq \text{vars}(P)$ . For each atomic pattern  $x:\bar{x}^C$  (and similarly  $(x,\bar{x}^I):\bar{x}^R$ ), there is a unique variable  $\bar{x}^C$  (similarly  $\bar{x}^R$  and  $\bar{x}^I$ ) per ordinary variable  $x$ , which does not occur again in the query, except in filter conditions. If a generic atomic pattern of this form occurs in  $H$ , the same pattern must also occur in  $P$ .

<sup>2</sup> ECCQ is not a true extension of the language presented in [41], since we omit certain patterns, e.g.,  $a:C$ . While they could be easily included, making ECCQ a true extension of this language, they are not required for the remainder of this work.

To formalize the semantics of ECCQ in Definition 4, we define valuations  $\mu : \mathbf{V} \rightarrow \mathbf{I} \cup \mathbf{C} \cup \mathbf{R}$  as mappings from variables to individual, concept, and role names. Note that the syntactic restrictions on  $\bar{x}^{\mathbf{C}}$  and  $\bar{x}^{\mathbf{R}}$  ensure that the respective types of these bindings cannot be mixed up, since these variables can only occur in the correct context. Two valuations  $\mu_1$  and  $\mu_2$  are *compatible*, denoted  $\mu_1 \sim \mu_2$ , if for every variable  $x$  occurring in both  $\mu_1$  and  $\mu_2$ ,  $\mu_1(x) = \mu_2(x)$ . A finite set of valuations is denoted  $\Omega$ . We define the *join* of two sets of valuations  $\Omega_1$  and  $\Omega_2$  as  $\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2, \mu_1 \sim \mu_2\}$ .

Note that in Definition 3 we do not include the usual syntactic restriction  $\text{vars}(H) \subseteq \text{vars}(P)$ . We therefore need the final clause in Definition 4 about variables from atomic patterns  $t \in H$  that do not occur in  $\text{vars}(P)$ . Here, we assume the generation of fresh IRIs for such variables, which will be required by our mappings later. While this is a deviation from ordinary SPARQL semantics<sup>3</sup>, our variant simplifies the formalization by avoiding the need to represent blank nodes in our RDF graph model and throughout the paper. In practical implementations, ordinary blank nodes should be used to comply with standard SPARQL semantics; indeed, our implementation generates SPARQL queries featuring blank node identifiers in their templates as well.

► **Definition 4** (Semantics of ECCQ). *The result of evaluating an ECCQ  $H \leftarrow (P, F)$  over an RDF graph  $G$  is the RDF graph denoted  $\llbracket H \leftarrow (P, F) \rrbracket_G^{\text{eccq}}$  and defined as*

$$\llbracket H \leftarrow (P, F) \rrbracket_G^{\text{eccq}} = \{\mu(t) \mid \mu \in \llbracket P, F \rrbracket_G^{\text{eccq-p}}, t \in H\}$$

where  $\llbracket P, F \rrbracket_G^{\text{eccq-p}} = \{\mu \mid \mu \in \bowtie_{t \in P} \llbracket t \rrbracket_G^{\text{eccq-a}}, \text{ for all } f \in F : \mu \vdash f\}$  and  $\llbracket t \rrbracket_G^{\text{eccq-a}} = \{\mu \mid \mu(t) \in G\}$ .  $f$  is a filter condition such that  $\mu \vdash (\bar{x}^{\mathbf{C}} \neq A)$  is true iff  $\mu(\bar{x}^{\mathbf{C}})$  is not  $A$ , and  $\mu \vdash (\bar{x}^{\mathbf{R}} \neq p)$  is true iff  $\mu(\bar{x}^{\mathbf{R}})$  is not  $p$ . In a slight abuse of notation, we write  $\mu(t)$  to mean  $\mu(u:A) = (\mu(u), \text{rdf:type}, A)$ , and similarly for the remaining cases. If a variable  $x$  in  $t \in H$  does not occur in  $\text{vars}(P)$ , then  $\mu(x)$  produces a fresh IRI. This IRI is unique for each variable  $x$  and each unique mapping  $\mu$ .

► **Example 5.** Consider the ECCQ  $q_1^{\text{eccq}} = \{x:\text{Agent}, y:\text{POI}\} \leftarrow (\{(x,y):\text{obs}\}, \{\})$ , selecting pairs of observers and observed before constructing a new graph labeling observers as **Agent** and observed as **POI**. When evaluating this query on graph  $G_1^{\text{rdf}}$  from Example 2, written  $\llbracket q_1^{\text{eccq}} \rrbracket_{G_1^{\text{rdf}}}$ , we first match the pattern  $P$  on the graph, resulting in the single mapping  $\mu$  where  $\mu(x) = s$  and  $\mu(y) = p$ . Thus, the template  $H$  is instantiated only once, namely for  $\mu$ , resulting in the graph  $\{(s, a, \text{Agent}), (p, a, \text{POI})\}$ .

Using generic patterns, we could, in this particular case, rewrite the query as  $\{x:\bar{x}^{\mathbf{C}}, y:\text{POI}\} \leftarrow (\{x:\bar{x}^{\mathbf{C}}, (x,y):\text{obs}\}, \{\})$ , copying all concepts for bindings for  $x$  instead of explicitly adding **Agent**. Since  $s$  has only the concept **Agent** in  $q_1^{\text{eccq}}$ , evaluation would yield the same result graph.

$s \text{ --- } a \rightarrow \text{Agent}$

$p \text{ --- } a \rightarrow \text{POI}$

<sup>3</sup> Usually, SPARQL CONSTRUCT templates can feature blank node identifiers (<https://www.w3.org/TR/sparql12-query/#templatesWithBNodes>) to this effect.

### 2.1.2 SHACL Shapes

SHACL is a validation language for RDF graphs, where a shape consists of two components: the target query selecting a subset of nodes and a constraint these nodes must conform to. We present here the Seifer et al. [41] definition of SHACL shapes, utilizing description logics to define the semantics of target queries and constraints based on work by Bogaerts et al. [8].

According to this definition, SHACL shapes are  $\mathcal{ALCHOI}$  axioms, i.e. target query subsumed by the constraint, both encoded as concept descriptions (Definition 6), and their semantics can be reduced to checking consistency with a *validation knowledge base* constructed for a given RDF graph  $G$  by its unique-name, closed-world, and domain-closure assumptions. We refer to [7] for the definition of the semantics of  $\mathcal{ALCHOI}$ . We give here the syntax of  $\mathcal{ALCHOI}$  axioms, which can express a subset of SHACL, and the core idea of the validation semantics introduced in [41]; we refer to that paper for the full derivation and proofs.

► **Definition 6** ( $\mathcal{ALCHOI}$  Axioms). Concept descriptions are defined as follows:

$$\begin{aligned} C &::= \top \mid \perp \mid A \mid \{a\} \mid C \sqcap C \mid C \sqcup C \mid \neg C \mid \exists p.C \mid \forall p.C \\ \rho &::= p \mid p^- \end{aligned}$$

where  $A$ ,  $a$ , and  $p$  stand for concept names, individual names, and role names, respectively, and  $\top$  and  $\perp$  are two special concept names. Given two concept descriptions  $C$  and  $D$ , and two role descriptions  $\rho_1$  and  $\rho_2$ ,  $C \sqsubseteq D$  and  $\rho_1 \sqsubseteq \rho_2$  are  $\mathcal{ALCHOI}$  axioms, which we also call shapes. The left-hand side ( $C$  and  $\rho_1$ ) we also call the target and the right-hand side ( $D$  and  $\rho_2$ ) the constraint of the shape.

Semantically, SHACL shapes target specific nodes of a graph, relying on constraints to validate these selected nodes. A consistency check for the subsumption axioms defined in Definition 6 with the validation knowledge base constructed for a given RDF graph  $G$  (Definition 7) encodes these semantics. Example 8 demonstrates the intuition.

► **Definition 7** (Validation Semantics (Adapted from [41])). The axioms of a simple RDF graph  $G$ , denoted  $\mathcal{T}_G$ , are the TBox consisting of the following  $\mathcal{ALCHOI}$  axioms:

1. Domain Closure Assumption (DCA):  $\top \equiv \bigsqcup_{a \in \mathbf{I}} \{a\}$ .
  2. Unique Name Assumption (UNA):  $\{a\} \sqcap \{b\} \equiv \perp$ , for each pair of distinct individual names  $a, b \in \mathbf{I}$ .
  3. Closed-World Assumption (CWA):
    - $A \equiv \bigsqcup_{a: A \in G} \{a\}$ , for each concept name  $A \in \mathbf{C}$ ,
    - $\exists p.\{a\} \equiv \bigsqcup_{(b,a): p \in G} \{b\}$ , and
    - $\exists p^-\{a\} \equiv \bigsqcup_{(a,b): p \in G} \{b\}$ , for each role name  $p \in \mathbf{R}$  and each individual name  $a \in \mathbf{I}$ .
- $(\mathcal{T}_G, G)$  is the validation knowledge base of  $G$ . A graph  $G$  is proof-valid according to a set  $\Sigma$  of  $\mathcal{ALCHOI}$  axioms if and only if  $\Sigma$  is consistent with the validation knowledge base of  $G$ , i.e. the knowledge base  $(\mathcal{T}_G \cup \Sigma, G)$  admits a model.

We write  $\text{valid}(G, S)$  to indicate that  $G$  is proof-valid regarding the set of shapes  $S$  and  $\text{valid}(G, s)$  to indicate that  $G$  is proof-valid regarding a single shape  $s$ . We use  $S$  and  $\Sigma$  interchangeably. In Example 8 we give an example shape and outline both the intuitive reason why this shape validates the running example graph and the formal reason considering the validation knowledge base.

► **Example 8.** Consider  $s_1^{\text{shacl}} = \text{Agent} \sqsubseteq \exists \text{obs}.\text{Person}$  as an example. Intuitively, the shape expresses that all agents must observe at least one person. The graph  $G_1^{\text{rdf}}$  (Example 2) is valid regarding  $s_1^{\text{shacl}}$  since all targets, i.e.  $s$ , which is an instance of **Agent**, conform to the constraint, given both  $(s, \text{obs}, p) \in G_1^{\text{rdf}}$  and  $(p, a, \text{Person}) \in G_1^{\text{rdf}}$ .

The validation knowledge base, constructed from the closed-world ( $\text{Agent} \equiv \{s\}$ ,  $\text{Person} \equiv \{p\}$ ,  $\exists \text{obs}. \{p\} \equiv \{s\}$ , and  $\exists \text{obs}^-. \{s\} \equiv \{p\}$ ), domain-closure ( $\{s\} \sqcup \{p\} \equiv \top$ ), and unique-name assumptions ( $\{s\} \sqcap \{p\} \equiv \perp$ ) for  $G_1^{rdf}$  is consistent with this subsumption axiom, too: essentially, there is the concept  $\{s\}$  which is equal to  $\text{Agent}$ , and also the range of  $\text{obs}$ ; in turn, the domain of  $\text{obs}$  is  $\{p\}$ , which is also equal to  $\text{Person}$ .

## 2.2 Property Graphs

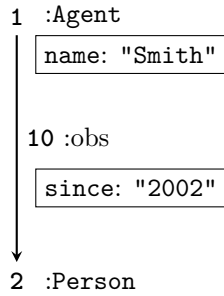
Property graphs (Definition 9) differ from RDF graphs in that edges have identities, and both nodes and edges can be annotated with sets of labels and sets of key-value pairs, the so-called properties. There are various notions of property graphs; we base our work largely on [3], that is, the graph model used for G-CORE, which is very similar to the graph model used in the context of Cypher<sup>4</sup> as well. We do not include first-class paths, however.

We define a set of labels  $L = L_N \cup L_E$  where  $L_N$  is a finite set of node labels and  $L_E$  a finite set of edge labels; a finite set of property names (or keys)  $K = K_N \cup K_E$  where  $K_N$  is a finite set of node keys and  $K_E$  a finite set of edge keys; and a finite set of literal values  $V$ . Without loss of generality, we assume that literal values are strings. Finally, let  $\text{FS}(s)$  denote all finite subsets of a set  $s$ , including the empty set. See Example 10 for the running example PG.

► **Definition 9** (Property Graph). A property graph ( $PG$ ) is a tuple  $G = (N, E, \rho, \lambda, \sigma)$ .  $N$  and  $E$  denote two disjoint, finite sets of node and edge identifiers, respectively. We write  $n \in N$ ,  $e \in E$ , and  $u \in N \cup E$ .  $\rho : E \rightarrow (N \times N)$  is a total function assigning edges to nodes;  $\lambda : (N \cup E) \rightarrow \text{FS}(L)$  is a total function assigning labels to nodes or edges; and  $\sigma : (N \cup E) \times K \rightarrow \text{FS}(V)$  is a total function assigning properties to nodes or edges and for which a set of tuples  $(x, k) \in (N \cup E) \times K$  exists such that  $\sigma(x, k) \neq \emptyset$ .

► **Example 10.** Consider  $G_1^{pg} = (N_1, E_1, \rho_1, \lambda_1, \sigma_1)$ , where  $N_1 = \{1, 2\}$  and  $E_1 = \{10\}$  are the sets of node and edge IDs. A single edge is defined as  $\rho_1(10) = (1, 2)$ . Labels are defined as  $\lambda_1(1) = \{\text{Agent}\}$ ,  $\lambda_1(2) = \{\text{Person}\}$ , and  $\lambda_1(10) = \{\text{obs}\}$ . Finally, we give the non-empty cases for the property function as  $\sigma_1(1, \text{name}) = \{\text{"Smith"}\}$  and  $\sigma_1(10, \text{since}) = \{\text{"2002"}\}$ .

Compare this graph to the RDF graph  $G_1^{rdf}$  in Example 2: RDF graphs do not support properties on edges; thus, there is no direct equivalent for  $\sigma_1(10, \text{since}) = \{\text{"2002"}\}$  in RDF. Otherwise, the graphs are equivalent.



### 2.2.1 Property Graph Queries

Composable property graph queries return graphs and function similarly to ECCQ queries. We define *Simple G-CORE Queries* (SGCQ) as a subset of G-CORE [3]. Our conjunctive fragment (Definition 11) supports matching multiple atomic patterns, including the most common constraints

<sup>4</sup> <https://opencypher.org/>



on labels and properties, and constructing a new graph from multiple atomic patterns while explicitly adding or removing labels and properties. In short, the fragment allows for selecting a subgraph, reshaping it, and adding constant elements. It does not support whole-graph operations, such as the union of a constructed graph with the original graph.

The basic semantics of SGCQ are similar to ECCQ: a *pattern*  $\Gamma$  defines variables and constrains their bindings through clauses  $\xi$ . These bindings are used to construct a new graph, specified in the *template*  $\mathcal{B}$ , together with *set* ( $\oplus$ ) and *remove* ( $\ominus$ ) clauses for explicitly adding or removing labels and properties. Unlike in the case of ECCQ, labels and properties persist in the result graph by default.

We give here in short the syntax and semantics of SGCQ adapted from [3], with the following change: since we allow only for a subset of filter expressions, we push these filters into the node (or edge) pattern. This is equivalent to the concrete syntax of G-CORE where we would write, e.g.,  $(x:\text{Person})$  to filter all nodes  $x$  for the label **Person**.

► **Definition 11** (Syntax of SGCQ). *A SGCQ  $\mathcal{B} \Leftarrow \Gamma$  consists of a set  $\Gamma$  of atomic components  $\gamma$  (pattern) and a set  $\mathcal{B}$  of atomic components  $\beta$  (template), defined by the following grammar:*

$$\gamma ::= (x_n, \xi) \mid (x_n, x_n):(x_e, \xi) \qquad \beta ::= (x_n, S) \mid (x_n, x_n):(x_e, S)$$

Here,  $x_n$  is a node variable, and  $x_e$  is an edge variable.  $\xi$  is a set of expressions of the form  $:l$ ,  $.k$ , or  $.k = v$ , where  $l \in L$  is a label,  $k \in K$  is a property name, and  $v \in V$  is a property value.  $S$  is a set of assignments for setting ( $\oplus l$  and  $\oplus k = v$ ) or removing ( $\ominus l$  and  $\ominus k$ ) labels or property names. We also write  $S_L$  for referring to operations involving labels and  $S_P$  for referring to operations involving properties. Furthermore, we write, e.g.,  $\ominus_x l \in \mathcal{B}$  to indicate that  $(x, S) \in \mathcal{B}$  and  $\ominus l \in S$ . As an additional syntactical constraint of the template, edge variables that occur in the pattern may only occur when their associated node variables also occur in the template.

For the semantics of SGCQ, we again consider valuations  $\mu$  on node or edge variables  $x_n$  or  $x_e$ , so that  $\mu(x_n) \in N$  and  $\mu(x_e) \in E$ , and similar definitions for compatibility  $\sim$  and the *join* operation  $\bowtie$  as for ECCQ. We also define the union of two PGs  $G_1 \cup G_2$  as  $(N_1 \cup N_2, E_1 \cup E_2, \rho, \lambda, \sigma)$  where  $\forall e \in E_1 \cup E_2 : \rho(e) = \rho_1(e)$  if  $e \in E_1$  else  $\rho_2(e)$ ,  $\forall x \in N_1 \cup N_2 \cup E_1 \cup E_2, k \in K : \lambda(x) = \lambda_1(x) \cup \lambda_2(x)$ , and  $\sigma(x, k) = \sigma_1(x, k) \cup \sigma_2(x, k)$ . The semantics for evaluating G-CORE queries is given in Definition 12, with the intuition outlined in Example 15.

► **Definition 12** (Semantics of SGCQ). *The result of evaluating a SGCQ  $g = \mathcal{B} \Leftarrow \Gamma$  over a PG  $G$  is defined as the PG constructed by*

$$\llbracket q \rrbracket_G^{sgcq} = \llbracket \mathcal{B} \rrbracket_{\Omega, G}^{sgcq-t} \text{ where } \Omega = \llbracket \Gamma \rrbracket_G^{sgcq-p}.$$

That is, similarly to ECCQ queries, sets of bindings  $\Omega$  are constructed from the pattern and fill out the template.  $\llbracket \Gamma \rrbracket_G^{sgcq-p}$  and  $\llbracket \mathcal{B} \rrbracket_{\Omega, G}^{sgcq-t}$  are specified in Definition 13 and Definition 14 based on [3], respectively.

► **Definition 13** (SGCQ Pattern Semantics). *The set of bindings  $\Omega$  resulting from evaluation of a SGCQ pattern  $\Gamma$  over a graph  $G$ , written  $\llbracket \Gamma \rrbracket_G^{sgcq-p}$ , is defined by the following cases:*

$$\begin{aligned} \llbracket (x, \xi) \rrbracket_G^{sgcq-p} &= \{ \mu \mid \mu(x) \in N, \\ &\quad \forall \xi_i \in \xi : \llbracket \xi_i \rrbracket_{\mu(x), G}^{sgcq-e} = \top \} \\ \llbracket (x, y):(z, \xi) \rrbracket_G^{sgcq-p} &= \{ \mu \mid \mu(x), \mu(y) \in N, \mu(z) \in E, \\ &\quad \rho(\mu(z)) = (\mu(x), \mu(y)), \\ &\quad \forall \xi_i \in \xi : \llbracket \xi_i \rrbracket_{\mu(z), G}^{sgcq-e} = \top \} \\ \llbracket \Gamma \rrbracket_G^{sgcq-p} &= \bowtie_{\gamma \in \Gamma} \llbracket \gamma \rrbracket_G^{sgcq-p} \end{aligned} \qquad \begin{aligned} \llbracket :l \rrbracket_{u, G}^{sgcq-e} &= \top \text{ iff } l \in \lambda(u) \\ \llbracket .k \rrbracket_{u, G}^{sgcq-e} &= \top \text{ iff } \sigma(u, k) \neq \emptyset \\ \llbracket .k = v \rrbracket_{u, G}^{sgcq-e} &= \top \text{ iff } v \in \sigma(u, k) \end{aligned}$$

► **Definition 14** (SGCQ Template Semantics). *Evaluation of a SGCQ template  $\mathcal{B}$  over a graph  $G$ , with bindings  $\Omega$ , written  $\llbracket \mathcal{B} \rrbracket_{\Omega, G}^{sgcq-t}$ , is defined by the following cases:*

$$\begin{aligned} \llbracket (x, S) \rrbracket_{\mu, G}^{sgcq-t} &= \{\{v\}, \emptyset, \emptyset, \llbracket S_L \rrbracket_{v, G}^{sgcq-s}, \llbracket S_P \rrbracket_{v, G}^{sgcq-s}\} \\ \llbracket (x, y):(z, S) \rrbracket_{\mu, G}^{sgcq-t} &= \{\{v, u\}, \{e\}, \{e \rightarrow (v, u)\}, \llbracket S_L \rrbracket_{e, G}^{sgcq-s}, \llbracket S_P \rrbracket_{e, G}^{sgcq-s}\} \\ \llbracket \mathcal{B} \rrbracket_{\Omega, G}^{sgcq-t} &= \bigcup_{\beta \in \mathcal{B}, \mu \in \Omega} \llbracket \beta \rrbracket_{\mu, G}^{sgcq-t} \end{aligned}$$

where  $v = \mu(x)$ ,  $u = \mu(y)$ , and  $e = \mu(z)$  if the variables are in the domain of  $\mu$ , or fresh identities otherwise. Evaluation of set and remove clauses is defined as

$$\begin{aligned} \llbracket S_L \rrbracket_{o_N, G}^{sgcq-s} &= (\lambda_{o_N} \cup \{o_N \rightarrow l_N \mid \oplus l_N \in S_L\}) \setminus \{o_N \rightarrow l_N \mid \ominus l_N \in S_L\} \\ \llbracket S_P \rrbracket_{o_N, G}^{sgcq-s} &= (\sigma' \cup \sigma_{\oplus}) \setminus \sigma_{\ominus} \end{aligned}$$

where

$$\begin{aligned} \sigma' &= \{(o_N, k_N) \rightarrow v \mid (o_N, k_N) \rightarrow v \in \sigma \wedge (o_N, k_N) \rightarrow v' \notin \sigma_{\oplus}\} \\ \sigma_{\oplus} &= \{(o_N, k_N) \rightarrow v \mid \oplus k_N = v \in S_P\} \\ \sigma_{\ominus} &= \{(o_N, k_N) \rightarrow v \mid (o_N, k_N) \rightarrow v \in \sigma \wedge \ominus k_N \in S_P\} \end{aligned}$$

► **Example 15.** Recall the ECCQ  $q_1^{eccq}$  from Example 5. A *seemingly* similar SGCQ could be defined as

$$q_0^{sgcq} = \{(\mathbf{x}, \{\oplus \text{Agent}\}), (\mathbf{y}, \{\oplus \text{POI}\})\} \Leftarrow \{(\mathbf{x}, \mathbf{y}):(\mathbf{e}, \{\text{:obs}\})\}$$

Evaluating  $\llbracket q_0^{sgcq} \rrbracket_{G_1^{pg}}^{sgcq}$  (cf. Example 10) produces first the single mapping  $\mu$  with  $\mu(\mathbf{x}) = \mathbf{1}$ ,  $\mu(\mathbf{y}) = \mathbf{2}$ , and  $\mu(\mathbf{e}) = \mathbf{10}$ ; then, the template constructs the graph  $G_0^{pg} = (\{\mathbf{1}, \mathbf{2}\}, \emptyset, \emptyset, \lambda_0, \sigma_0)$  with  $\lambda_0(\mathbf{1}) = \{\text{Agent}\}$ ,  $\lambda_0(\mathbf{2}) = \{\text{Person}, \text{POI}\}$ , and  $\sigma_0(\mathbf{1}, \text{name}) = \{\text{"Smith"}\}$ . That is, all labels and properties persist in the output graph. The following update to  $q_0^{sgcq}$  has equivalent semantics to the ECCQ query:

$$q_1^{sgcq} = \{(\mathbf{x}, \{\ominus \text{name}\}), (\mathbf{y}, \{\oplus \text{POI}, \ominus \text{Person}\})\} \Leftarrow \{(\mathbf{x}, \mathbf{y}):(\mathbf{e}, \{\text{:obs}\})\}$$

Evaluation produces the graph  $(\{\mathbf{1}, \mathbf{2}\}, \emptyset, \emptyset, \lambda_1, \emptyset)$  with  $\lambda_1(\mathbf{1}) = \{\text{Agent}\}$  and  $\lambda_1(\mathbf{2}) = \{\text{POI}\}$ , as we would expect (cf. Example 5).

1 :Agent

2 :POI

## 2.2.2 Simple Property Graph Shapes (sProGS)

We define the syntax and semantics of a subset of the *Property Graphs Shapes Language* [44] (ProGS) we call *Simple Property Graph Shapes Language* (sProGS). The shape language is heavily inspired by SHACL and adapted for property graphs by differentiating between node and edge shapes, introducing constraints for property annotations, and for constraining the nodes (or edges, respectively) reachable from edges (or nodes, respectively).

We omit named shapes, simplifying shape semantics by avoiding recursive shapes (non-recursive named shapes are merely syntactic sugar), as well as some other features such as qualified number restrictions or complex path expressions. A sProGS shape is either a node shape  $_N \langle q_N, \phi_N \rangle$  or an



$$\begin{aligned}
\llbracket n \rrbracket_G^{pro-q} &= \{n\} & \llbracket e \rrbracket_G^{pro-q} &= \{e\} \\
\llbracket l_N \rrbracket_G^{pro-q} &= \{n \mid n \in N \wedge l_N \in \lambda(n)\} & \llbracket l_E \rrbracket_G^{pro-q} &= \{e \mid e \in E \wedge l_E \in \lambda(e)\} \\
\llbracket k_N \rrbracket_G^{pro-q} &= \{n \mid n \in N \wedge \sigma(n, k_N) \neq \emptyset\} & \llbracket k_E \rrbracket_G^{pro-q} &= \{e \mid e \in E \wedge \sigma(e, k_E) \neq \emptyset\}
\end{aligned}$$

■ **Figure 4** Evaluation of target node and target edge queries.

edge shape  $E\langle q_E, \phi_E \rangle$ , where  $q_N$  and  $q_E$  are the *target queries*, and  $\phi_N$  and  $\phi_E$  are the respective *constraints* (see Definition 16 and Definition 17). Each node (or edge, respectively) of the graph that matches the target must satisfy the constraint.

► **Definition 16** (Node and Edge Targets). *Given target node or edge IDs  $n$  or  $e$ , node or edge labels  $l_N$  or  $l_E$ , and node or edge properties  $k_N$  or  $k_E$ , node or edge targets are defined as*

$$\begin{aligned}
q_N &::= n \mid l_N \mid k_N \\
q_E &::= e \mid l_E \mid k_E
\end{aligned}$$

► **Definition 17** (Node and Edge Constraints). *Node and edge constraints are defined by*

$$\begin{aligned}
\phi_N &::= \top \mid n \mid l_N \mid \neg \phi_N \mid \phi_N \wedge \phi_N \mid \exists k_N.(=v) \mid \exists k_N.\top \mid \exists \rightarrow \phi_E \mid \exists \leftarrow \phi_E \\
\phi_E &::= \top \mid e \mid l_E \mid \neg \phi_E \mid \phi_E \wedge \phi_E \mid \exists k_E.(=v) \mid \exists k_E.\top \mid \Rightarrow \phi_N \mid \Leftarrow \phi_N
\end{aligned}$$

where in particular  $\exists k_N.(=v)$  and  $\exists k_N.\top$  require properties with certain ( $v$ ) or any ( $\top$ ) values,  $\exists \rightarrow \phi_E$  requires an outgoing edge conforming to  $\phi_E$ , and  $\Rightarrow \phi_N$  requires the target node to conform to  $\phi_N$ . The remaining cases are similar.

Definition 18 introduces the validation semantics for a PG under a set of sProGS shapes. Figure 4 defines the semantics of target node and target edge queries. Figure 5 defines the evaluation semantics of node and edge constraints. These definitions differ from [44], using direct evaluation of constraints on target nodes instead of a semantics of faithful assignments, which is not needed since we omit recursion.

► **Definition 18** (sProGS Validation). *A PG  $G$  is valid regarding a set of sProGS shapes  $S$  if it is valid regarding all shapes  $s \in S$ ; validity regarding a single shape  $s = N\langle q_N, \phi_N \rangle$  (or  $s = E\langle q_E, \phi_E \rangle$ , respectively), denoted  $\text{valid}(G, s)$ , is defined as follows:*

$$\begin{aligned}
\text{valid}(G, N\langle q_N, \phi_N \rangle) &= \forall n \in \llbracket q_N \rrbracket_G^{pro-q} : \llbracket \phi_N \rrbracket_{n,G}^{pro-n} = \text{true} \\
\text{valid}(G, E\langle q_E, \phi_E \rangle) &= \forall e \in \llbracket q_E \rrbracket_G^{pro-q} : \llbracket \phi_E \rrbracket_{e,G}^{pro-e} = \text{true}
\end{aligned}$$

In a slight abuse of notation, we also write  $\text{valid}(G, S)$  to indicate that  $G$  is valid regarding each  $s \in S$ . Example 19 shows two example shapes.

► **Example 19.** Recall  $s_1^{shacl}$  from Example 8. We can express an equivalent shape using sProGS with  $s_1^{pro} = N\langle \text{Agent}, \exists \rightarrow (\text{obs} \wedge \Rightarrow \text{Person}) \rangle$ . This is a node shape that targets all nodes with the node label **Agent**; it requires for validity that these nodes have an outgoing edge ( $\exists \rightarrow \phi_E$ ) where the constraint  $\phi_E$  – an edge constraint – must validate at least one edge. For this edge, we require the edge label **obs**, and then, expressed with  $\Rightarrow \phi_N$ , that the corresponding destination node conforms to  $\phi_N$ , i.e., has the node label **Person**.

We can also express shapes that target edges, such as  $s_2^{pro} = E\langle \text{obs}, \Leftarrow \text{Agent} \wedge \Rightarrow \text{Person} \rangle$ , which states that edges labeled with **obs** must have an origin node labeled **Agent** ( $\Leftarrow \text{Agent}$ ) and the destination node labeled **Person** ( $\Rightarrow \text{Person}$ ). The example graph  $G_1^{pg}$  introduced in Example 10 is valid regarding both  $s_1^{pro}$  and  $s_2^{pro}$ .

$$\begin{aligned}
\llbracket \top \rrbracket_{n,G}^{pro-n} &= true \\
\llbracket n' \rrbracket_{n,G}^{pro-n} &= true \text{ if } n' = n \\
\llbracket l_N \rrbracket_{n,G}^{pro-n} &= true \text{ if } l_N \in \lambda(n) \\
\llbracket \neg \phi_N \rrbracket_{n,G}^{pro-n} &= true \text{ if } \text{not } \llbracket \phi_N \rrbracket_{n,G}^{pro-n} \\
\llbracket \phi_N^1 \wedge \phi_N^2 \rrbracket_{n,G}^{pro-n} &= true \text{ if } \llbracket \phi_N^1 \rrbracket_{n,G}^{pro-n} \text{ and } \llbracket \phi_N^2 \rrbracket_{n,G}^{pro-n} \\
\llbracket \exists k. (= v) \rrbracket_{n,G}^{pro-n} &= true \text{ if } v \in \sigma(n, k) \\
\llbracket \exists k. \top \rrbracket_{n,G}^{pro-n} &= true \text{ if } \sigma(n, k) \neq \emptyset \\
\llbracket \exists \rightarrow \phi_E \rrbracket_{n,G}^{pro-n} &= true \text{ if } \exists e \in E, n' \in N \text{ such that } \rho(e) = (n, n') \text{ and } \llbracket \phi_E \rrbracket_{e,G}^{pro-e} \\
\llbracket \exists \leftarrow \phi_E \rrbracket_{n,G}^{pro-n} &= true \text{ if } \exists e \in E, n' \in N \text{ such that } \rho(e) = (n', n) \text{ and } \llbracket \phi_E \rrbracket_{e,G}^{pro-e} \\
\llbracket \Rightarrow \phi_N \rrbracket_{e,G}^{pro-e} &= \llbracket \phi_N \rrbracket_{n_2,G}^{pro-n} \text{ where } (n_1, n_2) = \rho(e) \\
\llbracket \Leftarrow \phi_N \rrbracket_{e,G}^{pro-e} &= \llbracket \phi_N \rrbracket_{n_1,G}^{pro-n} \text{ where } (n_1, n_2) = \rho(e)
\end{aligned}$$

■ **Figure 5** Evaluation of node and edge constraints over a property graph  $G$ , omitting some cases for edges that are analogous to the node variants.

### 3 Formalizing the Problem

The essential problem statement of this work is formalized in the signature of Algorithm 1: We aim to decide, given a sProGS shape  $s$ , whether all possible output graphs of a SGCQ query  $q$  are valid regarding this shape. The set of *possible output graphs* is defined as the set of graphs produced by  $q$  from any graph that is valid regarding a given set of sProGS shapes  $\mathcal{S}_{in}$ , which we call the *input shapes* of our problem. With this decision procedure, we can find the set of all shapes characterizing the possible output graphs by enumerating a finite set of candidates and testing each candidate.

We refer to our prior work [41] for a detailed discussion on when such a finite set exists or how to deal with cases where it does not exist. In short, for restricted classes of shapes, the set of relevant candidates that need to be considered can be shown to be finite (e.g., *ALCHOI*-based shapes); for other cases, heuristics can be employed. Note how this allows our approach to work for compositions of queries: the output of a prior step becomes the input of the next<sup>5</sup>.

■ **Algorithm 1** High-level pseudocode outlining our core approach.

---

**Input** A finite set of shapes sProGS  $\mathcal{S}_{in}$ , a SGCQ  $q = \mathcal{B} \Leftarrow \Gamma$ , and a sProGS shape  $s$ .

**Output** Does  $\text{valid}(\llbracket q \rrbracket_{G_{in}}, s)$  hold for every graph  $G_{in}$  where  $\text{valid}(G_{in}, \mathcal{S}_{in})$ ?

- 1:  $s' \leftarrow \text{pro} \rightarrow \text{shacl}(s)$
  - 2:  $\mathcal{S}'_{in} \leftarrow \{ \text{pro} \rightarrow \text{shacl}(s_i) \mid s_i \in \mathcal{S}_{in} \}$
  - 3:  $q' \leftarrow \text{sgcq} \rightarrow \text{eccq}(q)$
  - 4:  $\Sigma \leftarrow \text{infer}(\mathcal{S}'_{in}, q')$
  - 5: **return** **if**  $\text{test}(\Sigma, s')$  **then** YES **else** UNKNOWN
- 

<sup>5</sup> Indeed, since, as we will later explore in detail, the input shapes are encoded as *ALCHOI* axioms, compositions of multiple queries do *not* require full exploration of the explicit set of output shapes but can rely on the inferred axioms directly.

As motivated in the introduction (Figure 3), we solve this problem by first mapping all components (that is, graphs, shapes, and queries) to an intermediate RDF layer and then utilizing an extended version of our algorithm presented in [41]. In Algorithm 1, we outline this approach in pseudocode: here, the functions  $_{\text{pro} \rightarrow \text{shacl}}(s)$  and  $_{\text{sgcq} \rightarrow \text{eccq}}(q)$  are the mapping functions for encoding sProGS shapes and SGCQ queries in SHACL and ECCQ, respectively. A final function for mapping the graphs themselves is not actually utilized – since our approach does not depend on concrete graph instances – but is required for the sake of formalizing the remaining mappings.

The functions *infer* and *test* refer to the extension of [41]: the function *infer* takes the RDF-mapped query and shapes, producing a set of DL axioms  $\Sigma$ ; this step extends the algorithm from [41], accounting for the additional features included in ECCQ (cf. Definition 3). The function *test* takes these axioms and checks whether it can be concluded that  $s'$  is guaranteed to hold for all possible output graphs; this step utilizes entailment.

Based on this formal problem signature, we define Theorem 20. We will construct the missing components of Algorithm 1 in the following two sections (the family of mappings in Section 4 and the extended inference approach in Section 5). Finally, in Section 6, we will show that the theorem holds.

► **Theorem 20** (Soundness of Algorithm 1). *Given a finite set of shapes  $\mathcal{S}_{\text{in}}$ , a SGCQ  $q = \mathcal{B} \Leftarrow \Gamma$ , and a shape  $s$ , then  $\text{valid}(\llbracket q \rrbracket_{G_{\text{in}}}, s)$  holds for every graph  $G_{\text{in}}$  where  $\text{valid}(G_{\text{in}}, \mathcal{S}_{\text{in}})$  if*

$$\text{test}(\text{infer}(\{_{\text{pro} \rightarrow \text{shacl}}(s_i) \mid s_i \in \mathcal{S}_{\text{in}}\}, _{\text{sgcq} \rightarrow \text{eccq}}(q)), _{\text{pro} \rightarrow \text{shacl}}(s)).$$

## 4 A Family of Mappings on Reified Property Graphs

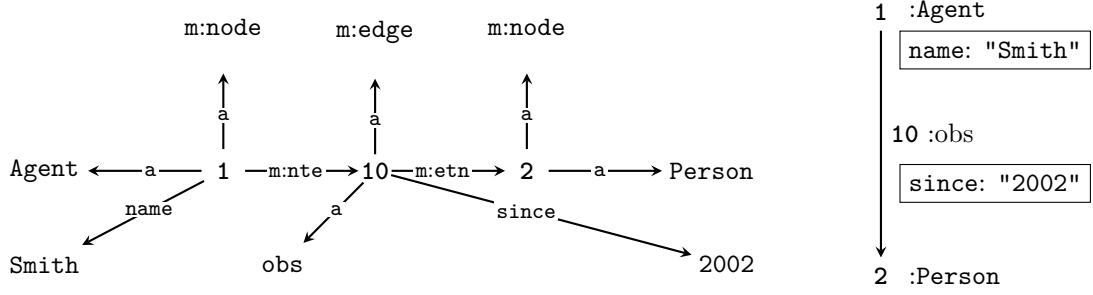
As outlined in Figure 3 and the previous section, we introduce a family of mappings: we map PGs  $G$  to RDF graphs  $_{\text{pg} \rightarrow \text{rdf}}(G)$ , utilizing reification for first-class edges (Section 4.1); this mapping is not directly utilized by our approach but instead comprises the formal basis on which the remaining mappings are defined. We map sProGS shapes  $s$  to *ALC*-encoded SHACL shapes  $_{\text{pro} \rightarrow \text{shacl}}(s)$ , accounting for the reification introduced with the graph mapping (Section 4.2). And finally, we map SGCQ  $q$  to ECCQ  $_{\text{sgcq} \rightarrow \text{eccq}}(q)$ , accounting for both reification and various semantic differences between the languages (Section 4.3).

### 4.1 PG to RDF Mapping

We first assume a utility function  $\hat{\_}$  that maps the individual elements of PGs to unique IRIs. The details of this function are left abstract; however, in real-world implementations, such a function could, for example, utilize prefixes to differentiate between PG constructs, including node and edge IDs, labels, properties, and values. We also introduce *m:nte* (*node-to-edge*) and *m:etn* (*edge-to-node*) as role names, and *m:node* and *m:edge* as special concept names distinct from *C*. The former are used for reification of first-class edges, the latter for differentiating nodes and edges explicitly<sup>6</sup>.

The straightforward mapping  $_{\text{pg} \rightarrow \text{rdf}}(G)$  is specified in Definition 21 and demonstrated in Example 22. It retains all information and can be reversed by simply inverting the presented mapping function. It utilizes  $\hat{\_}$  to map all nodes and edges, including their associated labels and properties, to RDF triples; IRIs for both PG nodes and PG edges are derived from their IDs. Edges are reified using two triples and the special role names *m:nte* and *m:etn*.

<sup>6</sup> For the sake of readability, we assign concrete names, using *m* to hint at some unique *meta* prefix that could be used in practice; they can be arbitrary but unique names.



■ **Figure 6** The reified RDF encoding  $pg \rightarrow rdf(G_1^{pg})$  (left) and the original graph  $G_1^{pg}$  (right). Note how the edge with ID 10 (right) becomes a normal RDF node via reification (left).

► **Definition 21** (Mapping from PG to RDF Graph). *The corresponding RDF graph for a PG  $G = (N, E, \rho, \lambda, \sigma)$  is the graph constructed by*

$$pg \rightarrow rdf(G) := \{ \overset{node}{pg \rightarrow rdf}(n) \mid n \in N \} \cup \{ \overset{edge}{pg \rightarrow rdf}(e) \mid e \in E \}$$

where  $pg \rightarrow rdf$  for nodes is defined as

$$\begin{aligned} \overset{node}{pg \rightarrow rdf}(n) := & \{ (\widehat{n}, \text{rdf:type}, \widehat{l_N}) \mid l_N \in \lambda(n) \} \\ & \cup \{ (\widehat{n}, \widehat{k_N}, \widehat{v}) \mid k_N \in K_N, v \in \sigma(n, k_N) \} \\ & \cup \{ (\widehat{n}, \text{rdf:type}, \text{m:node}) \} \end{aligned}$$

and for edges (with  $\rho(e) = (n_1, n_2)$ ) as

$$\begin{aligned} \overset{edge}{pg \rightarrow rdf}(e) := & \{ (\widehat{n_1}, \text{m:nte}, \widehat{e}), (\widehat{e}, \text{m:etn}, \widehat{n_2}) \} \\ & \cup \{ (\widehat{e}, \text{rdf:type}, \widehat{l_E}) \mid l_E \in \lambda(e) \} \\ & \cup \{ (\widehat{e}, \widehat{k_E}, \widehat{v}) \mid k_E \in K_E, v \in \sigma(e, k_E) \} \\ & \cup \{ (\widehat{n}, \text{rdf:type}, \text{m:edge}) \}. \end{aligned}$$

► **Example 22.** We map the graph  $G_1^{pg}$  from Example 10 to RDF, assuming a reasonable definition for  $\widehat{\phantom{x}}$  and omitting the display of any prefixes, to obtain the following set of triples for  $pg \rightarrow rdf(G_1^{pg})$ , as visualized side-by-side in Figure 6:

$$\begin{aligned} & \{(1, a, \text{m:node}), (1, a, \text{Agent}), (1, \text{name}, \text{Smith}), (2, a, \text{m:node}), (2, a, \text{Person}), \\ & (10, a, \text{m:edge}), (10, a, \text{obs}), (10, \text{since}, 2002), (1, \text{m:nte}, 10), (10, \text{m:etn}, 2)\}. \end{aligned}$$

## 4.2 sProGS to SHACL ( $\mathcal{ALCHOI}$ ) Mapping

We define a function  $pro \rightarrow shacl$  for mapping sProGS shapes to SHACL shapes. Since we follow our prior work [41] in representing SHACL shapes as  $\mathcal{ALCHOI}$  axioms, we map to  $\mathcal{ALCHOI}$  directly. That is, we map both target queries and constraints to concept descriptions and shapes to subsumption axioms (Definition 23).

► **Definition 23** (Mapping sProGS to  $\mathcal{ALCHOI}$ ). *A node shape  $N\langle q_N, \phi_N \rangle$  is mapped to the axiom  $pro \rightarrow shacl(N\langle q_N, \phi_N \rangle) = \overset{ntarget}{pro \rightarrow shacl}(q_N) \sqsubseteq \overset{nconstr}{pro \rightarrow shacl}(\phi_N)$ , where  $\overset{ntarget}{pro \rightarrow shacl}(q_N)$  and  $\overset{nconstr}{pro \rightarrow shacl}(\phi_N)$  are defined in Figure 7 and Figure 9, respectively. With equivalent definitions (Figure 8 and Figure 9), we can map edge shapes as  $pro \rightarrow shacl(E\langle q_E, \phi_E \rangle) = \overset{etarget}{pro \rightarrow shacl}(q_E) \sqsubseteq \overset{econstr}{pro \rightarrow shacl}(\phi_E)$ .*

| sProGS                                                     | $\mathcal{ALCHOI}$             |
|------------------------------------------------------------|--------------------------------|
| $\text{ntarget} \text{ pro} \rightarrow \text{shacl}(n)$   | $\{\widehat{n}\}$              |
| $\text{ntarget} \text{ pro} \rightarrow \text{shacl}(l_N)$ | $\widehat{l}_N$                |
| $\text{ntarget} \text{ pro} \rightarrow \text{shacl}(k_N)$ | $\exists \widehat{k}_N . \top$ |

■ **Figure 7** Mapping for node target queries.

| sProGS                                                     | $\mathcal{ALCHOI}$             |
|------------------------------------------------------------|--------------------------------|
| $\text{etarget} \text{ pro} \rightarrow \text{shacl}(e)$   | $\{\widehat{e}\}$              |
| $\text{etarget} \text{ pro} \rightarrow \text{shacl}(l_E)$ | $\widehat{l}_E$                |
| $\text{etarget} \text{ pro} \rightarrow \text{shacl}(k_E)$ | $\exists \widehat{k}_E . \top$ |

■ **Figure 8** Mapping for edge target queries.

| sProGS                                                                            | $\mathcal{ALCHOI}$                                                                                                                   |
|-----------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| $\text{nconstr} \text{ pro} \rightarrow \text{shacl}(\top)$                       | $\top$                                                                                                                               |
| $\text{nconstr} \text{ pro} \rightarrow \text{shacl}(n)$                          | $\widehat{n}$                                                                                                                        |
| $\text{nconstr} \text{ pro} \rightarrow \text{shacl}(l_N)$                        | $\widehat{l}_N$                                                                                                                      |
| $\text{nconstr} \text{ pro} \rightarrow \text{shacl}(\neg \phi_N)$                | $\neg \text{nconstr} \text{ pro} \rightarrow \text{shacl}(\phi_N)$                                                                   |
| $\text{nconstr} \text{ pro} \rightarrow \text{shacl}(\phi_N^1 \wedge \phi_N^2)$   | $\text{nconstr} \text{ pro} \rightarrow \text{shacl}(\phi_N^1) \sqcap \text{nconstr} \text{ pro} \rightarrow \text{shacl}(\phi_N^2)$ |
| $\text{nconstr} \text{ pro} \rightarrow \text{shacl}(\exists k_N.(= v))$          | $\exists \widehat{k}_N . v$                                                                                                          |
| $\text{nconstr} \text{ pro} \rightarrow \text{shacl}(\exists k_N.\top)$           | $\exists \widehat{k}_N . \top$                                                                                                       |
| $\text{nconstr} \text{ pro} \rightarrow \text{shacl}(\exists \leftarrow \phi_E)$  | $\exists \text{m:etn}^- . \text{econstr} \text{ pro} \rightarrow \text{shacl}(\phi_E)$                                               |
| $\text{nconstr} \text{ pro} \rightarrow \text{shacl}(\exists \rightarrow \phi_E)$ | $\exists \text{m:nte}^- . \text{econstr} \text{ pro} \rightarrow \text{shacl}(\phi_E)$                                               |
| ...                                                                               | ...                                                                                                                                  |
| $\text{econstr} \text{ pro} \rightarrow \text{shacl}(\Rightarrow \phi_N)$         | $\exists \text{m:etn}^- . \text{nconstr} \text{ pro} \rightarrow \text{shacl}(\phi_N)$                                               |
| $\text{econstr} \text{ pro} \rightarrow \text{shacl}(\Leftarrow \phi_N)$          | $\exists \text{m:nte}^- . \text{nconstr} \text{ pro} \rightarrow \text{shacl}(\phi_N)$                                               |

■ **Figure 9** Mapping sProGS node and edge constraints to  $\mathcal{ALCHOI}$ . We omit some cases for edge constraints that are equivalent to node constraint cases.

We write  $\text{pro} \rightarrow \text{shacl}(S)$  as a shorthand for mapping each  $s \in S$ . Most constructs from the syntax of sProGS translate directly to  $\mathcal{ALCHOI}$ , similarly to how SHACL is defined in terms of  $\mathcal{ALCHOI}$  in [41]. Interesting cases include the constraint  $\Rightarrow \phi_N$ , which maps to  $\exists \text{m:etn}^- . \text{nconstr} \text{ pro} \rightarrow \text{shacl}(\phi_N)$ . This node constraint requires the special role name **m:etn** according to the reification defined for PGs, since the original sProGS constraint refers to the target node of an edge. Given that this edge is an ordinary node in the RDF encoding, we must invoke  $\exists \text{m:etn}$  to navigate to the node representing that target and then use the mapped constraint  $\phi_N$ . Similarly,  $\exists \rightarrow \phi_E$  is mapped to  $\exists \text{m:nte}^- . \text{econstr} \text{ pro} \rightarrow \text{shacl}(\phi_E)$ . Example 24 shows the full mappings for the running example.

► **Example 24.** We map the shapes  $s_1^{\text{pro}}$  and  $s_2^{\text{pro}}$  from Example 19 to equivalent  $\mathcal{ALCHOI}$  subsumption axioms. For the node shape  $s_1^{\text{pro}}$ , this results in the following shape:

$$\begin{aligned} \text{pro} \rightarrow \text{shacl}(s_1^{\text{pro}}) &= \text{pro} \rightarrow \text{shacl}(N(\text{Agent}, \exists \rightarrow (\text{obs} \wedge \Rightarrow \text{Person}))) \\ &= \text{Agent} \sqsubseteq \exists \text{m:nte}^-(\text{obs} \sqcap \exists \text{m:etn}^-\text{Person}) \end{aligned}$$

Note how the reification is expressed as two nested existential quantifications: we first navigate to the reified edge (*node*) via **m:nte**, where not only a label (**obs**) but also a particular node constraint holds, to which we navigate via **m:etn**; we ensure  $1:\text{Agent} \rightarrow_{\text{m:nte}} 10:\text{obs} \rightarrow_{\text{m:etn}} 2:\text{Person}$  (cf. Figure 6). Similarly, for the edge shape  $s_2^{\text{pro}}$  we want  $1:\text{Agent} \leftarrow_{\text{m:nte}} 10:\text{obs} \rightarrow_{\text{m:etn}} 2:\text{Person}$ , but from the perspective of all reified edges, instead of nodes, such that in

$$\begin{aligned} \text{pro} \rightarrow \text{shacl}(s_2^{\text{pro}}) &= \text{pro} \rightarrow \text{shacl}(E(\text{obs}, \Leftarrow \text{Agent} \wedge \Rightarrow \text{Person})) \\ &= \text{obs} \sqsubseteq \exists \text{m:nte}^-\text{Agent} \sqcap \exists \text{m:etn}^-\text{Person} \end{aligned}$$

we target an edge to navigate back to the origin (inverse **m:nte**) as well as the destination via **m:etn**, requiring the node labels **Agent** and **Person**, respectively.

### 4.3 SGCQ to ECCQ Mapping

Finally, we map SGCQ to ECCQ. The principal challenge for this mapping – aside from accounting for reification – is that in SGCQ the semantics for constructing labels and properties are different from ECCQ in that all properties and labels are kept for constructed nodes or edges unless there are remove clauses. In ECCQ, on the other hand, all triples in the result graph must be explicitly constructed by atomic patterns in the query template.

### 3:16 Transforming Shape Schemas with Composable Property-Graph Queries

We solve this issue by utilizing generic concept and role variables with appropriate filter conditions to define a mapping  $sgcq \rightarrow eccq(q)$  for a SGCQ  $q$  (Definition 25). To this end, we introduce an intermediate representation referred to as IQL (*Intermediate Query Language*) in Definition 26 that aligns with the two types of patterns of SGCQ, i.e. matching nodes or edges, on the SGCQ side with full query patterns or templates on the ECCQ side. This aligns parts of queries on both sides that compose naturally. Note how the decision to syntactically include where clauses in node or edge patterns of SGCQ aids this composability.

Via this intermediate layer, we associate SGCQ variables with generic copy patterns expressed in ECCQ that explicitly construct the required labels and properties. *Set* clauses translate to explicit construction in ECCQ, while *remove* clauses translate to filter conditions on generic variables in the pattern of ECCQ queries, removing the offending labels from the bindings for these generic variables. We demonstrate this mapping in Example 27.

► **Definition 25** (Mapping SGCQ to ECCQ). *Given a SGCQ  $q$ , we define the mapping function  $sgcq \rightarrow eccq(q)$  by introducing the intermediate query language IQL (Definition 26) and several utility functions for mapping to and from IQL, as*

$$sgcq \rightarrow eccq(q) := iql \rightarrow eccq(sgcq \rightarrow iql(q))$$

where the following utility functions (defined in Figure 10) are utilized:

$$\begin{aligned} sgcq \rightarrow iql(\mathcal{B} \leftarrow \Gamma) &= \bigcup_{\beta \in \mathcal{B}} \text{template}_{sgcq \rightarrow iql}(\beta, \text{vars}(\Gamma)) \cup \bigcup_{\gamma \in \Gamma} \text{pattern}_{sgcq \rightarrow iql}(\gamma) \\ iql \rightarrow eccq(I_T \leftarrow I_P) &= \bigcup_{t \in I_T} \text{template}_{iql \rightarrow eccq}(t) \cup \bigcup_{p \in I_P} \text{pattern}_{iql \rightarrow eccq}(p) \end{aligned}$$

► **Definition 26** (Intermediate Query Language). *We define the syntax of the intermediate query language (IQL) as follows:*

$$\begin{aligned} q &:= I_T \leftarrow I_P \\ p &:= M_N(x_n, W_L, W_K, W_V, R_L, R_K) \mid M_E(x_n, x_n, x_e, W_L, W_K, W_V, R_L, R_K) \\ t &:= C_N(x_n, A_L, A_K, V) \mid C_E(x_n, x_n, x_e, A_L, A_K, V) \end{aligned}$$

where  $W_L$  is a set of label patterns  $:l$ ,  $W_K$  is a set of key patterns  $.k$ , and  $W_V$  is a set of patterns  $.k = v$ . Similarly,  $A_L$  is a set of set label patterns  $\oplus l$ ,  $A_K$  is a set of set key-value patterns  $\oplus k = v$ , and  $V$  is a set of variables. Finally,  $R_L$  is a set of remove label patterns  $\ominus l$ , and  $R_K$  is a set of remove key patterns  $\ominus k$ . The semantics of IQL are implicitly defined through the mappings in Figure 10.

► **Example 27.** The example SGCQ  $q_2^{sgcq}$  in Figure 11 (left) includes the major SGCQ features, such as matching on an edge pattern, setting and removing labels, and restricting results with constraints. The resulting ECCQ  $sgcq \rightarrow eccq(q_2^{sgcq})$  is shown in Figure 11 (right). The query is suitable for application to  $G_1^{pg}$  from Example 10. We focus here on the single variable  $y$ , which occurs in two atomic patterns:  $(y, :Person)$  (in  $\Gamma$ ) and  $(y, \{\oplus P0I, \ominus Person\})$  (in  $\mathcal{B}$ ).

$M_N(y, \{ :Person \}, \emptyset, \emptyset, \{ \ominus Person \}, \emptyset)$  and  $C_N(y, \{ \oplus P0I \}, \emptyset, \{ x, y, e \})$  are the corresponding IQL terms. The former maps to the ECCQ pattern including the generic pattern  $y : \bar{y}^C$ , copying all labels for  $y$ , the explicit pattern  $y : Person$  selecting instances of **Person**, but also the filter  $\bar{y}^C \neq Person$ , which prevents bindings, and thus also the construction, for label **Person** for the generic variable  $\bar{y}^C$ , included because of the remove term in the original SGCQ query. The ECCQ template includes, along with the generic patterns, the explicit construction of  $y : P0I$ .

$$\begin{aligned}
& \text{pattern}_{sgcq \rightarrow iql}((x, \xi)) := M_N(x, \{ :l \mid :l \in \xi \}, \{ .k \mid .k \in \xi \}, \\
& \quad \{ .k = v \mid .k = v \in \xi \}, \{ \ominus l \mid \ominus l \in S_x \}, \{ \ominus k \mid \ominus k \in S_x \}) \\
& \text{pattern}_{sgcq \rightarrow iql}(((x, y):z, \xi)) := M_E(x, y, z, \{ :l \mid :l \in \xi \}, \{ .k \mid .k \in \xi \}, \\
& \quad \{ .k = v \mid .k = v \in \xi \}, \{ \ominus l \mid \ominus l \in S_z \}, \{ \ominus k \mid \ominus k \in S_z \}) \\
\\
& \text{template}_{sgcq \rightarrow iql}((x, S), \mathcal{V}) := C_N(x, \{ \oplus l \mid \oplus l \in S \}, \{ \oplus k = v \mid \oplus k = v \in S \}, V) \\
& \text{template}_{sgcq \rightarrow iql}(((x, y):z, S), V) := C_E(x, y, z, \{ \oplus l \mid \oplus l \in S \}, \{ \oplus k = v \mid \oplus k = v \in S \}, V) \\
\\
& \text{pattern}_{iql \rightarrow eccq}(M_N(\dots)) := (\{x : \bar{x}^C, (x, \bar{x}^I) : \bar{x}^R, x : \mathbf{m}:\mathbf{node}\} \cup \{x : \hat{l} \mid :l \in W_L\} \\
& \quad \cup \{(x, o_{x,k}) : \hat{k} \mid .k \in W_K\} \cup \{(x, \hat{v}) : \hat{k} \mid .k = v \in W_V\}, \\
& \quad \{ \bar{x}^R \neq \mathbf{m}:\mathbf{n}te \} \cup \{ \bar{x}^C \neq \hat{l} \mid \ominus l \in R_L \} \cup \{ \bar{x}^R \neq \hat{k} \mid \ominus k \in R_K \}) \\
& \text{pattern}_{iql \rightarrow eccq}(M_E(\dots)) := (\{(x, z) : \mathbf{m}:\mathbf{n}te, (z, y) : \mathbf{m}:\mathbf{etn}, z : \bar{z}^C, (z, \bar{z}^I) : \bar{z}^R, z : \mathbf{m}:\mathbf{edge}\} \\
& \quad \cup \{z : \hat{l} \mid :l \in W_L\} \cup \{(z, o_{z,k}) : \hat{k} \mid .k \in W_K\} \\
& \quad \cup \{(z, \hat{v}) : \hat{k} \mid .k = v \in W_V\}, \\
& \quad \{ \bar{z}^R \neq \mathbf{m}:\mathbf{etn} \} \cup \{ \bar{z}^C \neq \hat{l} \mid \ominus l \in R_L \} \cup \{ \bar{z}^R \neq \hat{k} \mid \ominus k \in R_K \}) \\
\\
& \text{template}_{iql \rightarrow eccq}(C_N(\dots)) := \{x : \mathbf{m}:\mathbf{node}\} \cup \{(x : \bar{x}^C, (x, \bar{x}^I) : \bar{x}^R \text{ if } x \in V \text{ else } \{\}) \\
& \quad \cup \{x : \hat{l} \mid \oplus l \in A_L\} \cup \{(x, \hat{v}) : \hat{k} \mid \oplus k = v \in A_K\} \\
& \text{template}_{iql \rightarrow eccq}(C_E(\dots)) := \{(x, z) : \mathbf{m}:\mathbf{n}te, (z, y) : \mathbf{m}:\mathbf{etn}, z : \mathbf{m}:\mathbf{edge}\} \\
& \quad \cup (\{z : \bar{z}^C, (z, \bar{z}^I) : \bar{z}^R \text{ if } z \in V \text{ else } \{\}) \\
& \quad \cup \{x : \hat{l} \mid \oplus l \in A_L\} \cup \{(x, \hat{v}) : \hat{k} \mid \oplus k = v \in A_K\}
\end{aligned}$$

■ **Figure 10** Component-wise mappings from SGCQ to IQL and IQL to ECCQ, both for query templates and patterns. We write  $S_x$  to refer to the union of all sets of add or remove patterns  $S$ , such that  $(x, S) \in \mathcal{B}$  or  $(x_1, x_2) : (x, S) \in \mathcal{B}$ .

## 5 Shape Inference for ECCQ

As introduced in the formalization of our core problem in Section 3, we aim to infer a set of  $\mathcal{ALCHOT}$  axioms  $\Sigma$  from an ECCQ  $q$  and a set of shapes  $\mathcal{S}_{in}$  (written  $\Sigma = \text{infer}(\mathcal{S}_{in}, q)$ ), such that  $\Sigma \models s_o$  (test( $\Sigma, s_o$ ) in Section 3) implies that  $s_o$  is a shape validating all possible output graphs of  $q$ , under the precondition that the input graphs of  $q$  are guaranteed to be valid regarding  $\mathcal{S}_{in}$ .

### 5.1 Encoding Constraints in DL

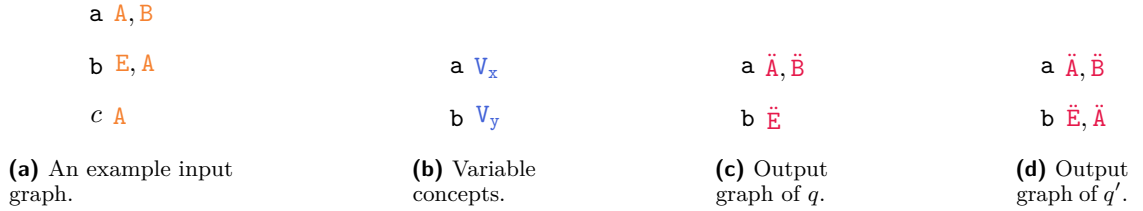
We build on and extend [41], which essentially approximates axioms of the closed-world assumption from sub-expressions of the query and joins them with the input shapes and a few additional axioms. To this end, three namespaces need to be distinguished: those of the input graphs, of the subgraphs matched by the query, and of the new output graphs constructed as a result. The extensions of concepts involved clearly differ between these three namespaces; thus, we follow [41] in renaming any concept name  $A$  in the input namespaces as  $\dot{A}$  and  $\ddot{A}$  in the matched and output namespaces. We do the same for each role name  $p$  (with  $\dot{p}$  and  $\ddot{p}$ ).



### 3:18 Transforming Shape Schemas with Composable Property-Graph Queries

|                                                                                                                                                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $\{(x, \emptyset),$ $(x, y):(e, \emptyset),$ $(y, \{\oplus \text{POI}, \ominus \text{Person}\})\}$ $\leftarrow$ $\{(x, \{ \text{name} = \text{"Smith"} \}),$ $(x, y):(e, \{ \text{:obs}\}),$ $(y, \{ \text{:Person}\})\}$ | $\{x:\text{m:node}, x:\bar{x}^C, (x, \bar{x}^I):\bar{x}^R,$ $e:\text{m:edge}, e:\bar{e}^C, (e, \bar{e}^I):\bar{e}^R, (x, e):\text{m:nte}, (e, y):\text{m:etn},$ $y:\text{m:node}, y:\bar{y}^C, (y, \bar{y}^I):\bar{y}^R, y:\text{POI}\}$ $\leftarrow$ $\{x:\text{m:node}, x:\bar{x}^C, (x, \bar{x}^I):\bar{x}^R, (x, \text{Smith}):\text{name},$ $e:\text{m:edge}, e:\bar{e}^C, (e, \bar{e}^I):\bar{e}^R, e:\text{obs}, (x, e):\text{m:nte}, (e, y):\text{m:etn},$ $y:\text{m:node}, y:\bar{y}^C, (y, \bar{y}^I):\bar{y}^R, y:\text{Person}\},$ $\{\bar{y}^C \neq \text{Person}, \bar{x}^R \neq \text{m:nte}, \bar{e}^R \neq \text{m:etn}, \bar{y}^R \neq \text{m:nte}\}$ |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

■ **Figure 11** SGCQ  $q_2^{sgcq}$  on the left, and ECCQ  $q_2^{sgcq} \rightarrow_{sgcq \rightarrow eccq} q_2^{sgcq}$  on the right. Note how we visually align the lines of the left and right columns for their common variables, hinting at the IQL rules, e.g., the POI set label and respective triple pattern (blue). The only exception is the inclusion of  $\bar{y}^C \neq \text{Person}$  (yellow) in the filter patterns of the ECCQ (right column), highlighting why the IQL pattern  $M_N$  needs the remove clauses as part of their arguments: remove clauses from the template of SGCQ affect the pattern (filters) of ECCQ.



■ **Figure 12** Example for graphs involved in constructing the validation knowledge base (cf. [41]) for the query  $q$  with  $P = H = \{x:A, x:B, y:E\}$  from Example 28. This case demonstrates the potential issue arising when adding the atomic pattern  $y:\bar{y}^C$  in  $q'$ , since variable  $y$  matches **b** which is an instance of **A**; this violates the assumptions about the relationship of  $\ddot{A}$  and  $\ddot{B}$  in the output graph. Colors visualize different namespaces, while floating labels indicate concept names (e.g.,  $a$  is both **A** and **B**).

Query variables can be represented with *variable concepts*, that is, fresh concept names  $V_x$  for each variable  $x$ . The extensions of these concepts are equal to the set of all of their bindings. Thus, variable concepts remain the same in all namespaces and are not subject to any renaming. They are constrained by the basic graph patterns of the query pattern  $P$  they occur in and therefore play a key role in encoding the input-output relationship of queries.

The essential extension of our SPARQL subset ECCQ over the subset considered in [41] are generic copy operations through variables for concepts and properties and certain filter expressions on these variables. The minimal Example 28 shows how this invalidates the axioms inferred by [41], while also giving the intuition of this method.

► **Example 28.** Assume a query  $q$  with  $P = H = \{x:A, x:B, y:E\}$  and no input shapes. The axioms inferred by [41] would include  $\{V_x \equiv A \sqcap B, V_y \equiv E, \ddot{A} \equiv V_x, \ddot{B} \equiv V_x, \ddot{E} \equiv V_y\}$ . We demonstrate these for an example input graph (Figure 12a) in Figure 12. That is, we define the variable concept  $V_x$  (visualized as a graph in Figure 12b) in terms of its bindings, namely the intersection of **A** and **B**, and the new concept names in the output namespace  $\ddot{A}$  and  $\ddot{B}$  in terms of these  $V_x$ . Since these axioms entail, e.g.,  $\ddot{A} \sqsubseteq \ddot{B}$ , just as in the example Figure 12c, we can conclude that the shape  $A \sqsubseteq B$  holds on all output graphs.

If we were to extend query  $q$  as  $q'$  by adding  $\{y:\bar{y}^C\}$  to both  $P$  and  $H$ , the axiom  $\ddot{A} \sqsubseteq \ddot{B}$  *should* no longer be entailed; indeed, variable  $y$  might now also include – such as in Figure 12d – bindings that are of type  $A$  exclusive or  $B$  and modify  $\ddot{A}$  or  $\ddot{B}$  through the generic pattern  $y:\bar{y}^C$ , thereby invalidating this subsumption.

We can remedy this broken inference introduced in Example 28 by including *components* of all variables that have generic copy operations in the definition of concepts like  $\ddot{A}$ . Such components are defined in Definition 30 and included in axioms constructed by the adapted method outlined in Definition 31. We write  $\text{vcg}(q)$  (*variable connectivity graph*) to mean a graph where variables  $\text{var}(q)$  are nodes and there exists an edge between two variables if and only if they both occur in a pattern (e.g.,  $x$  and  $y$  in  $(x,y):p$ ) in  $\text{sgcq} \rightarrow \text{eccq}(q)$ . Example 28 can therefore be fixed by using this extension as shown in Example 29. We demonstrate a full example of the algorithm in Section 5.3.

► **Example 29.** With the extended method, we infer the set of axioms including  $\{V_x \equiv A \sqcap B, V_y \equiv E, \ddot{A} \equiv V_x \sqcup V_y^A \sqcup V_x^A, \ddot{B} \equiv V_x \sqcup V_x^B \sqcup V_y^B, \ddot{E} \equiv V_y \sqcup V_x^E \sqcup V_y^E\}$  where  $V_y^A \equiv V_y \sqcap A$ ,  $V_y^B \equiv V_y \sqcap B$ ,  $V_y^E \equiv V_y \sqcap E$  and  $V_x^A \equiv V_x^B \equiv V_x^E \equiv \perp$ .

Here, the axioms defined for the output namespace (e.g.,  $\ddot{A} \equiv V_x \sqcup V_x^A \sqcup V_y^A$ ) include additional component concepts (e.g.,  $V_y^A$ ) for all variables in the query. Thus, we ensure that the possibility of atomic pattern  $y:\bar{y}^C$  constructing instances of  $\ddot{A}$  in output graphs is accounted for. As a result, we can no longer *erroneously* infer  $A \sqsubseteq B$  without additional knowledge in terms of input shapes about the relationship of  $A$ ,  $B$ , and  $E$ .

► **Definition 30 (Concept Components).** We define the concept components  $\text{Cc}(q)$  of an ECCQ  $q = H \leftarrow (P, F)$  as the set of axioms including:

1. For all  $x \in \text{var}(P)$  and  $A \in \mathcal{V}$ , if  $x:\bar{x}^C \in P \cap H$  and  $(\bar{x}^C \neq A) \notin F$ , then  $V_x^A \equiv V_x \sqcap A$ . Otherwise,  $V_x^A \equiv \perp$ .
2. For all  $x, y \in \text{var}(P)$  and  $p \in \mathcal{V}$ , if  $(x, \bar{x}^I):\bar{x}^R \in P \cap H$ ,  $(\bar{x}^R \neq p) \notin F$  and  $(x, y):p \notin P$ , then  $V_x^p \equiv V_x \sqcap \exists p.V_x^{p,o}$ . Otherwise,  $V_x^p \equiv \perp$  and  $V_x^{p,o} \equiv \perp$ .

Here,  $\mathcal{V} = \text{voc}(q) \cup \text{voc}(\mathcal{S}_{\text{in}})$  is the vocabulary of all concept, role, and individual names that occur in the query or in input shapes.

► **Definition 31 (Axiom Encoding).** The axiom encoding of an ECCQ  $q$  and a set of  $\mathcal{ALCHOI}$  shapes  $\mathcal{S}_{\text{in}}$ , denoted  $\text{infer}(\mathcal{S}_{\text{in}}, q)$ , is the minimal set of  $\mathcal{ALCHOI}$  axioms that include  $\text{Cc}(q)$  (Definition 30) as well as the following axioms:

1. The set of input shapes  $\mathcal{S}_{\text{in}}$ .
2. For each variable  $x$  in  $\text{var}(q)$ , the axiom

$$V_x \sqsubseteq \prod_{x:A \in P} A \sqcap \prod_{(x,u):p \in P} \exists p.C_u \sqcap \prod_{(u,x):p \in P} \exists p^-.C_u,$$

and if  $\text{vcg}(P \cup H)$  is acyclic regarding  $x$ , then also the axiom

$$V_x \sqsupseteq \prod_{x:A \in P} A \sqcap \prod_{(x,u):p \in P} \exists p.C_u \sqcap \prod_{(u,x):p \in P} \exists p^-.C_u.$$

3. For each concept name  $A$  in  $\text{voc}(H)$ ,  $\ddot{A} \equiv \bigsqcup_{u:A \in H} C_u \sqcup \bigsqcup_{x \in \text{var}(H)} V_x^A$ .

4. For each  $p$  in  $\text{voc}(H)$

$$\begin{aligned} \bigsqcup_{(u,v):p \in H} C_u &\sqsubseteq \exists \ddot{p}. C_v, \\ \exists \ddot{p}. C_v &\sqsubseteq \bigsqcup_{(u,v):p \in H} C_u \sqcup \bigsqcup_{(x,\bar{x}^I): \bar{x}^R \in H} V_x, \\ \exists \ddot{p}. \top &\equiv \bigsqcup_{(u,v):p \in H} (C_u \sqcap \exists \ddot{p}. C_v) \sqcup \bigsqcup_{x \in \text{var}(H)} V_x^p, \\ \bigsqcup_{(u,v):p \in H} C_v &\sqsubseteq \exists \ddot{p}^-. C_u, \\ \exists \ddot{p}^-. C_u &\sqsubseteq \bigsqcup_{(u,v):p \in H} C_v \sqcup \bigsqcup_{(x,\bar{x}^I): \bar{x}^R \in H} V_x^{p,o}, \\ \exists \ddot{p}^-. \top &\equiv \bigsqcup_{(u,v):p \in P} (C_v \sqcap \exists \ddot{p}^-. C_u). \end{aligned}$$

5. For each  $p \in P$ ,  $\dot{p} \sqsubseteq p$ .

6. For each  $p \in P$ ,  $p \sqsubseteq \dot{p}$  if all atomic patterns including  $p$  in  $P$  have the form  $(x,y):p$  where  $x, y$  occur in no other atomic pattern in  $P$  and  $x \neq y$ .

7. For each pair  $p, r$  with  $(x,y):p \in P$  and  $(x,y):r \in H$  or  $(y,x):r \in H$

- a.  $\dot{p} \sqsubseteq \ddot{r}$  (if  $(x,y):r \in H$ ) or  $\dot{p} \sqsubseteq \ddot{r}^-$  (if  $(y,x):r \in H$ ) – if  $P$  does not contain any other atomic patterns with  $p$ , and
- b.  $\ddot{r} \sqsubseteq \dot{p}$  (if  $(x,y):r \in H$ ) or  $\ddot{r}^- \sqsubseteq \dot{p}$  (if  $(y,x):r \in H$ ) – if  $H$  does not contain any other atomic patterns with  $r$ .

Cases 1, 2, and 5-7 are taken from, and 3 and 4 are adapted from [41].

## 5.2 Enriching the Query

Generic copy operations can invalidate output shapes in certain cases, as shown in Example 29. On the other hand, they enable means of semantically enriching the ECCQ query, leading to additional shapes that *can* be shown to hold. To this end, we define the extension rules in Definition 32 that essentially copy constraints from the query pattern to the template in cases where this is allowed by generic patterns. We include an example in Section 5.3.

► **Definition 32** (Extension of ECCQ). *The semantics-preserving extension of a ECCQ  $q = H \leftarrow (P, F)$ , written  $\text{Ext}(q)$ , is defined by inserting the following atomic patterns to the template  $H$ :*

- 1. *If  $x:\bar{x}^C \in P \cap H$ , then for each  $x:A \in P$  such that  $(\bar{x}^C \neq A) \notin F$ , we include the atomic pattern  $x:A$  in  $H$ , and*
- 2. *if  $(x,\bar{x}^I):\bar{x}^R \in P \cap H$ , then for each  $(x,a):p \in P$  such that  $(\bar{x}^R \neq p) \notin F$ , we include the atomic pattern  $(x,a):p$  in  $H$ .*

## 5.3 Example

We now demonstrate some axioms in  $\Sigma$  constructed for the problem inputs  $S_1^{pro} = \{s_1^{pro}, s_2^{pro}\}$  (Example 19) and  $q_2^{sgcq}$  (Example 27). Again, we will focus on axioms related to concept names, as they are more intuitive, yet in principle very similar to axioms related to role names. As a first step, we need to map the query to ECCQ as shown in Example 27 and then extend the mapped query to obtain the ECCQ input as  $q_2^{eccq} = \text{Ext}(sgcq \rightarrow eccq(q_2^{sgcq}))$ . The extended query template includes the atomic patterns  $\{e:\text{obs}, (x, \text{"Smith"}):\text{name}\}$  in addition to the atomic patterns that are shown in Figure 11, which are copied from the query pattern due to the presence of generic patterns for  $x$  and  $e$ . On the other hand, since the only constraining atomic pattern for  $y$ , namely  $y:\text{Person}$ , is also included in the filter  $\bar{y}^C \neq \text{Person}$ , there is no template extension for variable  $y$ , even though there are generic patterns (cf. Definition 32).

According to Definition 31.1 we need to include the input shapes in  $\Sigma$  after mapping them to  $\mathcal{ALCHQI}$  axioms via  $pro \rightarrow shacl(S_1^{pro})$ . These axioms encode constraints that are known to hold on the input graph by definition, since we only consider valid input graphs (cf. Example 24).

According to Definition 31.2 we next define variable concepts encoding known constraints for the bindings of all query variables. We include the following axioms in  $\Sigma$ :

$$\begin{aligned} V_e &\equiv \exists m:etn.V_y \sqcap \exists m:nte.V_x \sqcap obs \\ V_x &\equiv \exists name.Smith \sqcap \exists m:nte.V_e \\ V_y &\equiv \exists m:etn.V_e \sqcap Person \end{aligned}$$

These variable concepts are almost direct translations of the query pattern; for example, variable  $x$  occurs in  $(x, Smith):name$  and  $(x, e):m:nte$  (Example 27), which translates to the intersection of  $\exists name.Smith$  and  $\exists m:nte.V_e$  in the definition of  $V_x$ . Note how reification again plays a role here, with the presence of the role name  $m:nte$ .

Constraints related to concept names that explicitly occur in the query template are defined according to Definition 31.3 in terms of these variable concepts, including the concept components that were introduced in Definition 30. In the following set, we also include one axiom inferred from the rules for properties, namely  $V_e \sqsubseteq \exists m:etn.V_y$ .

$$\begin{aligned} \ddot{obs} &\equiv V_e \sqcup V_e^{obs} & P\ddot{OI} &\equiv V_y \sqcup V_x^{POI} \sqcup V_y^{POI} \\ \ddot{Agent} &\equiv V_x^{Agent} \sqcup V_y^{Agent} & V_x^{POI} &\equiv V_x \sqcap POI \\ V_x^{Agent} &\equiv V_x \sqcap Agent & V_y^{POI} &\equiv V_y \sqcap POI \\ V_y^{Agent} &\equiv V_y \sqcap Agent & V_e &\sqsubseteq \exists m:etn.V_y \\ V_e^{obs} &\equiv V_e \end{aligned}$$

There are a few noteworthy aspects to these axioms: firstly, note how  $P\ddot{OI}$  (defined with dots, since it differs from the concept  $POI$ , which may occur in input graphs) is defined in terms of  $V_y$ , because it exclusively occurs in the atomic pattern  $y:POI$  of the query template, and  $V_y$  encodes the bindings of variable  $y$ . Furthermore, we must include the concept components  $V_x^{POI}$  (and also  $V_y^{POI}$ , though this is redundant in this case) in the definition of  $P\ddot{OI}$  since variable  $x$  occurs with the generic pattern  $x:\bar{x}^C$ , meaning that the new concept  $P\ddot{OI}$  may extend over some bindings for  $x$ , too. We treat  $\ddot{Agent}$  similarly, with the difference that there is no actual construct pattern featuring  $\ddot{Agent}$ ; thus, its definition consists only of concept components.

The complete set of inferred axioms entails, e.g.,  $\ddot{obs} \sqsubseteq \exists m:etn.P\ddot{OI}$ . Therefore, we know that  $obs \sqsubseteq \exists m:etn.POI$  validates all output graphs of the query. For the final shapes, we drop the namespace distinction, since it is not part of our (or any regular) query semantics.

## 6 Metatheory

To encode our inference task in *ALCHOI*, we first map shapes and queries to *ALCHOI* and *ECCQ* then infer a set of axioms  $\Sigma$  from these mapped queries and shapes (cf. Algorithm 1). Therefore, we need to prove that any mapped shape  $C \sqsubseteq D$  holds on all output graphs of the query if  $\Sigma \models \ddot{C} \sqsubseteq \ddot{D}$ . That is, if our inferred set of axioms  $\Sigma$  entails the respective axiom (i.e. encoded shape) in the output graph namespace annotated with two dots.

The primary outcome of the following considerations is the soundness of Algorithm 1, as expressed in Theorem 20. To this end, we first prove the semantic equivalence for our mappings in Proposition 33 and Proposition 34. Full proofs are available in our extended version.

► **Proposition 33** (Soundness of Shape Mapping). *Given a PG  $G$  and a set of sProGS shapes  $S$ , the graph  $G$  is valid regarding each shape  $s \in S$  if and only if the mapped graph  $pg \mapsto r_{df}(G)$  is valid regarding the mapped shapes  $pro \mapsto shacl(s)$ .*

**Proof Sketch.** We prove the proposition in two steps. First, we show that given a sProGS shape  $s$ , the targets of  $s$  in  $G$  correspond to the set of targets of  $_{\text{pro} \rightarrow \text{shacl}}(s)$  in  $_{\text{pg} \rightarrow \text{rdf}}(G)$ . Then, we show that for any node (or edge) in  $G$ , it conforms to the constraint of a shape  $s$  if and only if the mapped node in  $_{\text{pg} \rightarrow \text{rdf}}(G)$  conforms to the constraint of  $_{\text{pro} \rightarrow \text{shacl}}(s)$ . ◀

► **Proposition 34** (Soundness of Query Mapping). *Given a PG  $G$  and a SGCQ  $q$ , the following holds:  $\llbracket q \rrbracket_G^{\text{sgcq}} = _{\text{pg} \rightarrow \text{rdf}}^{-1}(\llbracket \text{sgcq} \rightarrow \text{eccq}(q) \rrbracket_{_{\text{pg} \rightarrow \text{rdf}}(G)}^{\text{eccq}})$ , where  $_{\text{pg} \rightarrow \text{rdf}}^{-1}$  is the inverse mapping of  $_{\text{pg} \rightarrow \text{rdf}}$ . That is, evaluating the query  $q$  is the same as evaluating the mapped query on the mapped graph and then mapping the result graph back.*

**Proof Sketch.** We separate the *matching* and *constructing* parts of queries for this proof. To this end, we rely on the composable intermediate query language (IQL), making individual node or edge patterns compose neatly with full ECCQ patterns (or templates). We define an equivalence relation between sets of bindings  $\Omega$  for queries expressed as both SGCQ and ECCQ, which is not straightforward, given that SGCQ queries bind only nodes and edges (while labels and properties are implicit), but the corresponding reified labels and properties in ECCQ are explicitly bound as variables. We prove that both SGCQ and mapped ECCQ produce equivalent bindings. In the second part of the proof, we show that – given sets of equivalent bindings – the constructed graphs are again the same, modulo the mapping relation. ◀

It remains to show that shapes entailed by the axioms  $\Sigma$  constructed according to Definition 31 are guaranteed to validate output graphs of the query, modulo renaming in the appropriate namespace.

► **Proposition 35.** *Given an ECCQ  $q$  and a set of  $\mathcal{ALCHOI}$  shapes  $\mathcal{S}_{\text{in}}$  it holds for all RDF graphs  $G$  with  $\text{valid}(G, \mathcal{S}_{\text{in}})$  that  $\text{infer}(\mathcal{S}_{\text{in}}, q) \models \tilde{s}$  implies  $\text{valid}(\llbracket q \rrbracket_G^{\text{eccq}}, s)$ .*

**Proof Sketch.** For this proof we first formalize the different conceptual graphs involved (according to the namespaces separated with one or two dots) as joined *extended graphs* and then show for the axioms constructed by the rules of Definition 31 that all extended graphs are indeed valid regarding these axioms. From this, we then follow (utilizing some auxiliary lemmas) that any other entailed shapes, when renamed accordingly, validate all possible output graphs of the query. This proof builds on the proofs in [41]. ◀

Finally, we show that extending the ECCQ preserves query semantics.

► **Proposition 36.** *For any ECCQ  $q$  and RDF graph  $G$ ,  $\llbracket q \rrbracket_G^{\text{eccq}} = \llbracket \text{Ext}(q) \rrbracket_G^{\text{eccq}}$ .*

**Proof Sketch.** This proposition follows directly from the specified semantics of ECCQ since all concept or role names explicitly added to the template of  $q$  by rules for  $\text{Ext}(q)$  (Definition 32) are also constructed by the respective generic patterns  $x:\bar{x}^{\mathbf{C}}$  and  $(x, \bar{x}^{\mathbf{I}}):\bar{x}^{\mathbf{R}}$ , except those that occur in filter conditions, which are excluded in the definition of  $\text{Ext}(q)$ . ◀

With these propositions, we can finally address the main claim of this paper. We rewrite the theorem initially introduced as Theorem 20 in Theorem 37, replacing the function *test* with concrete entailment and including the family of mappings. The proof follows directly from the propositions discussed in this section.

► **Theorem 37** (Reformulation of Theorem 20). *Given a finite set of sProGS shapes  $\mathcal{S}_{\text{in}}$ , a SGCQ  $q = \mathcal{B} \Leftarrow \Gamma$ , and a sProGS shape  $s$ , then  $\text{valid}(\llbracket q \rrbracket_{G_{\text{in}}}, s)$  holds for every graph  $G_{\text{in}}$  where  $\text{valid}(G_{\text{in}}, \mathcal{S}_{\text{in}})$  if  $\text{infer}(_{\text{pro} \rightarrow \text{shacl}}(\mathcal{S}_{\text{in}}), \text{sgcq} \rightarrow \text{eccq}(q)) \vdash _{\text{pro} \rightarrow \text{shacl}}(\tilde{s})$ .*

**Proof Sketch.** This follows directly from the propositions introduced in this section. ◀

## 7 Implementation and Evaluation

We implemented the algorithm described in this paper, including both the mapping and inference components, in Scala. To this end, we built on our previous implementation of [41] (see [42]), which is available under a free software license. Our fork of this implementation is available on GitHub<sup>7</sup> [43] as well.

We implemented parsers, as well as our formally specified mappings, for both G-CORE to SPARQL CONSTRUCT and ProGS to SHACL; modified the inference algorithm of the implementation [42] with our extended variant as specified, constructing different sets of description logic axioms that are then passed to the Hermit [28] reasoner for inference; implemented unit and end-to-end test cases that cover G-CORE and the extended SPARQL features; and validated the algorithm based on query execution over randomly generated graphs.

### 7.1 Experimental Setup and Results

To validate the correctness of our implementation and gather empirical evidence, in addition to the proofs provided by this work, on the correctness of our method, we generated random problem instances, consisting of a set of input shapes and queries. We generated random queries first and then sampled shapes based on their vocabularies, extended with additional random components that do not occur in queries. For each sample input, we applied our algorithm to obtain a set of output shapes; to this end, we generated a set of candidate shapes and checked each for entailment. We only kept samples that produced at least one valid output shape. We covered four different scenarios labelled corresponding to the nature of the shapes involved. While *small* and *large* mainly refer to the number of shapes, the *wide* sample includes conjunction and disjunction, whereas the *deep* sample includes more deeply nested shapes.

Figure 13 shows the distribution of a few key properties across the experiments we performed, including the basic structure of queries and shapes, as well as the candidate shapes that were considered. For additional details, we refer to our extended version or the documentation of the implementation itself. The upper half of the tables describes queries, while the lower half corresponds to shapes. For both *wide* and *large* we report only data on shapes, since the queries are mostly identical to the *deep* scenario. Regarding queries, we indicate the number of basic node or edge atomic clauses in the pattern and template, the number of set, remove, or where clauses, and the distinct number of labels and keys. Regarding shapes, we include the number of sampled input shapes, their structure (number of basic components and clauses in conjunction or disjunction), the number of candidates over the vocabulary tested as potential output shapes, and the inferred set of output shapes, i.e., the subset of candidates that is entailed by the constructed knowledge base. Our scenarios were designed to cover various interactions of query structures and shapes and facilitate our validation approach (random sample graphs), not necessarily to correspond to real-world data, where a baseline is also not available.

Our validation framework operates fully on the mapped SHACL and SPARQL level, taking the input SHACL shapes mapped from ProGS as the basis for generating a random RDF graph valid regarding these shapes. The generation approach is rather brute force, to not bias the sample graphs with additional information from shapes or queries, taking only information from the vocabulary of the problem instance while removing any parts that violate input shapes. We next applied the (mapped-from G-CORE) SPARQL CONSTRUCT query to this sample graph to obtain the corresponding output graph. Finally, we validated the output graph with the inferred

<sup>7</sup> <https://github.com/softlang/s2s>

| Type            | Avg.   | Med.   |
|-----------------|--------|--------|
| MATCH           | 1.00   | 1.00   |
| CONSTRUCT       | 1.00   | 1.00   |
| WHEN Clauses    | 3.15   | 3.00   |
| SET Clauses     | 1.59   | 2.00   |
| REMOVE Clauses  | 0.99   | 1.00   |
| Distinct Labels | 3.67   | 4.00   |
| Distinct Keys   | 1.31   | 1.00   |
| Input Shapes    | 1.00   | 1.00   |
| Components      | 2.16   | 2.00   |
| Clauses         | 1.00   | 1.00   |
| Candidates      | 256.24 | 240.00 |
| Output Shapes   | 7.54   | 2.00   |

(a) Sample *small* (25,000).

| Type          | Avg.   | Med.   |
|---------------|--------|--------|
| Input Shapes  | 2.61   | 3.00   |
| Components    | 4.95   | 5.00   |
| Clauses       | 1.86   | 2.00   |
| Candidates    | 510.25 | 468.00 |
| Output Shapes | 9.85   | 2.00   |

(c) Sample *wide* (25,000).

| Type            | Avg.   | Med.   |
|-----------------|--------|--------|
| MATCH           | 1.21   | 1.00   |
| CONSTRUCT       | 2.21   | 2.00   |
| WHEN Clauses    | 3.20   | 3.00   |
| SET Clauses     | 1.57   | 2.00   |
| REMOVE Clauses  | 0.99   | 1.00   |
| Distinct Labels | 3.64   | 4.00   |
| Distinct Keys   | 1.33   | 1.00   |
| Input Shapes    | 2.63   | 3.00   |
| Components      | 2.96   | 3.00   |
| Clauses         | 1.00   | 1.00   |
| Candidates      | 408.25 | 370.00 |
| Output Shapes   | 7.53   | 2.00   |

(b) Sample *deep* (25,000).

| Type          | Avg.   | Med.   |
|---------------|--------|--------|
| Input Shapes  | 5.07   | 5.00   |
| Components    | 2.16   | 2.00   |
| Clauses       | 1.00   | 1.00   |
| Candidates    | 610.32 | 576.00 |
| Output Shapes | 11.18  | 2.00   |

(d) Sample *large* (25,000).

■ **Figure 13** Overview of statistical measures across different generator configurations.

output shapes. For each sample, we report either the class of a generation-related issue (leading to vacuous satisfaction of output shapes), output graph validation failure (shape violations), or success (non-vacuously valid graphs).

In 100,000 samples we observed *no* validation failures, providing evidence towards the correctness of our method and implementation. A total of 37,959 samples were full successes, meaning that the output graph was not only valid regarding the output shapes but also included all their targets. In 12,858 cases, the output graph only included a subset of the targets present in output shapes, while in 266 cases, the input graph included only a subset of all input shape targets; we consider these partial successes. The remainder (48,917 samples) were other structural issues, where in most cases result graphs were either entirely empty (26,591) or did not include any of the required target nodes (15,560), while in fewer cases we were unable to produce a suitable input graph (6,622), or the query exceeded the allowed execution time of 60 seconds (144). These results did not vary significantly between the different classes.

Note that in all of these cases, modulo the few timeouts where we do not know the result, the output shapes were still vacuously satisfied. Since our validation attempts each sample generation up to 100 times, we hypothesize that the reason for these cases mostly lies in the structure of the sample itself: in these cases, graphs conforming to input shapes are unlikely to be matched by the query, since both do not align well. Again, we decided against optimizing for these scenarios to avoid overfitting generated samples to *preferred* cases, which could bias results.



We validated the implementation of the validation itself with 1,000 samples over the same four classes, where output shapes (7 per sample) were drawn randomly from the set of candidate shapes instead of utilizing the inference procedure; here, we obtain 76.7% validation failures and only 0.2% complete successes, indicating a very high probability for detecting failures. Our complete results, including additional metadata for each sample, results for the individual classes of samples, all details regarding the structure and generation of samples, and the tools to replicate these and similar experiments, are provided with our implementation.

## 7.2 Runtime Performance

While the original project, on which our implementation is based, includes tooling to demonstrate feasibility in terms of runtime performance (cf. [41]), the results would be difficult to compare between the different classes of queries and shapes used in our implementation. Additionally, we lack datasets for realistic properties of G-CORE or ProGS queries and shapes. Instead, we therefore report only a preliminary indication of the feasibility of runtime performance, within the same order of magnitude as the prior results, by measuring the inference time (testing all shape candidates reported in Figure 13) for all 100,000 samples generated for the correctness validation above. Here, we observe average (median) inference times per sample of 13.94ms (11.00ms) for *small*, 34.81ms (25.00ms) for *deep*, 42.23ms (29.50ms) for *wide*, and 51.68ms (36.00ms) for *large* samples on commodity hardware (Ryzen 7 9700X), without any dedicated Scala optimization.

## 8 Related Work

Mappings between query languages and related (graph) data models have been considered for various languages (cf. [31]), including mappings among graph query languages (e.g., [48]) as well as between graph query languages and SQL (e.g., [39, 12, 46]). Closely related to our mapping problem, SPARQL `SELECT` queries have been mapped to Cypher [52, 2]. However, in this direction, the divide between RDF and PG is less challenging, given that PG can directly represent RDF graphs. Reification of PG as RDF (e.g., [33, 51]) has been discussed under consideration of querying performance [24], though only under manually defined, ad hoc reformulation of Cypher queries in SPARQL `SELECT`.

At the data model level, the lack of interoperability between RDF and PG is a known issue [5], and mappings from RDF to PG have been considered as well, e.g., [6]. In [37], the authors consider both RDF graphs and SHACL shapes, transforming them to PG and PG-Schema, respectively. RDF-star [19] is an extension to RDF for the representation of metadata as nested triples, allowing for similar features as PG, thus working towards bridging the formal gap between these data models. Indeed, both PG mappings to RDF-star [20, 24], and the reverse [1] have been defined. In particular, this also includes execution of SPARQL-star queries over PG [20].

In contrast, in our work, we leverage reification in pure RDF for the sake of utilizing DL via SHACL, and we are concerned with bridging the semantic differences between graph construction in G-CORE, with its implicit labels and properties, and SPARQL `CONSTRUCT`, with its explicit triple construction, which are not addressed in previous work.

Inference of shapes or schemas can be approached essentially in two ways:

- Option 1* – inference from instance data, e.g., concrete graphs or tables;
- Option 2* – inference from queries and other functions constructing data.

The first option has been considered for RDF using various algorithms based on logical, statistical, or ML-driven inference for constructing SHACL or ShEx [36] shapes from sample graph instances [45, 16, 38, 32, 34, 17, 9]. Similar approaches have been considered recently for PG and their schema languages [29, 11, 17].

In our work, we follow the second option and do not consider any instance data but instead infer shapes valid in the context of any possible graph that queries might operate on in the future. However, inference from concrete graphs may complement our approach when initial input shapes for queries should be determined. Our approach leverages a prior method [41], as discussed throughout the paper, but we cover PG and G-CORE through mappings; in due course, we also extend the subset of SPARQL considered in [41] with generic copy patterns.

Related work on the inference of SHACL shapes from direct mappings [49] and RML rules [14] also follows the second option, but these approaches lack the arbitrary construction through query patterns we support, in addition to explicit input constraints. Views over relational databases can be considered equivalent mechanisms to composable graph queries; SQL schemas have been inferred for such views in [25, 15, 47, 22] though suffer from infeasibility of first-order formulas (as compared to DL) or restricted semantics of functional or join dependencies, which are orthogonal to constructs supported by ProGS in our case. [10] solves a similar problem, focusing on basic cardinality constraints (as graph schemas) and formally defined graph transformations (conjunctive path queries) over a concrete query language. They, too, rely on description logics as an encoding mechanism.

## 9 Conclusion

In this paper, we have presented an algorithm for inferring ProGS shapes validating all possible property graphs constructed by queries specified in a subset of G-CORE. Our algorithm encodes constraints inferred from the underlying query, as well as any known shapes that hold on the inputs of the query. Internally, we rely on the description logic *ALCHOI* by constructing a knowledge base that entails a sound set of shapes validating all output graphs of this query.

Since this DL encoding requires a reification step for first-class edges with labels and properties, we choose a clean, intermediate abstraction layer between property graphs and *ALCHOI*, namely the SPARQL *CONSTRUCT* query language and SHACL shapes, over property graphs reified in RDF. This also allows us to reuse an underlying algorithm defined for these languages, which we must extend to support a larger subset of SPARQL, including the features required by our mapping. We prove the soundness of this shape inference approach and the semantic equivalence of the shapes and queries constructed by our mappings.

As future work, a larger, non-conjunctive subset of G-CORE could be supported by extending both the mapping and the capabilities of the underlying inference approach on SPARQL queries, supporting operators such as *UNION* in SPARQL. Such extensions may approach a limit when considering additional generic or type-level query constructs. For example, while we support a specific set of generic patterns in SPARQL, arbitrary constraints on type-level variables (i.e. variables that bind concepts) cannot be easily encoded in DL.

The query mapping itself also poses interesting questions: While in our work, we do not intend to execute the SPARQL *CONSTRUCT* queries, the mappings could inform implementations of G-CORE over SPARQL triple stores. To this end, the RDF reification would have to be investigated empirically regarding runtime and space performance and possibly adapted, given that our mapping favors a formal ease of use over efficiency concerns.

---

## References

- 1 Ghadeer Abuoda, Daniele Dell’Aglia, Arthur Keen, and Katja Hose. Transforming rdf-star to property graphs: A preliminary analysis of transformation approaches. In *Proceedings of the QuWeDa 2022: 6th Workshop on Storing, Querying and Benchmarking Knowledge Graphs co-located with 21st International Semantic Web Conference (ISWC 2022)*, Hangzhou, China, 23-27 October 2022, volume 3279 of *CEUR Workshop Proceedings*, pages 17–32. CEUR-WS.org, 2022. URL: <https://ceur-ws.org/Vol-3279/paper2.pdf>.
- 2 Lakshya A. Agrawal, Nikunj Singhal, and Raghava Mutharaju. A SPARQL to cypher transpiler: Proposal and initial results. In *CODS-COMAD 2022*:

- 5th Joint International Conference on Data Science & Management of Data (9th ACM IKDD CODS and 27th COMAD), Bangalore, India, January 8 - 10, 2022, pages 312–313. ACM, 2022. doi:10.1145/3493700.3493757.
- 3 Renzo Angles, Marcelo Arenas, Pablo Barceló, Peter A. Boncz, George H. L. Fletcher, Claudio Gutierrez, Tobias Lindaaaker, Marcus Paradies, Stefan Plantikow, Juan F. Sequeda, Oskar van Rest, and Hannes Voigt. G-CORE: A core for future graph query languages. In *Proc. of SIGMOD*, pages 1421–1432. ACM, 2018. doi:10.1145/3183713.3190654.
  - 4 Renzo Angles, Angela Bonifati, Stefania Dumbra, George Fletcher, Alastair Green, Jan Hidders, Bei Li, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Stefan Plantikow, Ognjen Savkovic, Michael Schmidt, Juan Sequeda, Slawek Staworko, Dominik Tomaszuk, Hannes Voigt, Domagoj Vrgoc, Mingxi Wu, and Dusan Zivkovic. Pg-schema: Schemas for property graphs. *Proc. ACM Manag. Data*, 1(2):198:1–198:25, 2023. doi:10.1145/3589778.
  - 5 Renzo Angles, Harsh Thakkar, and Dominik Tomaszuk. RDF and property graphs interoperability: Status and issues. In *Proceedings of the 13th Alberto Mendelzon International Workshop on Foundations of Data Management, Asunción, Paraguay, June 3-7, 2019*, volume 2369 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019. URL: <https://ceur-ws.org/Vol-2369/paper01.pdf>.
  - 6 Renzo Angles, Harsh Thakkar, and Dominik Tomaszuk. Mapping RDF databases to property graph databases. *IEEE Access*, 8:86091–86110, 2020. doi:10.1109/ACCESS.2020.2993117.
  - 7 Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
  - 8 Bart Bogaerts, Maxime Jakubowski, and Jan Van den Bussche. SHACL: A description logic in disguise. In *Logic Programming and Nonmonotonic Reasoning - 16th International Conference, LP-NMR 2022, Genova, Italy, September 5-9, 2022, Proceedings*, volume 13416 of *Lecture Notes in Computer Science*, pages 75–88. Springer, 2022. doi:10.1007/978-3-031-15707-3\_7.
  - 9 Iovka Boneva, Jérémie Dusart, Daniel Fernández-Álvarez, and José Emilio Labra Gayo. Shape designer for shex and SHACL constraints. In *Proc. of the ISWC 2019 Satellite Tracks co-located with ISWC 2019*, volume 2456 of *CEUR Workshop Proceedings*, pages 269–272. CEUR-WS.org, 2019. URL: <https://ceur-ws.org/Vol-2456/paper70.pdf>.
  - 10 Iovka Boneva, Benoît Groz, Jan Hidders, Filip Murlak, and Slawek Staworko. Static analysis of graph database transformations. In Floris Geerts, Hung Q. Ngo, and Stavros Sintos, editors, *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2023, Seattle, WA, USA, June 18-23, 2023*, pages 251–261. ACM, 2023. doi:10.1145/3584372.3588654.
  - 11 Angela Bonifati, Stefania Dumbra, and Nicolas Mir. Hierarchical clustering for property graph schema discovery. In *Proceedings of the 25th International Conference on Extending Database Technology, EDBT 2022, Edinburgh, UK, March 29 - April 1, 2022*, pages 2:449–2:453. OpenProceedings.org, 2022. doi:10.48786/EDBT.2022.39.
  - 12 Artem Chebotko, Shiyong Lu, and Farshad Fotouhi. Semantics preserving SPARQL-to-SQL translation. *Data Knowl. Eng.*, 68(10):973–1000, 2009. doi:10.1016/J.DATAK.2009.04.001.
  - 13 Richard Cyganiak, David Wood, Markus Lanthaler, Graham Klyne, Jeremy J. Carroll, and Brian McBride. RDF concepts and abstract syntax, 2014. URL: <https://www.w3.org/TR/rdf11-concepts/>.
  - 14 Thomas Delva, Birte De Smedt, Sitt Min Oo, Dylan Van Assche, Sven Lieber, and Anastasia Dimou. RML2SHACL: RDF generation taking shape. In *Proc. of Knowledge Capture Conference*, pages 153–160. ACM, 2021. doi:10.1145/3460210.3493562.
  - 15 Wenfei Fan, Shuai Ma, Yanli Hu, Jie Liu, and Yinghui Wu. Propagating functional dependencies with conditions. *Proc. VLDB Endow.*, 1(1):391–407, 2008. doi:10.14778/1453856.1453901.
  - 16 Daniel Fernández-Álvarez, José Emilio Labra Gayo, and Daniel Gayo-Avello. Automatic extraction of shapes using shexer. *Knowledge-Based Systems*, 238:107975, 2022. doi:10.1016/J.KNSYS.2021.107975.
  - 17 Benoît Groz, Aurélien Lemay, Slawek Staworko, and Piotr Wiecek. Inference of shape graphs for graph databases. In *ICDT*, volume 220 of *LIPICs*, pages 14:1–14:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ICDT.2022.14.
  - 18 Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. SPARQL 1.1 query language, 2013. URL: <https://www.w3.org/TR/sparql11-query/>.
  - 19 Olaf Hartig. Foundations of RDF\* and SPARQL\* (an alternative approach to statement-level metadata in RDF). In *Proceedings of the 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web, Montevideo, Uruguay, June 7-9, 2017*, volume 1912 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017. URL: <https://ceur-ws.org/Vol-1912/paper12.pdf>.
  - 20 Olaf Hartig. Foundations to query labeled property graphs using SPARQL. In *Joint Proceedings of the 1st International Workshop On Semantics For Transport and the 1st International Workshop on Approaches for Making Data Interoperable co-located with 15th Semantics Conference (SEMANTiCS 2019), Karlsruhe, Germany, September 9, 2019*, volume 2447 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019. URL: <https://ceur-ws.org/Vol-2447/paper3.pdf>.
  - 21 ISO. Information technology – Database languages – GQL. Standard, International Organization for Standardization, Geneva, CH, 2024.
  - 22 Barry E. Jacobs, Alan R. Aronson, and Anthony C. Klug. On interpretations of relational languages and solutions to the implied constraint problem.

- ACM Trans. Database Syst.*, 7(2):291–315, 1982. doi:10.1145/319702.319730.
- 23 JTC1. GQL Database Language, 2024. URL: <https://jtc1info.org/slug/gql-database-language>.
  - 24 Shahrzad Khayatbashi, Sebastián Ferrada, and Olaf Hartig. Converting property graphs to RDF: a preliminary study of the practical impact of different mappings. In *GRADES-NDA '22*, pages 10:1–10:9. ACM, 2022. doi:10.1145/3534540.3534695.
  - 25 Anthony C. Klug and Rod Price. Determining view dependencies using tableaux. *ACM Trans. Database Syst.*, 7(3):361–380, 1982. doi:10.1145/319732.319738.
  - 26 Holger Knublauch and Dimitris Kontokostas. Shapes Constraint Language (SHACL), 2017. URL: <https://www.w3.org/TR/shacl/>.
  - 27 Egor V. Kostylev, Juan L. Reutter, and Martín Ugarte. CONSTRUCT queries in SPARQL. In *Proc. of International Conference on Database Theory, ICDT*, volume 31 of *LIPICs*, pages 212–229. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICs.ICDT.2015.212.
  - 28 KRR Group at University of Oxford. Hermit OWL reasoner, 2008. URL: <http://www.hermit-reasoner.com/>.
  - 29 Hanâ Lbath, Angela Bonifati, and Russ Harmer. Schema inference for property graphs. In *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*, pages 499–504. OpenProceedings.org, 2021. doi:10.5441/002/EDBT.2021.58.
  - 30 Martin Leinberger, Philipp Seifer, Claudia Schon, Ralf Lämmel, and Steffen Staab. Type checking program code using SHACL. In *Proc. of ISWC*, volume 11778 of *LNCS*, pages 399–417. Springer, 2019. doi:10.1007/978-3-030-30793-6\_23.
  - 31 Mohamed Nadjib Mami, Damien Graux, Harsh Thakkar, Simon Scerri, Sören Auer, and Jens Lehmann. The query translation landscape: a survey. *CoRR*, abs/1910.03118, 2019. arXiv:1910.03118.
  - 32 Nandana Mihindukulasooriya, Mohammad Rifat Ahmmad Rashid, Giuseppe Rizzo, Raúl García-Castro, Óscar Corcho, and Marco Torchiano. RDF shape induction using knowledge base profiling. In *Proc. of the Symposium on Applied Computing*, pages 1952–1959. ACM, 2018. doi:10.1145/3167132.3167341.
  - 33 Vinh Nguyen, Hong Yung Yip, Harsh Thakkar, Qingliang Li, Evan Bolton, and Olivier Bodenreider. Singleton property graph: Adding A semantic web abstraction layer to graph databases. In *(BlockSW) (CKG) co-located with ISWC 2019*, volume 2599 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019. URL: [https://ceur-ws.org/Vol-2599/CKG2019\\_paper\\_4.pdf](https://ceur-ws.org/Vol-2599/CKG2019_paper_4.pdf).
  - 34 Pouya Ghiasnezhad Omran, Kerry Taylor, Sergio José Rodríguez Méndez, and Armin Haller. Learning SHACL shapes from knowledge graphs. *Semantic Web*, 14(1):101–121, 2023. doi:10.3233/SW-223063.
  - 35 Rossana Paciello, Luca Trani, Daniele Bailo, Valerio Vinciarelli, and Manuela Sbarra. Shape-ness: A shacl-driven metadata editor. In *Meta-data and Semantic Research - 16th Research Conference, MTSR 2022, London, UK, November 7-11, 2022, Revised Selected Papers*, volume 1789 of *Communications in Computer and Information Science*, pages 274–288. Springer, 2022. doi:10.1007/978-3-031-39141-5\_23.
  - 36 Eric Prud'hommeaux, José Emilio Labra Gayo, and Harold R. Solbrig. Shape expressions: an RDF validation and transformation language. In *Proc. of the International Conference on Semantic Systems, SEMANTiCS*, pages 32–40. ACM, 2014. doi:10.1145/2660517.2660523.
  - 37 Kashif Rabbani, Matteo Lissandrini, Angela Bonifati, and Katja Hose. Transforming RDF graphs to property graphs using standardized schemas. *Proc. ACM Manag. Data*, 2(6):242:1–242:25, 2024. doi:10.1145/3698817.
  - 38 Kashif Rabbani, Matteo Lissandrini, and Katja Hose. Extraction of validating shapes from very large knowledge graphs. *Proc. VLDB Endow.*, 16(5):1023–1032, 2023. doi:10.14778/3579075.3579078.
  - 39 Mariano Rodríguez-Muro and Martín Rezk. Efficient SPARQL-to-SQL with R2RML mappings. *J. Web Semant.*, 33:141–169, 2015. doi:10.1016/J.WEBSEM.2015.03.001.
  - 40 Philipp Seifer. softlang/s2s. Software, swbId: swb:1:dir:a38354c2cab3efb8abdc9e80f611d19aa3531d82 (visited on 2026-08-17). URL: <https://github.com/softlang/s2s>, doi:10.4230/artifacts.27685.
  - 41 Philipp Seifer, Daniel Hernández, Ralf Lämmel, and Steffen Staab. From shapes to shapes: Inferring SHACL shapes for results of SPARQL CONSTRUCT queries. In *Proceedings of the ACM Web Conference 2024, (WWW '24)*. ACM, 2024. doi:10.1145/3589334.3645550.
  - 42 Philipp Seifer, Daniel Hernández, Ralf Lämmel, and Steffen Staab. Code for From Shapes to Shapes (V1), 2024. URL: <https://darus.uni-stuttgart.de/dataset.xhtml?persistentId=doi:10.18419/DARUS-3977&version=1.0>.
  - 43 Philipp Seifer, Daniel Hernández, Ralf Lämmel, and Steffen Staab. Code for From Shapes to Shapes (V2), 2024. doi:10.18419/darus-3977.
  - 44 Philipp Seifer, Ralf Lämmel, and Steffen Staab. Progs: Property graph shapes language. In *ISWC 2021*, volume 12922 of *LNCS*, pages 392–409. Springer, 2021. doi:10.1007/978-3-030-88361-4\_23.
  - 45 Blerina Spahiu, Andrea Maurino, and Matteo Palmonari. Towards improving the quality of knowledge graphs with data-driven ontology patterns and SHACL. In *Proc. of the Workshop on Ontology Design and Patterns (WOP 2018) co-located with ISWC 2018*, volume 2195 of *CEUR Workshop Proceedings*, pages 52–66. CEUR-WS.org, 2018. URL: [https://ceur-ws.org/Vol-2195/research\\_paper\\_2.pdf](https://ceur-ws.org/Vol-2195/research_paper_2.pdf).
  - 46 Benjamin A. Steer, Alhamza Alnaimi, Marco A. B. F. G. Lotz, Félix Cuadrado, Luis M. Vaquero, and Joan Varvenne. Cytosm: Declarative property graph queries without data migration. In *Proceedings of the Fifth International Workshop on Graph Data-management Experiences & Systems, GRADES@SIGMOD/PODS 2017, Chicago, IL*,

- USA, May 14 - 19, 2017, pages 4:1–4:6. ACM, 2017. doi:10.1145/3078447.3078451.
- 47 Michael Stonebraker. Implementation of integrity constraints and views by query modification. In *Proc. of SIGMOD*, pages 65–78. ACM, 1975. doi:10.1145/500080.500091.
- 48 Harsh Thakkar, Dharmen Punjani, Jens Lehmann, and Sören Auer. Two for one: querying property graph databases using SPARQL via gremlinator. In *Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, Houston, TX, USA, June 10, 2018, pages 12:1–12:5. ACM, 2018. doi:10.1145/3210259.3210271.
- 49 Ratan Bahadur Thapa and Martin Giese. A source-to-target constraint rewriting for direct mapping. In *Proc. of ISWC*, volume 12922 of *LNCS*, pages 21–38. Springer, 2021. doi:10.1007/978-3-030-88361-4\_2.
- 50 Ratan Bahadur Thapa and Martin Giese. Optimizing SPARQL queries with SHACL. In *The Semantic Web - ISWC 2023 - 22nd International Semantic Web Conference, Athens, Greece, November 6-10, 2023, Proceedings, Part I*, volume 14265 of *Lecture Notes in Computer Science*, pages 41–60. Springer, 2023. doi:10.1007/978-3-031-47240-4\_3.
- 51 Dominik Tomaszuk, Renzo Angles, and Harsh Thakkar. PGO: describing property graphs in RDF. *IEEE Access*, 8:118355–118369, 2020. doi:10.1109/ACCESS.2020.3002018.
- 52 Zihao Zhao, Xiaodong Ge, Zhihong Shen, Chuan Hu, and Huajin Wang. S2ctrans: Building a bridge from SPARQL to cypher. In *Database and Expert Systems Applications - 34th International Conference, DEXA 2023, Penang, Malaysia, August 28-30, 2023, Proceedings, Part I*, volume 14146 of *Lecture Notes in Computer Science*, pages 424–430. Springer, 2023. doi:10.1007/978-3-031-39847-6\_33.





# Multi-Layered Legislative Knowledge Management with Property Graphs

Andrea Colombo ✉ 

Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy

Francesco Cambria ✉ 

Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy

Francesco Invernici ✉ 

Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy

## — Abstract —

The sparse nature and intricate set of relationships between legislative acts pose a significant challenge in the choice of the underlying database model, which both allows for performing structured queries and developing intuitive and smooth knowledge management. In this paper, we propose to use Property Graphs as a powerful alternative for managing legislative knowledge. First, we discuss how graph queries are a valid alternative solution to standard legislative knowledge management by showing how our data model fully captures the problem of law versioning (i.e., the existence of many versions for the same law). Then, we analyze, propose and implement innovative ways for monitoring the legislative system using Property Graph tools that have

been recently standardized and developed, such as triggers and graph-based association rules, which empower our model of advanced ways of handling legislative data. For instance, we will show how we can use these tools to develop intelligent warning systems that inform stakeholders of critical changes in legislation through active rule reasoning or to detect shifts in graph patterns via continuous monitoring of significant association patterns. Lastly, we will present an approach to expand the Property Graph model to also include laws from an institutional lower layer: regional governments. We then show how this would open new ways to exploit Property Graph tools for legislative knowledge management.

**2012 ACM Subject Classification** Applied computing → Law; Information systems → Graph-based database models; Information systems → Triggers and rules

**Keywords and phrases** data management, property graphs, triggers, law

**Digital Object Identifier** 10.4230/TGDK.4.2.4

**Category** Research

**Related Version** *Previous Version:* <https://edbticdt2025.upc.edu/files/TGD/TGD-1.pdf>

**Supplementary Material** *Dataset:* <https://zenodo.org/records/13798158> [14]

*Software:* <https://github.com/FrancescoCambria/Cypher-Queries-TGDK26> [9]  
archived at `swh:1:dir:c175195a74440b097ec11f0fb49fdc67cf616930`

**Received** 2025-07-15 **Accepted** 2026-06-22 **Published** 2026-09-03

**Editors** Stefania Dumbrava, George Fletcher, Matteo Lissandrini, and Riccardo Tommasini

**Special Issue** Data Management for (Knowledge) Graphs

**Generative AI Declaration** Generative AI was used during the preparation of this manuscript exclusively for grammatical refinement and literature discovery. The authors carefully reviewed all AI-assisted outputs and assume full responsibility for the accuracy of the final work.

## 1 Introduction

Legislative data comprises the set of acts, bills, or other normative documents produced by an authority that holds legislative power to regulate some domains on a national, regional, or international level. In most democratic countries, such power is vested in parliaments, whether



© Andrea Colombo, Francesco Cambria, and Francesco Invernici;  
licensed under Creative Commons License CC-BY 4.0

*Transactions on Graph Data and Knowledge*, Vol. 4, Issue 2, Article No. 4, pp. 4:1–4:23



Transactions on Graph Data and Knowledge

TGDK Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



national, regional, or federal, depending on the country's political system. One of the main challenges when dealing with such textual documents is converting them into machine-readable formats that enable adopting structured approaches when querying and analysing legislative acts.

To tackle this, the computer law community has devoted many efforts to proposing appropriate international standards to capture the common grounds of laws enacted in different legislative systems. Most of previous works have been based on the eXtensible Markup Language (XML) format, a semi-structured data model that has been naturally used for representing textual and hierarchical data, such as laws [32]. Relevant XML-based proposals include the Legal Knowledge Interchange Format (LKIF) [26], LegalRuleML [3] and Akoma Ntoso [4].

However, the need for a user-friendly storing of textual documents (with XML tags and hierarchical structures being ideal for such aims) collides with the possibility of conducting quantitative analysis of the complexities and features of the legislative system [30], which strongly depends on the ability to traverse relationships (e.g., dependencies and references between laws and articles). In fact, while the creation of XML formats to represent the components of an individual law (i.e., article or section structure, tags for references to other laws) is extremely useful for structured access to a document, it becomes cumbersome to manage and analyze trends and the complexity of the legislation. Furthermore, while XML-native databases generally support tools such as triggers [8], their focus on the document-level logic (e.g., handling events as document create, document update, document delete, property changes) harms the potential of *reasoning* over legislative data, leveraging relationships among acts to discover and detect anomalous patterns that require attention. For instance, abrogations of some portions of legislation can inadvertently create voids in the legal foundations of third acts. Thus, a data model that prioritizes relationships among acts is crucial in monitoring such scenarios.

In this paper, we present and discuss an innovative approach to legislative knowledge management based on Property Graphs (PG). By leveraging state-of-the-art advancements from the database community in the formal standardization of Property Graphs and their tools [2, 11, 10], we present an alternative approach for handling legislative acts, which maintains the ability to manage documents as hierarchical structures, with the creation of dedicated graph relationships, but that enriches it with tools of additional utility. First, we demonstrate how our proposal seamlessly handles textual documents by focusing on one of the main issues of legislative knowledge management, that is, the law versioning problem, i.e., retrieving the text of a law in force in a certain timestamp, thus accounting for successive modifications and/or abrogations.

Then, we show how, by explicitly modelling references among acts as graph relationships, we enable more sophisticated tools to analyze and monitor trends in the legislative activity [17], such as by the development of intelligent systems that can *reason* over legislative data through the use of active rules in the form of Property Graph triggers [11]. To this aim, we develop a trigger-based warning system that monitors the legislation, capable of detecting harmful scenarios within the legislative system, such as the creation of legislative voids. We demonstrate how we capture the latter scenario by using active rules in the form of triggers expressed in Cypher, the closest language compliant with the Graph Query Language standard. We demonstrate the effectiveness of this warning system over the Italian legislation, for which we have implemented our PG-based schema and stored it within a Neo4j database. We evaluate how warnings signaled by our reasoning-based approach significantly positively correlate with ex-post-legislative interventions that aim at fixing the legislative void, demonstrating their predictive power for a timely detection of legislative voids.

Additionally, we examine how association rules can serve as a valuable tool for knowledge discovery, offering insights into trends in the evolution of the legislative system. To this end, by leveraging the recently developed association rule operator for property graphs [10], we present an application in which association rules can aid in detecting the attitude of governments towards a

topic of particular interest, i.e., whether there is a discontinuity among successive governments. We focus on legislation related to “procurement”, and we find that, in Italy, there is a continuous significant stream of creations and abrogations of rules governing this topic, indicating a certain degree of instability in the topic in recent years.

In this paper, we aim to exploit the dual nature of the Property Graph model, which combines a network structure with rich data properties. Unlike traditional legislative models that prioritize the hierarchical organization of documents, our approach leverages both the relational connections between legal acts (the graph) and their textual content and metadata (the properties), allowing a flexible representation of both content and, as we will see, the inter-level connection among laws.

In addition, we demonstrate that tools native to Property Graphs effectively address legislative management challenges. This model provides an effective way to store and model legislative texts, leveraging the graph structure to enable better querying and reasoning over legal dependencies and identify behavioral patterns.

We leverage the flexibility of the Property Graph model to extend the existing representation of legislative knowledge by incorporating laws issued by lower institutional levels, specifically regional governments. While national legislation is often centralized and relatively uniform, regional laws introduce additional complexity due to the heterogeneous ways in which each regional government stores and publishes legal documents. In particular, we focus on integrating legal data from two regions: Lombardia and Liguria. Each region adopts different formats and publication mechanisms, which require dedicated processing pipelines. By adding this new regional layer to the Italian Legislative Graph, we not only enrich the overall structure of the graph but also introduce new opportunities for advanced legal knowledge analysis, which we later illustrate with an example.

In this work, we address three main research questions that highlight the effectiveness of Property Graphs for legislative data:

- RQ1:** How can the Property Graph model be effectively exploited to manage the evolution of legislative acts (law versioning)?
- RQ2:** How can recently developed PG tools like Cypher-like triggers and association rules mining serve legal experts in their work?
- RQ3:** How can the Italian legislative graph be extended to effectively represent different *levels* of legislation within the same graph?

**Overview.** The remainder of the paper is organized as follows. In Section 2 the Related Work is introduced. Section 3 presents the Property Graph Schema that we adopt for modelling legislative knowledge, discussing its implementation in the Italian system and how it can retrieve time-dependent versions of laws. In Section 4, we present how tools like PG triggers and graph association rules can be used to facilitate knowledge discovery and automated monitoring systems. In Section 5, we perform our experiments to demonstrate the benefits of using a graph database and its rich set of tools to manage legislative knowledge. Then, in Section 6, we describe the necessary steps for adding regional laws to the Italian graph and we show how this new layer of data can be exploited for legislative knowledge management.

## 2 Related Work

The Property Graph (PG) data model has gained significant traction for managing complex, interconnected data, serving as a flexible alternative to relational models. In [2], Angles et al. proposed PG-Schema, a standardized language for defining graph structures, which allows for the rigorous validation of node and edge types within a property graph. In addition, a new standard for GQL, the emerging query language for property graph databases, has been proposed in [25].

Property graphs have been enriched with active data management tools. In [11], we proposed a formalization of triggers, originally a staple of relational database systems, for property graphs. The PG-Triggers enable reactive behaviors, such as automatically detecting complex events or maintaining consistency, directly within the graph storage. In [10], instead, Cambria et al. present the MINE GRAPH RULE operator to adapt association rules mining from transactional databases to property graphs.

While legislative data has traditionally been managed as static documents [26, 3, 4], recent approaches have demonstrated the utility of Property Graphs for capturing the dynamic nature of legal systems. In [19], we proposed the first approach for representing the Italian legislative system, in terms of documents and references, through a Property Graph, together with its schema [2]. While our proposal was country-specific, its foundation relies on an internationally adopted standard (OASIS) [36], which aimed to define common elements that characterize legislative acts across multiple legislative traditions [46]. For instance, in [20] we constructed the graph reflecting the legislation of the United States, modeled around the same graph schema.

Other than Property Graphs, RDF (Resource Description Framework) and Knowledge Graphs based on Semantic Web standards have been used to manage legislative knowledge exploiting graphs. For instance, in [37] Oliveira et al. propose an RDF model for a legislative system and apply it on the Brazilian legislation. In [43], the authors propose an approach based on a Knowledge Graph to model legal documents so that the legal expert can design queries to easily retrieve relevant information.

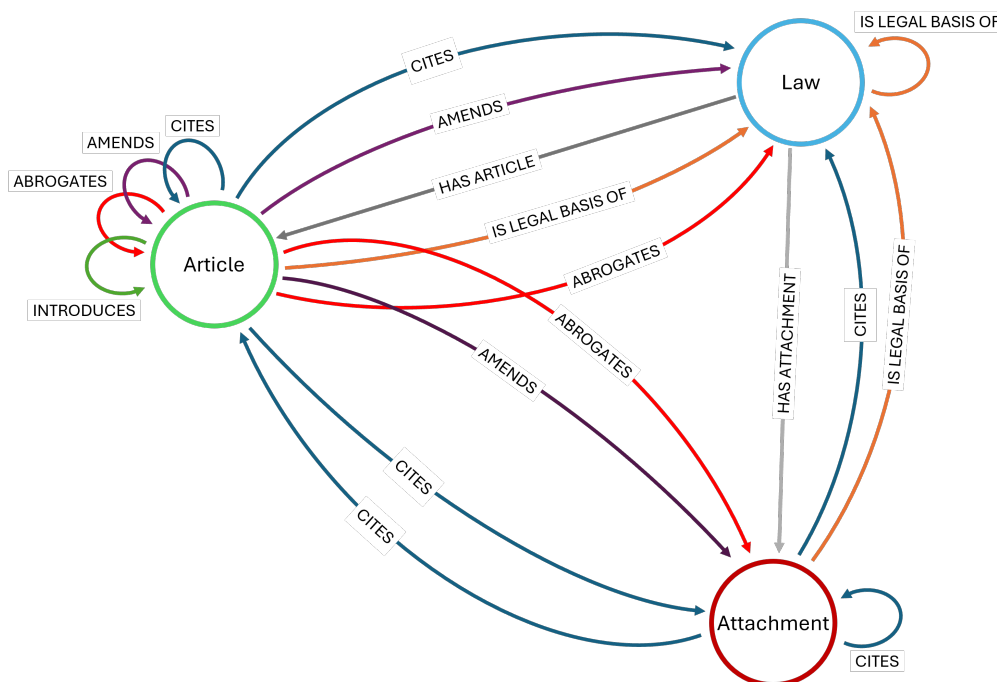
### 3 Property Graphs for Legislative Data

Modelling legislative data and the interconnections between laws and articles in a property graph offers a structured and dynamic approach to understanding and managing the knowledge of a legislative system.

In this section, we present a Property Graph Schema that aims to handle legislative data and discuss how we can leverage the expressiveness of a graph query language, such as Cypher, the closest GQL-compliant language, to handle a fundamental feature in legislative knowledge management: the retrieval of the correct version of a law based on a desired timestamp.

#### 3.1 Property Graph Schema

In Figure 1, we visually present the schema we proposed in [19] for the Italian legislation. In a graph, nodes can represent laws connected to their articles (or, according to the legislative tradition, sections, clauses, etc.) through a parthood relationship. Metadata relevant to the law, such as title, publication date, and law domain (indicating the ministries responsible for the law's content), are assigned as properties directly to the law node, enabling a structured approach to query and analyze the legislative corpus. Articles are distinct nodes with properties that detail the specific sections of the law they represent. These properties include the article text, any headings or titles specific to the article, and a list of policy-related topics that the article governs. Attachments, which serve a different function within a law document than articles, are also modelled as distinct nodes with their characteristics. Law, article and attachment nodes are interconnected using a set of directed edges that capture references, dependencies, and amendments that denote the evolution of a legislative system. We included four fundamental types of relationships which we identified as generic enough across multiple legislative traditions: an *“is the legal basis of”* dependence, an *amends*, an *abrogates* and a more generic *cites* reference. The classification of references in the schema categories follows the official annotations present in the original sources. Such a dataset of



■ **Figure 1** Graphical visualization of the Property Graph Schema. According to this schema, legislation can be represented by three node labels, (`:Law`), (`:Article`), and (`:Attachment`). Laws can be the legal basis with each other, or have articles and attachments. Articles and attachments can cite each other or laws, and be the legal basis of laws. Finally, articles can also introduce other articles, or amend and abrogate laws, attachments, or other articles. – *Properties are omitted for better visualization.*

official annotation can be used to train reference classifiers, based on NLP models as BERT, to classify edges in scenarios where the annotation is not available (i.e., in the regional layer, as we will see).

The formal PG-Schema, as defined in [2], that we use to model legislative data is depicted in Figure 2.

Regarding edge properties, all defined types of references share the common attribute *paragraph*, which denotes, if applicable, the specific portion of the article’s text that is being referenced. For instance, one could be interested in citing or amending only a specific portion of text within the article. Thus, such a feature can be captured by inserting the paragraph of interest. As a consequence of such a modelling choice, an *abrogates* edge whose *paragraph* property is NULL implies a full abrogation of the destination article. Such a modeling choice follows the empirical observation that a complete repeal of an article never occurs by repealing all its paragraphs on an individual basis; this means that, a complete repeal occurs regardless of the specific paragraph previously deleted. A law becomes abrogated whenever all its articles have been abrogated. Thus, a law abrogation can be inferred by looking at whether all its articles have an *ABROGATES* edge without the *paragraph* property (which would denote a partial abrogation).<sup>1</sup>

Finally, *amends* and *abrogates* edges are the only types of references that also share a second property: *newText*. Such (textual) property captures the new text of the destination article node, as modified/abrogated by the source article node.

<sup>1</sup> Abrogation edges, according to our model, can theoretically also be global repeals, deleting an entire law (i.e., a *Law* node is the destination node). However, a global repeal can be equally achieved by creating multiple abrogation edges which delete each article of the destination law. Consequently, in the rest of the paper, we will only consider the scenario of abrogation edges with an *Article* node as destination.

```

CREATE GRAPH TYPE lawsGraphType STRICT{(lawType: Law {id STRING, title STRING,
  lawNum INT, typeLaw STRING, publicationDate DATE, numArt INT, numAttach INT}),
(articleType: Article {id STRING, title STRING, number INT}),
(attachmentType: Attachment {id STRING, title STRING, type STRING}),
(:lawType)-[hasArticleType: has_article]->(:articleType),
(:lawType)-[hasAttachmentType: has_attachment]->(:attachmentType),
(:lawType)-[referenceType: is_legal_basis_of]->(:lawType),
(:articleType)-[referenceType: is_legal_basis_of|amends|abrogates|cites]->(:lawType),
(:articleType)-[referenceType: amends|abrogates|cites
{paragraph STRING, newText STRING}]->(:articleType),
(:articleType)-[referenceType: introduces]->(:articleType),
(:articleType)-[referenceType: amends|abrogates
{paragraph STRING, newText STRING}]->(:attachmentType),
(:attachmentType)-[referenceType: is_legal_basis_of|cites]->(:lawType),
(:attachmentType)-[referenceType: cites{paragraph STRING}]->(:articleType),
(:attachmentType)-[referenceType: cites {paragraph STRING}]->(:attachmentType)}

```

■ **Figure 2** PG-schema of the legislative data model proposed.

**The Italian Legislative Property Graph.** We implemented our schema for the Italian legislation and derived a graph consisting of Italian national laws. The graph is stored in a Neo4j database (available at [16]), comprising 74k law nodes, 318k article nodes, 127k attachment nodes, 107k legal basis dependencies, 64k abrogations, 80k amendments, and 228k generic citations. The graph is actively maintained, loading any new laws or modifications daily. For the aim of this paper, we will use acts and laws as interchangeable terms to refer to the official normative documents produced by the Italian institutions. Articles represent portions of the law instead, following the structure of the law. Attachments represent a particular type of articles, e.g., tables, figures or other annexes that are part of the law.

As discussed in [19], the quality of the graph of Italian legislation was evaluated using a multidimensional framework that focused on accuracy, completeness, and consistency. The ETL pipeline achieved high accuracy, correctly inferring 98% of true law abrogations, as demonstrated by comparing the legacy system (the official portal of Italian legislation, which stores updated versions of laws). The high consistency was evaluated by detecting material errors in edge classification using text diffs as a heuristic to classify edge types, which accounted for only 1.8% of total edges.

### 3.2 Managing Law Versioning via Graph Queries

Legislative systems evolve over time, generating a continuous stream of new laws that amend or repeal existing ones. Each change represents a new version of the law, capturing the updated text whenever a modification occurs. This process often results in rapidly growing use of textual documents, as even minor changes necessitate the storage of additional files with the novel text.

By adopting the graph model and the schema that we proposed, we can seamlessly handle such features by (i) storing the original textual version of each article of law within a node property, (ii) storing the modified textual version, as per amended by successive laws, as an edge property and (iii) delegating the computation of the in-force text for a law at a certain timestamp to a graph query, that, by leveraging publication dates available in law nodes, is capable of always inferring the desired law version. By doing so, we avoid storing multiple textual versions of the same law, achieving a more efficient storage of legislative data and, at the same time, allowing a more structured and straightforward approach for querying statistics or detecting trends of a legislative system.

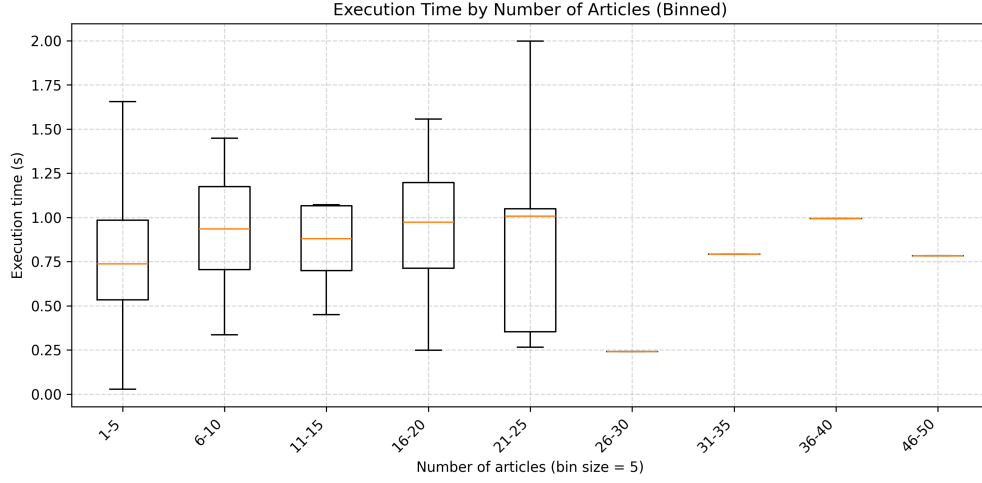


**Figure 3** Annotated Cypher query deriving the in-force text of law IDLAW at a certain TIMESTAMP. In red, the two variables to be set according to the user's requests. First, a subquery (within the CALL construct) is used to derive and union the law articles in-force at the desired timestamp. Then, a COLLECT function is used to produce the correct law text as a document.

In Figure 3, we present the query in Cypher that we designed to manage law versioning based on our proposed schema. Such a query strongly relies on navigating graph patterns to combine temporal information. In detail, by specifying a TIMESTAMP  $T$  and a law of interest  $L$  (i.e., an IDLAW), we can get the text version of  $L$  at  $T$  by performing a union that concatenates articles of  $L$  in their latest modified version, as per modified by the law with the maximum publication date before  $T$ , with articles that have never been amended or have been amended only after  $T$ , thus do not have any incoming edge of types ABROGATES and AMENDS.

Finally, we note how this approach is replicable even on a more granular level, i.e., by considering paragraphs as graph nodes. The benefit would be even more pronounced since most of the modifications change only some words of a law, and a standard document versioning approach would require the storage of an additional document, even for tiny substitutions. Here, the challenge would be consistently modelling paragraph nodes and linking textual references to such nodes, since references do not always explicitly mention the paragraph, as we experienced with the Italian legislation.

In Figure 4, we computed the query execution times of law versions based on the number of articles to evaluate the scalability of our versioning approach. The results indicate that the execution time remains consistently low, with the median duration for all bins staying within a narrow range of approximately 0.75s to 1.1s. Although we observe a higher variance in the 21–25 article bin – likely due to specific document complexities or system cold starts, there is no discernible trend in execution time as the number of articles increases. This stability suggests that the query performance is largely independent of the document size within the tested parameters, confirming that our indexing strategy effectively mitigates the overhead typically associated with larger legal datasets.



**Figure 4** Execution time of the versioning query (Figure 3) grouped by the number of articles, from a random sample of 250 laws. The median execution time remains relatively constant (under 1.25s) across all bins, suggesting that query performance scales well regardless of the article count.

```
(l1:Law)-[:HAS_ARTICLE]->(a1:Article)<-[:ABROGATES|AMENDS|INTRODUCES]-(a2:Article)
<-[:HAS_ARTICLE]-(l2:Law)
```

**Figure 5** Path for amending laws to modified articles.

### 3.3 Benefits of a Legislative Property Graph Modeling

In this section, we examine the practical motivations for modeling legislative data as a property graph rather than a traditional relational schema. To illustrate the advantages, consider Figure 3, which depicts a core operation of a legislative database: computing updated text versions while maintaining a historical record of modifications. In our model, this is efficiently handled through graph patterns, as shown in Figure 5.

This pattern directly connects an amending law to the modified law via its articles. While various relational schemas could be implemented to represent this (also in a more compact form, which, however, might lose some other features), a direct SQL translation approach of our schema would be significantly more verbose and computationally intensive. In a relational system, traversing these relationships typically requires navigating multiple associative (many-to-many) tables or complex self-referencing foreign keys. Furthermore, because each relationship type (ABROGATES, AMENDS, and INTRODUCES) has some unique properties, thus a normalized relational model would likely require dedicated join tables for each, further increasing query complexity.

Beyond performance, the graph approach offers superior semantic readability. In a property graph, the query syntax visually maps to the conceptual domain model. This reduces the cognitive gap between a legislative researcher’s question and the technical code required to answer it.

## 4 Triggers and Association Rules

Triggers have been a fundamental feature since the inception of relational databases [24]; they were explored in depth in [47] and formalized in the ISO-ANSI SQL3 Standard [35]. Traditionally, triggers have been utilized to automate tasks that facilitate the development of automation



systems [41]. In graph databases, adding reactive components has also been helpful in supporting important applications, such as the monitoring of pandemic events [11], which naturally are modeled into knowledge graphs stored in graph databases. Formalizations of triggers for graph databases and Property Graphs have recently been proposed [11, 5]. Due to the richness of the graph data model w.r.t. the relational model, triggers can also be leveraged to encode more complex (and interesting) events. For instance, since property graph items can be either nodes or relationships, triggers can include the creation and deletion of nodes/relationships as well as the setting and removal of their labels and properties. The strong analogy between triggers, i.e., active rules, and deductive rules, as defined in [12], has also enabled the use of triggers for adopting *reasoning* approaches over data modelled into knowledge graphs stored as property graphs.

Association rules (AR) mining was first introduced in [1, 44] and aim to uncover patterns or correlations within data. Classically, association rules take the form of  $X \rightarrow Y$  where  $X$  is referred to as the body of the rule and  $Y$  as its head. Both body and head represent predicates or conditions that can be validated within the data, based on two key metrics: support and confidence. Association rules mining has been applied in many different domains: for instance, for highlighting which items are frequently purchased together [13, 38], for detecting connections between symptoms and diseases in medicine [45], or to uncover patterns of user interest in social networks [42]. Like triggers, the flexibility and richness of the graph data model allow association rule mining to uncover more insightful patterns within the data. The MINE GRAPH RULE operator, introduced in [10], leverages all the capabilities of the graph query language to simplify the definition and extraction of association rules from graph databases. While the operator is not fully integrated into the standard Cypher language, its syntax closely follows Cypher, ensuring clarity and readability. Furthermore, the queries used to extract association rules can be implemented in any graph database management system that supports GQL.

In the rest of this section, we present two practical applications of such tools, demonstrating their contribution to implementing more sophisticated monitoring activities in the legislative domain.

## 4.1 Warnings for Legal Basis Abrogation

In the legislative context, reasoning through active rules can be used to create automatic monitoring systems that derive warnings for anomalous events based on graph patterns. We identified an important application of such a feature in monitoring events related to repealing key acts that serve to form a *legal basis* for other pieces of legislation. Such abrogations might create legislative voids and hinder the applicability of third laws, which have their foundation in the repealed law. In other words, the content and rules stated in an act might depend on legislation abrogated by the newly enacted law.

In our schema, we modelled legislative dependencies through **IS LEGAL BASIS OF** edges, which define the foundations of a given law. Thus, to implement the application, we can utilize the graph structure to generate timely warnings about potentially hazardous situations that require monitoring. Such warnings take the form of new special edges that can be visually inspected. In detail, we can define the following deductive rules that generate warnings for this problem:

1. Whenever a new law containing abrogations is published, we *compute* whether the new law has repealed an older law by leveraging the graph schema, as discussed in Section 3. In such cases, we *merge* **REPEAL** edges that connect the new law node with the abrogated law nodes.
2. By traversing the **IS LEGAL BASIS OF** edges, a second reasoning rule monitors whether the **REPEAL** of a law has created a legislative void in the legal foundation of a third in-force law. For such cases, it creates a warning edge that informs about the potentially harmful situation (which we denote as a **UNDERMINES** edge).

A visual example of the reasoning process is available in Figure 6.

---

**Algorithm 1** Legislative Warning Generation.

---

**Input:**  $\mathcal{G} = (V, E)$  (Graph),  $L_{new}$  (New Law)  
**Output:**  $\mathcal{G}' = (V, E')$  (Updated Graph with new REPEAL and UNDERMINES edges)  
**// Phase 1: Aggregate Article Abrogations into Law Repeals**  
 $a := article$   
 $\mathcal{L}_{target} = \{L | (L) \xrightarrow{HAS\_ARTICLE} (a) \wedge (L_{new}) \xrightarrow{ABROGATES} (a)\}$   
1: **for each**  $L_{target}$  **in**  $\mathcal{L}_{target}$  **do**  
2:    $A_{total} \leftarrow \text{count } a \text{ where } (L_{target}) \xrightarrow{HAS\_ARTICLE} (a)$   
3:    $A_{abrogated} \leftarrow \text{count } a \text{ where } (L_{new}) \xrightarrow{ABROGATES} (a)$   
4:   **if**  $A_{abrogated} \geq A_{total}$  **then**  
5:     **merge**  $(L_{new}) \xrightarrow{REPEAL} (L_{target})$   
6:   **end if**  
7: **end for**  
**// Phase 2: Detect Impacts on Dependent Legislation**  
 $e := edge$   
 $\mathcal{E} = \{\forall e | e : (L_{new}) \xrightarrow{REPEAL} (L_{target})\}$   
8: **for each**  $e$  **in**  $\mathcal{E}$  **do**  
 $\mathcal{L}_{dep} = \{\forall L_{dep} | (L_{target}) \xrightarrow{IS\_LEGAL\_BASIS\_OF} (L_{dep})\}$   
9:   **for each**  $L_{dep}$  **in**  $\mathcal{L}_{dep}$  **do**  
10:     **if**  $\nexists L \xrightarrow{REPEAL} L_{dep}$  **then**  
11:       **merge**  $(L_{new}) \xrightarrow{UNDERMINES} (L_{dep})$   
12:     **end if**  
13:   **end for**  
14: **end for**

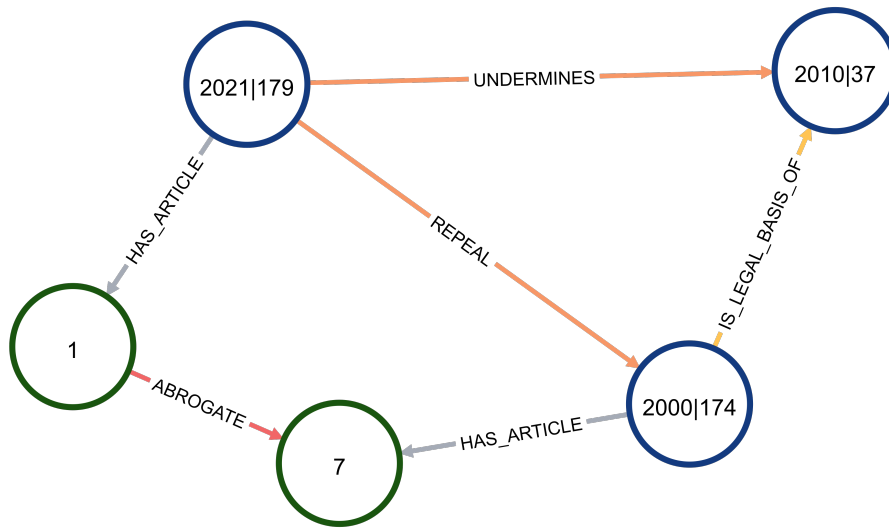
---

The logic described in the pseudocode in Algorithm 1 is implemented via Cypher triggers. The first phase (line 1 to 7) monitors the insertion of new **ABROGATES** edges; for each of these edges, we count the total distinct number of abrogations to each article of a law (line 2, considering only edges whose *paragraph* property is null (denoting full article abrogation)). When all articles have been abrogated, then a new **REPEAL** edge is created (line 4 – 6). The second phase captures the newly created **REPEAL** edges and checks for dependent laws ( $L_{dep}$ ) (lines 8 – 9) that lack repealing edges (line 10): for such cases, a new **UNDERMINES** edge is merged, highlighting the possibility of the creating a potential harmful legislative void.

More in detail, the first rule can be encoded within the trigger expressed in Cypher in Figure 7, a possible implementation of PG-Triggers, as defined in [11].

In the PG-trigger, the **UNWIND** clause captures, within the Neo4j ecosystem, the insertion of a new **ABROGATES** edges which derive from the insertion of a new law node. If a graph pattern denoting an article abrogation without the *paragraph* property exists (denoting, according to our schema, a full abrogation), then the full abrogation is tested by monitoring whether all its articles have been repealed. In such a scenario, the trigger *merges* a new edge denoting an abrogation (i.e., a **REPEAL**) of the destination law node.

The second rule can be implemented as shown in Figure 8, where, here, the **UNWIND** clause captures the newly created **REPEAL** edges and checks whether there exists a non-abrogated law whose the repealed one is the legal basis of. Within the **WHERE** conditions, we exclude the cases in which the repealing law already contains any explicit “correction” to the destination law and its articles. If such a pattern exists, we merge new derived relationships, i.e., **UNDERMINES** edges, which denote a potential legislative void since the newly published law is deleting a law that is



■ **Figure 6** Law 2021/179 repeals law 2000/174, by abrogating its remaining in-force article (i.e., article 7). However, since law 2000/174 is a legal foundation of law 2010/37, law 2021/179 also undermines its validity since its foundation has been removed, raising a potential legislative void.

```

UNWIND $createdRelationship AS newRel
MATCH p=(law:Law)-[:HAS_ARTICLE]->(art:Article)<-[newRel:ABROGATES]-(art2:Article)
      <-[:HAS_ARTICLE]-(newLaw:Law)
WHERE newRel.paragraph IS NULL
WITH law, art, newLaw
MATCH (law)-[:HAS_ARTICLE]->(art)<-[abr:ABROGATES]-(newLaw)
WHERE abr.paragraph IS NULL
WITH law, law.numArt AS NUMART, COUNT(DISTINCT art) AS NUMREPEALS, newLaw
WHERE NUMREPEALS >= NUMART
MERGE (newLaw)-[rep:REPEAL]->(law)
  
```

■ **Figure 7** PG-Trigger encoding the first rule.

used as a legal foundation in a third law. Thus, such edges require attention from the legislator and any interested stakeholders in the domain of the law. The warnings might translate into corrective actions, i.e., new laws or articles that address potential legislative voids.

These triggers can be executed asynchronously, thus ensuring that the monitoring system does not introduce significant overhead during the ingestion of large legislative datasets. This reactive approach transforms the graph from a passive repository of legal texts into an active decision-support system. Specifically, the emergence of an **UNDERMINES** edge serves as a semantic alert for legal experts. Instead of manually auditing the cascading effects of a repeal – a task that grows more complex with the size of the corpus – legislators can query for these specific patterns to prioritize their interventions.

```

UNWIND $createdRelationships AS newRel
MATCH (art)<-[:HAS_ARTICLE]-(newLaw:Law)-[newRel:REPEAL]->(law:Law)
      -[:IS_LEGAL_BASIS_OF]->(law2:Law)-[:HAS_ARTICLE]->(art2:Article)
WHERE NOT EXISTS ((art)-[:REPEAL]->(law2))
AND newLaw.id <> law2.id
AND NOT EXISTS ((art)-[:ABROGATES|CITES|AMENDS|INTRODUCES]->(art2))
MERGE (newLaw)-[:UNDERMINES]->(law2)

```

■ **Figure 8** PG-Trigger encoding the second rule.

```

MINE GRAPH RULE FrequentChangesOnTopic
GROUPING ON (topic:Topic)
WHERE t.name CONTAINS '$topicName'
DEFINING
  BODY AS (topic)<-[:OF_TOPIC]-(art:Article)<-[:HAS_ARTICLE]-(law:Law)
          <-[:CREATED_UNDER]-(gov:Government)-[:SUCCEEDED_BY]->(gov2:Government)
  HEAD AS (topic)<-[:OF_TOPIC]-(art2:Article) <-[:ABROGATES]-(art3:Article)
          <-[:HAS_ARTICLE]-(law2:Law)<-[:CREATED_UNDER]-(gov3:Government)
WHERE gov2 = gov3
IGNORE art, law, gov2, art2, art3, law2
EXTRACTING RULES WITH SUPPORT > 0.2
      AND CONFIDENCE > 0.2

```

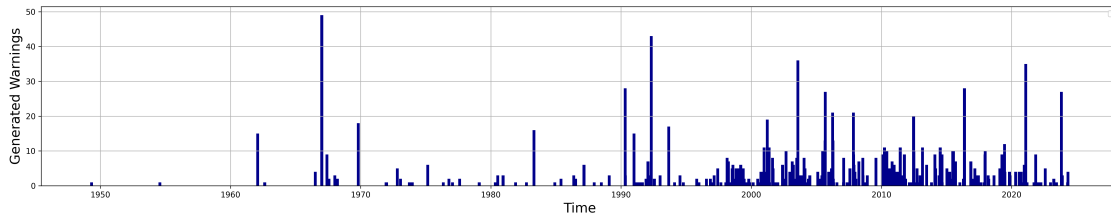
■ **Figure 9** Mine Graph Rule operator for extracting governments behavioural patterns.

## 4.2 Patterns of Government Attitude

In a legislative graph proposed in Section 3, items represent laws, articles, and attachments. Such a schema can be easily enriched by adding governments and topic nodes<sup>2</sup>. The former represents the governments under which laws have been enacted; the latter connects each legislative node to the topics its content refers to. With these enrichments, we can exploit (graph) association rules to understand which and if governments have the attitude to profoundly change legislation referring to a topic for which its predecessor government had also previously implemented legislation. Through this application, we can detect *temporal graph patterns* that monitor the behaviour of governments and identify discontinuity evolution of the legislation. The mining of this kind of pattern can be achieved by adopting the MINE GRAPH RULE operator, whose syntax was proposed in [10], and can be written as shown in Figure 9. In [10], the operator’s performance was also evaluated, proving that it can extract rules reasonably fast on graphs with similar dimensions. For this reason, we chose to exploit it for deeper analysis of the Italian Legislation graph.

In particular, through the operator we extract whether *a certain topic of interest, regulated under a certain government, is also significantly reshaped by its direct successor in government*. Therefore, the output of such a rule is statistically significant patterns of discontinuity that allow us to monitor a possible important shift of regulation from one government to another regarding a certain topic. We highlight how such output is more high-level than looking for modifications and abrogations that a government made to the legislation implemented by its predecessor. In fact, the complexity of the legislative system hinders the result of such a query since these shifts could also be captured indirectly through third laws, which share the same (or similar) topic.

<sup>2</sup> Multiple approaches can be used to derive topics from textual documents, either semi-supervised ones [6] or state-of-the-art LLM-based topic extraction tools [34]



**Figure 10** Warnings generated by our trigger-based reasoning system, aiming to monitor the abrogations of relevant laws. Each bar represents the number of warnings generated whenever a new law is published, i.e., the number of **UNDERMINES** relationships that are merged within the graph.

In addition, we also leveraged association rules to study other topological structures. For instance, we uncovered citation patterns from articles to laws. For instance, high-confidence rules reveal that articles citing “data protection” laws from the Economy and Communications departments consistently cite “privacy” legislation as well.

## 5 Implementation and Results

In this section, we execute the proposed triggers and association rules within the Italian legislation that we have already modelled in a property graph [16], aiming to demonstrate the efficacy and utility of such tools in a real-world scenario. In particular, we demonstrate how warnings generated by our triggers exhibit a significant correlation with ex-post interventions, i.e., correction measures that modify or add details to the law signalled by the alert system based on triggers. Then, we implement an AR in the Italian graph to analyze whether we can monitor and identify patterns in legislation related to procurement rules, which are under intense scrutiny in daily Italian news. We discover that in recent years, many governments have had a reshaping attitude toward such legislation, with profound changes and discontinuity among recent governments, which harm the stability of legislation, one of the drivers of economic growth [21].

### 5.1 Relevance of Warnings

As discussed in Section 4.1, we consider the emergence of a **UNDERMINES** edge, derived via reasoning with triggers, as a warning. We implemented the triggers in a Neo4j database, which we also used to store the graph of the Italian legislation. To implement triggers as reasoning rules, capable of inferring novel knowledge through deductions, as required by this use case, we made use of a publicly available Neo4j plugin<sup>3</sup> that, through the use of a trigger controller, manages the correct execution order of triggers that enable reasoning.

A full quantitative assessment is not feasible since there is no ground truth that allows the derivation of a confusion matrix. Thus, to test the significance of these warnings, we adopted a historical analysis strategy: we replicated the (ordered) insertion of each law within the graph to let the Neo4j triggers activate based on the legislative situation at each timestamp.

First, we computed the number of warnings generated for each law, which we depict in Figure 10. Then, we correlated these values with the number of ex-post interventions that the legislator had to make within the year after the law’s publication to address or rectify a legislative void. We count interventions as the distinct number of references to the “undermining” law that occurs in the following 365 days after the official publication. Thus, for each law that activates the triggers,

<sup>3</sup> The implementation is available at [33]

■ **Table 1** Sample of the warnings generated by a repealing law and the number of legislative interventions that have occurred, within one year after the publication of the repealing law, to third laws having as foundation a law abrogated by the “repealing” law.

| Repealing Law | Warnings | Legislative Interventions |
|---------------|----------|---------------------------|
| 1949/264      | 1        | 4                         |
| 1954/615      | 1        | 0                         |
| 1967/18       | 49       | 20                        |
| 1967/601      | 9        | 3                         |
| ...           | ...      | ...                       |

we get two values: the number of warnings generated and the number of interventions. In Table 1, we present a portion of the two values that we computed for laws that have been signalled as warnings.

Finally, we compute the Pearson correlation coefficient to measure the linear relationship between the two datasets. The idea is that a significant positive correlation coefficient implies that the higher the number of warnings generated, the higher the probability of necessary intervention to address the legislative void, indicating that warning counts are associated with the likelihood of subsequent intervention. Since the distributions of warnings and interventions are not normally distributed, we adopted a bootstrap approach [23, 40] to estimate the sampling distribution of the correlation coefficient and to test whether it is significantly different from zero. We observed a Pearson coefficient of 0.394 and a p-value of 0.02 (within a confidence interval of [0.269, 0.506]), indicating a significant positive relationship between the two series. This can be interpreted as a higher probability of intervention whenever more warnings are generated, i.e., a central law is repealed and requires more interventions.

While the positive correlation provides a quantitative foundation for the system’s predictive power, we acknowledge that correlation alone does not account for all nuances of legislative complexity; for instance, it cannot establish direct causality or distinguish between the extension of different interventions. In this case, legal experts can be involved to evaluate and refine the reasoning process, ensuring the warnings effectively capture the most critical legislative voids and account for the material nature of the corrections.

## 5.2 Evolution of Procurement Regulation

As a representative example, we identified regulation regarding “procurement” as an interesting topic to analyze through association rules. The theme is under scrutiny by the general public for its long-standing episodes of corruption in the country, as also studied in dedicated economic research [22]. Through the running of our association rule tool for this topic, we aim to monitor the behaviour of governments in continuously modifying legislation in this area.

We implemented the MINE GRAPH RULE operator in Neo4j<sup>4</sup> as in previous work (see [10]). The output of the AR run for the procurement legislation is presented in Table 2. Considering the number of topics in the graph, setting the support and confidence thresholds to 0.2 effectively identifies strict rules. Out of the 69 governments of the Italian Republic, 16 have significantly reshaped the topics their predecessor had worked on. An alarming trend is that this “practice” has intensified in recent years. Notably, 15 out of the 16 governments identified began after 1990.

<sup>4</sup> The implementation is available at [27]

■ **Table 2** Output of the MINE GRAPH RULE operator used to discover patterns of changes in procurement regulation. Results have been ordered by the temporal dimension, i.e., starting from the most recent pair of adjacent governments, and depict a scenario in which the government in the body column enacted relevant legislation about “procurement” which was profoundly reshaped by the (successor) government in the head column.

| Head<br>Government | Body<br>Predecessor | Support | Confidence |
|--------------------|---------------------|---------|------------|
| I Meloni           | I Draghi            | 0.57    | 0.80       |
| II Conte           | I Conte             | 0.29    | 0.66       |
| I Conte            | I Gentiloni         | 0.57    | 1.00       |
| I Gentiloni        | I Renzi             | 0.57    | 0.80       |
| I Renzi            | I Letta             | 0.28    | 1.00       |
| I Letta            | I Monti             | 0.28    | 0.66       |
| I Monti            | IV Berlusconi       | 0.57    | 0.80       |
| IV Berlusconi      | II Prodi            | 0.57    | 1          |
| II Prodi           | III Berlusconi      | 0.43    | 0.5        |
| II Berlusconi      | II Amato            | 0.29    | 1          |
| II D'Alema         | I D'Alema           | 0.29    | 0.33       |
| I D'Alema          | I Prodi             | 0.29    | 0.5        |
| I Prodi            | I Dini              | 0.29    | 0.5        |
| I Ciampi           | I Amato             | 0.29    | 1          |
| VII Andreotti      | VI Andreotti        | 0.43    | 0.75       |
| III Moro           | II Moro             | 0.43    | 1          |

Another concerning behavior highlighted by the extracted rules is that governments succeeding those of a different political alignment are more likely to repeal pieces of procurement legislation, suggesting a dismantling of the predecessors' work. For instance, one of the rules with the highest support and confidence values reveals that the fourth Berlusconi government (right-wing) reshaped procurement legislation previously modified or enacted by the second Prodi government (left-wing).

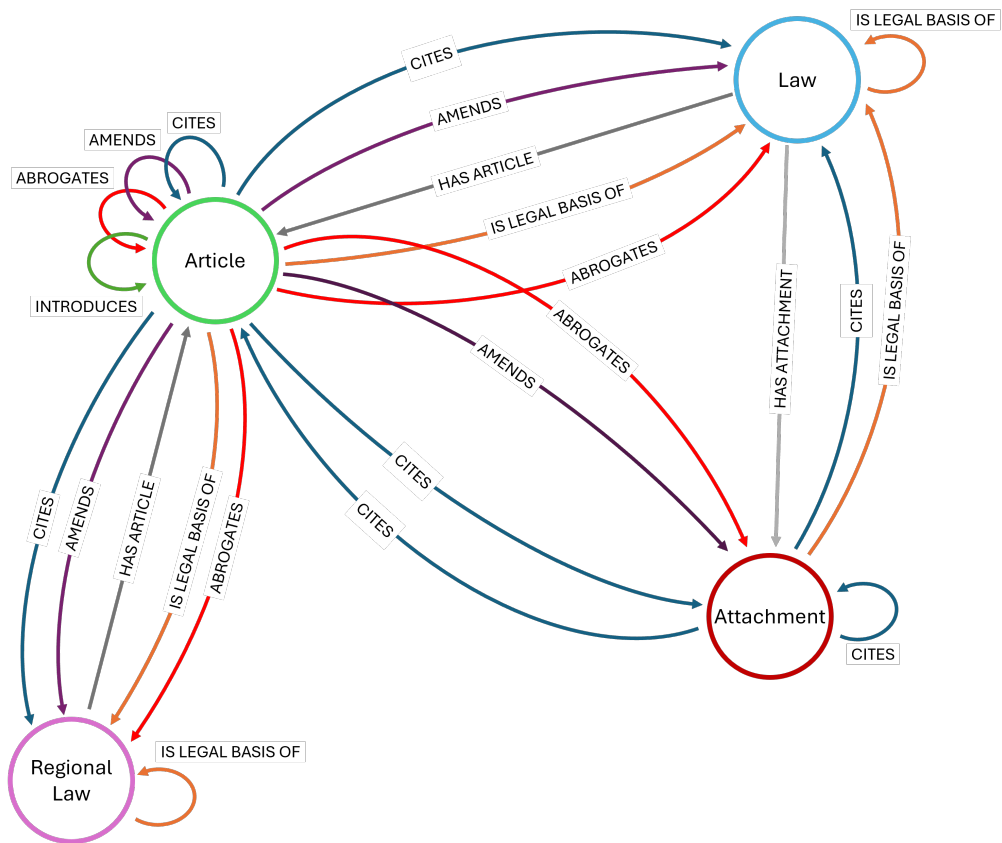
The utility of these extracted patterns extends beyond simple historical description, providing a quantitative metric for regulatory instability. High confidence values (e.g., 1.00 in several transitions) serve as a proxy for **legal uncertainty**, identifying periods where legislative continuity is entirely broken.

This transformation of the legislative graph into a set of predictable association rules provides a data-driven foundation for evaluating the quality and reliability of a country's institutional environment.

## 6 Adding the Regional Layer to the Graph

Legislative systems are complex not only because of how documents relate to each other, but also because they are built from many different sources. These sources contribute to a rich and layered structure of meanings and interpretations. So far, in the process of building the Italian legislative graph, our work has mainly focused on national laws issued by central institutions such as the Parliament and the Government. However, the Italian legal system is much more distributed, and there are other institutions, like regional governments, that produce legal documents. These additional sources must be taken into account to create a more complete and representative model of the legal landscape in Italy.





■ **Figure 11** Graphical visualization of the updated Property Graph Schema including Regional Laws. The update schema included an additional node label `:RegionalLaw`, that has the same connections to `(:Article)` nodes as `(:Law)` nodes.

Therefore, as shown in Figure 11, we extended the Property Graph Schema. We introduced a new node label, **RegionalLaw**, which represents regional legislative documents. These nodes store metadata specific to regional laws but, like their national counterparts, they connect to **Article** nodes that contain the actual legal text.

In this updated model, **Article** nodes can reference both national and regional laws. They can cite other laws, serve as the legal basis for new laws, and participate in legal modifications such as amendments or abrogations. Importantly, these relationships are not limited to within the same jurisdiction: regional articles may refer to national laws, and vice versa, capturing the interconnected nature of Italy’s multi-level legal system. Nevertheless, such documents are still laws, sharing all features of the national-level legislation, with respect to references and properties (e.g., type of law, publication date), and having specific features (e.g., region). Thus, the flexibility of the Property Graph Schema allows us to model such laws as an additional multi-labelled node, sharing the **Law** label with national nodes but also having the **RegionalLaw** label, which characterizes their nature.

Because each region follows different formats, handling all of them simultaneously would be highly complex and lead to inconsistencies in the resulting graph. To address this, we take an incremental approach: thanks to the flexibility of the property graph model, we start by incorporating regional data step by step. In this paper, we focused on refining the process for two regions, Lombardia and Liguria, and designed a general pipeline that will be implemented incrementally for all regions.

## 6.1 Lombardia

Regional laws from Lombardia are easily accessible thanks to a dedicated API provided by the regional government [31]. Each law is available in its most recent version, and, unfortunately, there is no consistent way to retrieve previous versions. The documents are provided in NIR-HTML format - which stands for *Norme in Rete* [7], an Italian standard used to structure legal documents.

References and citations to other laws can be extracted through specific HTML tags that contain the a unique code to the cited laws. To classify these relationships, we developed a set of heuristics that leverage the repetitive language commonly used in formal documents, allowing us to distinguish between abrogations, amendments, or general citations.

More than 2,500 law documents were processed and uploaded into the Italian Legislative Graph, including all laws promulgated by the regional government of Lombardia since 1971 when it started being active. For each law, a corresponding node was added, along with nodes for each of its articles, resulting in a total of 9,514 new **Article** nodes containing the text of the laws. In addition, over 9,000 edges were created. Of these, nearly 4,000 connect regional article nodes to articles in national laws, while the remaining edges represent intra-regional connections. Only a small fraction of these edges are labeled with something other than **CITES**, which may indicate the need for a more refined classification approach.

## 6.2 Liguria

Liguria's regional government does not provide an API to access its laws, but they are accessible through the official portal [29]. Each law is available in two formats: NIR-HTML, similar to the format used in Lombardia, and AKM-XML, also known as the Akoma Ntoso format. Akoma Ntoso (short for *Architecture for Knowledge-Oriented Management of African Normative Texts using Open Standards and Ontologies* [39]) is a widely adopted international standard for structuring legal documents in a machine-readable way.

However, NIR-HTML versions of Liguria's laws are only available for bulk download, which makes them impractical for building an end-to-end process that keeps the knowledge graph up to date as new laws are published. Instead, laws can be downloaded individually in the AKM-XML format, which requires a different process compared to Lombardia's laws. Even though the two process must be different, we aim to have a unified format for the processed data so that both regions can be uploaded and updated simultaneously daily.

Thanks to XML tags, we have direct references to citations to other laws. Using similar heuristics, we classified these references and label the corresponding edges in the knowledge graph.

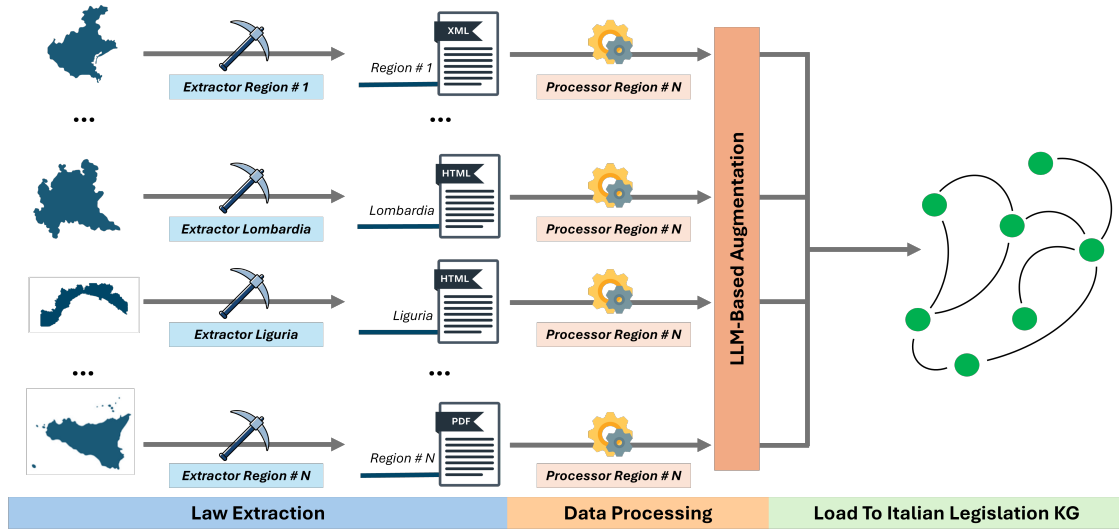
The regional government of Liguria has published approximately 2,000 laws since 1971, and for the purpose of this work, we processed and uploaded them in the Italian Legislative Graph. In addition, more than 14,000 **Article** nodes have also been uploaded, containing the text of all the laws they refer to.

## 6.3 Pipeline for Regional Integration

The successful integration of Lombardia and Liguria highlights how regional governments publish legislative data in different formats. To extend the Italian Legislative Graph to cover all 20 regions, we have designed a generalized pipeline that processes this heterogeneity while ensuring the consistency of the final knowledge graph. We aim to implement this proposed pipeline in future works.

The pipeline is depicted in Figure 12. It is structured into four distinct steps:

- **Step 1. Region Law Extraction.** For each region, we have designed a specific extractor to fetch the missing data. For a region that needs to be added to the Italian legislation graph, it downloads the full corpus of laws. Instead, it only retrieves the laws that have been published the day before for those regions already in the graph.



■ **Figure 12** Proposed Pipeline for integrating Regional Laws into the Italian Legislation Graph.

- **Step 2. Region Data Processing.** Once the laws have been extracted, we process them to have a unified data structure. A specific processor is designed for the specific file format of the region. First, it splits the text of the law into granular Article nodes. It then extracts the metadata of the law, such as the regional law univocal code and its publication date. Lastly, through tailored heuristics and tags, it looks for citations to national laws, or to laws of the same or other regions.
- **Step 3. LLM-based Augmentation.** During this step, an enhancement layer using Large Language Models is applied to the data. LLMs are used to refine citation extraction, identifying references that traditional heuristic techniques fail to capture. Furthermore, this step assigns semantic topics to laws with LLMs, enriching the graph's metadata for advanced data queries.
- **Step 4. Data Loading.** The final step maps the processed and augmented data to the unified Property Graph Schema presented. It loads the regional laws into the graph database, merging with existing national and regional layers.

**Edge Classification and Data Quality.** The extraction pipeline adopts a hybrid approach to ensure data quality and accurate edge classification across different institutional layers. For national legislation, edge types are derived from official annotations provided by the original sources, ensuring high reliability in the primary dataset. However, regional governments (e.g., Lombardia and Liguria) often use heterogeneous formats and lack these standardized official annotations. Therefore, we used a BERT-based model fine-tuned on national legislation data and available in [18]. The model was fine-tuned from Italian Legal BERT [28], achieving 92% accuracy on the test classification set for national edges (5k random national edges, 4 unbalanced classes). We applied this model to the regional layer, whose citation recitals are similar to those of the national layer, although we expect slightly lower performance.

## 6.4 Graph Queries for Regional Legislative Knowledge Management

The introduction of regional laws adds a new layer to the legislative system and opens up new opportunities for managing legal knowledge using property graphs. Having all the data in a single graph instance makes it easy for applications to focus on individual regions, compare regions, and interact with regional and national laws.

```

MATCH (:Law)-[:HAS_ARTICLE]->(art:Article)
WHERE ('regioni' in art.topics or 'lombardia' in art.topics) and
NOT (EXISTS((art)-[:CITES]-(:Article)-[:HAS_ARTICLE]-(law:RegionalLaw)
WHERE law.region = 'lombardia'))
or EXISTS((art)-[:IS_LEGAL_BASIS_OF]->(law:RegionalLaw)
WHERE law.region = 'lombardia'))
RETURN art

```

■ **Figure 13** Query to detect national laws citing regional laws.

■ **Table 3** Top three most recent results of the query in Figure 13.

| Article Title | Article ID                                                                                                        |
|---------------|-------------------------------------------------------------------------------------------------------------------|
| 2020 9#17     | Cassa integrazione in deroga per Lombardia, Veneto ed Emilia Romagna                                              |
| 2002 81#1     | Sospensione dei termini processuali e dei termini di prescrizione e decadenza per la Regione Lombardia            |
| 1998 6#22     | Ulteriori interventi urgenti nei territori della Lombardia interessati dagli eventi idrogeologici del giugno 1997 |

For instance, a regional lawmaker might want to identify national laws that explicitly mention regional institutions in general, or their specific region in particular. This can help highlight areas where the region may need to take action, either by intervening or by passing new laws that align with or respond to national legislation.

In the query presented in Figure 13, we use the `topics` property of national `Article` nodes, which captures the main subjects addressed by the legal text. This allows the query to find all national legal documents that refer to regions in general or to a specific region (such as Lombardia in our example, though any region could be used). The query then filters out the documents that have already been addressed by the region – either by being cited or serving as the legal basis for other regional laws – highlighting those that may still require attention from the lawmaker.

Table 3 displays the three most recent articles that find no citation in the regional legislative articles. In all cases, such laws do not require a specific reference in regional legislation (as they concern exclusive national competencies). However, the query suggests that the regional government is up to date with national legislation and faces no urgent legislative backlogs.

Another example, reported in Figure 14, analyses how laws from different regions are interconnected. This would show how several regions rely on similar legal frameworks to govern areas that have not yet been addressed at the national level.

The query presented counts how many regions have directly cited a regional law that does not have a legal basis in any national law. It then returns only those regional laws with a count above a certain threshold – 1 in this example – which suggests that more than half of the Italian regions have cited that law. This would help national lawmakers identify regional laws that have gained widespread attention and may indicate a shared need for regulation at the national level. This example illustrates how modeling the legislative system as a property graph enables a deeper understanding of cross-regional legal patterns. While each region develops its own legal network, analyzing the connections between them can support more informed and coordinated decision-making at the national level.

For instance, Table 4 showcases the laws from Lombardy, citing other regions' legislative framework. The laws cover a wide array of regulatory domains, specifically transport regulations, animal usage in research, and wood logistics.

```

MATCH (law:RegionalLaw)-[:HAS_ARTICLE]->(art:Article)
      <-[:CITES]-(art2:Article)<-[:HAS_ARTICLE]-(law2:RegionalLaw)
WHERE law2.region <> law.region
WITH law, art, count(DISTINCT law2.region) AS countRegion
WHERE NOT EXISTS((:Law)-[:HAS_ARTICLE]->(:Article)-[:IS_LEGAL_BASIS_OF]->(law)) and
      countRegion >= 1
RETURN law, art

```

■ **Figure 14** Query to detect regional laws that deserve the attention of national lawmakers.

■ **Table 4** Three laws from Lombardia that have been cited by other regions.

| Law ID  | Law Title                                                                                                                           |
|---------|-------------------------------------------------------------------------------------------------------------------------------------|
| 2012 06 | Disciplina del settore dei trasporti                                                                                                |
| 2015 04 | Norme in materia di protezione degli animali utilizzati ai fini di ricerca e sperimentazione e di promozione dei metodi alternativi |
| 2016 24 | Ratifica dell'accordo interregionale sul prelievo legnoso in ambito boschivo e sulla filiera legno                                  |

## 7 Conclusions and Perspectives

In this paper, we built upon recent efforts in modelling legislative systems within graph databases, describing the motivations and benefits of adopting Property Graphs as a powerful solution for comprehensive legislative knowledge management. We demonstrated how adopting a graph database and our schema can seamlessly handle the temporal aspects and evolution of legislative acts, delegating to queries the task of retrieving the correct information, such as computing the in-force text, instead of storing multiple versions of the same act.

Then, we looked at recent tools recently adopted and developed by the Property Graph community, i.e., triggers and association rules, to offer innovative and powerful applications that can be useful in conducting advanced legislative knowledge management. To this aim, we demonstrated how triggers can be used to build a simple but effective warning system that monitors whether legislative voids are created. Finally, we used association rules to discover significant patterns in the evolution of the Italian legislation referred to procurement, discovering an increasing trend of instability in such legislation. In future work, we will further explore the utility of these tools for knowledge discovery, refining their applications in the legislative domain with the help of experts and identifying new domains that could benefit from these advancements.

Finally, we began expanding the graph of Italian legislation to cover an additional layer of the legislative system. Specifically, we modified the graph schema to include laws issued by regional governments, and we integrated the regional laws of both Lombardia and Liguria, covering their legislative activity since 1971. This allowed us to incorporate historical laws and set up a system that keeps the graph up to date with new regional legislation. We then demonstrated how adding the regional layer enhances legislative knowledge management, using three Cypher query examples that explore connections within a single region, between different regions, and between regional and national laws.

Our work suggests that Property Graphs offer distinct advantages for legislative knowledge management compared to other approaches exploiting graphs, like RDF and Knowledge Graphs.

First, the ability to store rich properties directly on edges allows the model to naturally handle both the structural relationships and the document content, specifically enabling efficient law versioning without the complex logic or redundancy often required in RDF-based temporal modeling.

Second, the inherent flexibility of the Property Graph Schema facilitates the evolution of the data model, allowing for the integration of multi-level hierarchies, like the regional layer introduced in this study, which can be more challenging to expand in rigid structural formats.

Finally, while analytical techniques for RDF have existed for some time, this paper demonstrates how some recently developed tools designed for Property Graphs can effectively close this gap. Thanks to this, Property Graphs are a robust choice for modelling and handling legislative data.

Our main focus in the future will be to scale the integration pipeline to cover all twenty Italian regions. While the current implementation validates the approach for Lombardia and Liguria, expanding to the remaining regions requires tailoring extractors and processors for their specific publication formats. Furthermore, we will leverage the proposed LLM-based augmentation to enhance metadata and citation extraction, matching the quality of the data of national laws. With a fully populated graph, we also aim to design regional-level triggers and to extend association rule mining to better study interactions between national and regional layers.

Finally, we propose developing a user interface to make these graph-based tools accessible to non-technical stakeholders. Furthermore, we intend to integrate in the interface linguistic metrics, such as readability and syntactic complexity. Thanks to this, we aim to offer a tool designed to assess both the structural coherence and the textual quality of the legislation.

While this study focuses on the Italian legislative system, the underlying framework is designed for cross-jurisdictional portability.

First, since the model utilizes the universal OASIS Akoma Ntoso standard, it can be easily applied to other legislation. This has already been proven viable by applying the same schema to United States legislation [20] and to Switzerland legislation [15]. Then, the extractors and classifiers used to handle relationships would need to be adjusted on the specific legal constructs of the target jurisdiction.

## Statement on Supplementary Materials and Additional Resources

The Italian Legislation Graph dataset is stored at [14], and the supporting query files are provided in the GitHub repository at [9]. We used version 5.26 of Neo4j.

---

## References

- 1 Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216, 1993. doi:10.1145/170035.170072.
- 2 Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Alastair Green, Jan Hidders, Bei Li, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Stefan Plantikow, Ognjen Savkovic, Michael Schmidt, Juan Sequeda, Slawek Staworko, Dominik Tomaszuk, Hannes Voigt, Domagoj Vrgoc, Mingxi Wu, and Dusan Zivkovic. Pg-schema: Schemas for property graphs. *Proc. ACM Manag. Data*, 1(2), June 2023. doi:10.1145/3589778.
- 3 Tara Athan, Harold Boley, Guido Governatori, Monica Palmirani, Adrian Paschke, and Adam Wyner. Oasis legalruleml. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Law, ICAIL '13*, pages 3–12, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2514601.2514603.
- 4 Gioele Barabucci, Luca Cervone, Monica Palmirani, Silvio Peroni, and Fabio Vitali. Multi-layer markup and ontological structures in akoma ntoso. In *International Workshop on AI Approaches to the Complexity of Legal Systems*, pages 133–149, Beijing, China, 2009. Springer, Springer. doi:10.1007/978-3-642-16524-5\_9.
- 5 Luigi Bellomarini, Anna Bernasconi, Stefano Ceri, Alessia Gagliardi, Davide Magnanini, Davide Martinenghi, et al. Towards a standard for trig-



- gers in property graphs. In *CEUR WORKSHOP PROCEEDINGS*, volume 3741, pages 70–79. CEUR-WS, 2024. URL: <https://ceur-ws.org/Vol-3741/paper02.pdf>.
- 6 Anna Bernasconi, Francesco Invernici, Stefano Ceri, et al. TETYS: Towards the next-generation open-source web topic explorer. In *CEUR WORKSHOP PROCEEDINGS*, volume 3692, pages 26–33. CEUR-WS, 2024. URL: <https://ceur-ws.org/Vol-3692/paper4.pdf>.
  - 7 C Biagioli, E Francesconi, P Spinosa, and M Taddei. The NIR project: Standards and tools for legislative drafting and legal document web publication. In *Proceedings of ICAIL workshop on e-government: modelling norms and concepts as key issues*, pages 69–78, 2003.
  - 8 A. Bonifati, D. Braga, A. Campi, and S. Ceri. Active xquery. In *Proceedings 18th International Conference on Data Engineering*, pages 403–412, 2002. doi:10.1109/ICDE.2002.994754.
  - 9 Francesco Cambria. Cypher queries for legislative property graphs. swHId: swH1:dir:c175195a74440b097ec11f0fb49fdc67cf616930 (visited on 2026-08-12). URL: <https://github.com/FrancescoCambria/Cypher-Queries-TGDK26.git>.
  - 10 Francesco Cambria, Francesco Invernici, Anna Bernasconi, and Stefano Ceri. Mine graph rule: A new gql operator for mining association rules in property graph databases. *The VLDB Journal*, 34(4):1–21, 2025.
  - 11 Stefano Ceri, Anna Bernasconi, Alessia Gagliardi, Davide Martinenghi, Luigi Bellomarini, and Davide Magnanini. Pg-triggers: Triggers for property graphs. In *Companion of the 2024 International Conference on Management of Data, SIGMOD/PODS '24*, pages 373–385, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3626246.3653386.
  - 12 Stefano Ceri and Jennifer Widom. Deriving incremental production rules for deductive data. *Information Systems*, 19(6):467–490, 1994. doi:10.1016/0306-4379(94)90002-7.
  - 13 S Wesley Changchien and Tzu-Chuen Lu. Mining association rules procedure to support online recommendation by customers and products fragmentation. *Expert systems with applications*, 20(4):325–335, 2001. doi:10.1016/S0957-4174(01)00017-3.
  - 14 Andrea Colombo. Italian legislation graph. <https://zenodo.org/records/13798158>.
  - 15 Andrea Colombo. Switzerland legislation graph. <https://zenodo.org/records/17615989>.
  - 16 Andrea Colombo. Knowledge graph of the italian legislation, 2024. doi:10.5281/ZENODO.13798158.
  - 17 Andrea Colombo. Leveraging knowledge graphs and llms to support and monitor legislative systems. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management, CIKM '24*, pages 5443–5446, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3627673.3680268.
  - 18 Andrea Colombo. Bert-edges (revision 7fadf37), 2026. doi:10.57967/hf/8346.
  - 19 Andrea Colombo, Anna Bernasconi, and Stefano Ceri. An llm-assisted etl pipeline to build a high-quality knowledge graph of the italian legislation. *Information Processing and Management*, 62(4):104082, 2025. doi:10.1016/j.ipm.2025.104082.
  - 20 Andrea Colombo and Francesco Cambria. Llm-assisted construction of the united states legislative graph. *Proceedings of the VLDB Endowment*. ISSN: 2150:8097, 2025.
  - 21 Kenneth W Dam. *The law-growth nexus: The rule of law and economic development*. Rowman & Littlefield, 2007.
  - 22 Francesco Decarolis, Raymond Fisman, Paolo Pinotti, and Silvia Vannutelli. Rules, discretion, and corruption in procurement: Evidence from italian government contracting. Technical report, National Bureau of Economic Research, 2020.
  - 23 Bradley Efron. *Bootstrap methods: Another look at the Jackknife*, pages 569–593. Springer New York, New York, NY, 1992. doi:10.1007/978-1-4612-4380-9\_41.
  - 24 Kapali P. Eswaran. Aspects of a trigger subsystem in an integrated database system. In *Proceedings of the 2nd International Conference on Software Engineering, ICSE '76*, pages 243–250, Washington, DC, USA, 1976. IEEE Computer Society Press. URL: <http://dl.acm.org/citation.cfm?id=807683>.
  - 25 Amélie Gheerbrant, Leonid Libkin, Liat Peterfreund, and Alexandra Rogova. GQL and SQL/PGQ: Theoretical models and expressive power. *arXiv preprint arXiv:2409.01102*, 2024. doi:10.48550/arXiv.2409.01102.
  - 26 Rinke Hoekstra, Joost Breuker, Marcello Di Bello, Alexander Boer, et al. The LKIF core ontology of basic legal concepts. *CEUR Workshop Proceedings*, 321:43–63, 2007. 2nd Workshop on Legal Ontologies and Artificial Intelligence Techniques, LOAIT 2007 ; Conference date: 04-06-2007 Through 04-06-2007. URL: <https://ceur-ws.org/Vol-321/paper3.pdf>.
  - 27 Francesco Invernici and Francesco Cambria. Neo4j Mine Graph Rule Operator. [https://github.com/FrInve/mine\\_graph\\_rule](https://github.com/FrInve/mine_graph_rule), 2024.
  - 28 Daniele Licari and Giovanni Comandè. ITALIAN-LEGAL-BERT: A Pre-trained Transformer Language Model for Italian Law. In Danaï Symeonidou, Ran Yu, Davide Ceolin, María Poveda-Villalón, Davide Audrito, Luigi Di Caro, Francesca Grasso, Roberto Nai, Emilio Sulis, Fajar J. Ekaputra, Oliver Kutz, and Nicolas Troquard, editors, *Companion Proceedings of the 23rd International Conference on Knowledge Engineering and Knowledge Management*, volume 3256 of *CEUR Workshop Proceedings*, Bozen-Bolzano, Italy, September 2022. CEUR. ISSN: 1613-0073. URL: <https://ceur-ws.org/Vol-3256/#km41aw3>.
  - 29 Regione Liguria. Banca dati della regione liguria. [https://lrv.regione.liguria.it/liguriass\\_prod/](https://lrv.regione.liguria.it/liguriass_prod/).
  - 30 Wolf Linder. Swiss legislation in the era of globalisation: A quantitative assessment of federal legislation (1983-2007). *Swiss Political Science Review*, 20(2):223–231, 2014. doi:10.1111/spsr.12100.
  - 31 Regione Lombardia. CRL – Leggi regionali della lombardia. <https://www.dati.lombardia.it/government/>



- CRL-Leggi-Regionali-della-Lombardia/abjw-hhay/data\_preview.
- 32 C Lupo, Fabio Vitali, E Francesconi, Monica Palmirani, R Winkels, E de Maat, A Boer, P Mascellani, et al. General XML format (s) for legal sources. *Technical report, IST-2004-027655 ESTRELLA European project for Standardised Transparent Representations in order to Extend Legal Accessibility*, 2007.
  - 33 Davide Magnanini. Neo4j APOC light-weight reasoner. <https://github.com/davidemagnanini/apoc>, 2023.
  - 34 Reza Yousefi Maragheh, Chenhao Fang, Charan Chand Irugu, Parth Parikh, Jason Cho, Jianpeng Xu, Saranyan Sukumar, Malay Patel, Evren Korpeoglu, Sushant Kumar, et al. LLM-take: Theme-aware keyword extraction using large language models. In *2023 IEEE International Conference on Big Data (BigData)*, pages 4318–4324. IEEE, 2023.
  - 35 Jim Melton and Alan R Simon. *SQL: 1999: Understanding relational language components*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.
  - 36 OASIS. Akoma Ntoso Version 1.0 becomes an OASIS Standard, 2018. <https://www.oasis-open.org/news/announcements/akoma-ntoso-version-1-0-becomes-an-oasis-standard/>.
  - 37 Francisco de Oliveira and Jose Maria Parente de Oliveira. A RDF-based graph to representing and searching parts of legal documents. *Artificial Intelligence and Law*, 32(3):667–695, 2024. doi:10.1007/S10506-023-09364-9.
  - 38 Timur Osadchiy, Ivan Poliakov, Patrick Olivier, Maisie Rowland, and Emma Foster. Recommender system based on pairwise association rules. *Expert Systems with Applications*, 115:535–542, 2019. doi:10.1016/J.ESWA.2018.07.077.
  - 39 Monica Palmirani and Fabio Vitali. Akoma-ntoso for legal documents. *Legislative XML for the Semantic Web: Principles, Models, Standards for Document Management*, pages 75–100, 2011.
  - 40 Jeffrey L Rasmussen. Estimating correlation coefficients: Bootstrap and parametric approaches. *Psychological Bulletin*, 101(1):136, 1987.
  - 41 Stefan Schmitz, Markus Schluetter, and Ulrich Epple. Automation of automation — definition, components and challenges. In *2009 IEEE Conference on Emerging Technologies and Factory Automation*, pages 1–7, 2009. doi:10.1109/ETFA.2009.5347197.
  - 42 Huayou Si, Jiayong Zhou, Zhihui Chen, Jian Wan, Neal N Xiong, Wei Zhang, and Athanasios V Vasilakos. Association rules mining among interests and applications for users on social networks. *IEEE Access*, 7:116014–116026, 2019. doi:10.1109/ACCESS.2019.2925819.
  - 43 Francesco Sovrano, Monica Palmirani, Fabio Vitali, et al. Legal knowledge extraction for knowledge graph based question-answering. *Frontiers in Artificial Intelligence and Applications*, 334:143–153, 2020. doi:10.3233/FAIA200858.
  - 44 Ramakrishnan Srikant and Rakesh Agrawal. Mining generalized association rules. *Future generation computer systems*, 13(2-3):161–180, 1997. doi:10.1016/S0167-739X(97)00019-8.
  - 45 Meera Tandan, Yogesh Acharya, Suresh Pokharel, and Mohan Timilsina. Discovering symptom patterns of COVID-19 patients using association rule mining. *Computers in biology and medicine*, 131:104249, 2021. doi:10.1016/J.COMPBIOMED.2021.104249.
  - 46 Fabio Vitali, Monica Palmirani, et al. Akoma ntoso: Flexibility and customization to meet different legal traditions. *BALISAGE SERIES ON MARKUP TECHNOLOGIES*, 24:1–9, 2019.
  - 47 Jennifer Widom and Stefano Ceri. *Active database systems: Triggers and rules for advanced database processing*. Morgan Kaufmann, 1995.



# Towards a Theory of Façade-X Data Access: Satisfiability of SPARQL Basic Graph Patterns

Luigi Asprino   

Università Telematica San Raffaele, Rome, Italy

Enrico Daga<sup>1</sup>   

Knowledge Media Institute (KMI), The Open University, Milton Keynes, UK

## Abstract

Data integration is the primary use case for knowledge graphs. However, integrated data are not typically graphs but come in different formats, for example, CSV, XML, or a relational database. Façade-X is a recently proposed method for providing direct access to an open-ended set of data formats. The method includes a meta-model that specialises RDF to fit general data structures. This model allows to express SPARQL queries targeting data sources with those structures. Previous work formalised Façade-X and demonstrated how it can theoretically represent any format expressible with a context-free grammar, as well as the relational model. A reference implementation, SPARQL Anything, demonstrates the feasibility of the approach in practice.

It is noteworthy that Façade-X utilises a fraction of RDF, and, consequently, not all SPARQL queries yield a solution (i.e. are satisfiable) when evaluated over a Façade-X graph. In this article, we consolidate Façade-X and we study the *satisfiability* of basic graph patterns. The theory is accompanied by an algorithm for deciding the satisfiability of basic graph patterns on Façade-X data sources. Furthermore, we provide extensive experiments with a proof-of-concept implementation, demonstrating practical feasibility, including with real-world queries. Our results pave the way for studying query execution strategies for Façade-X data access with SPARQL and supporting developers to build more efficient data integration systems for knowledge graphs.

**2012 ACM Subject Classification** Theory of computation → Data structures design and analysis; Information systems → Graph-based database models; Information systems → Query languages for non-relational engines; Information systems → Mediators and data integration

**Keywords and phrases** Façade-X, RDF, SPARQL, Knowledge Graphs, Data Integration

**Digital Object Identifier** 10.4230/TGDK.4.2.5

**Category** Research

**Supplementary Material** *Software (Source Code)*: <https://zenodo.org/records/18196574> [15]

*Dataset (Data)*: <https://zenodo.org/records/18133036> [14]

**Funding** *Enrico Daga*: EU funded projects (a) MultiPod: grant agreement 101178821, <https://cordis.europa.eu/project/id/101178821>; and (b) Climate Misinformation Surveillance and Analysis with Multidimensional GIS, <https://climatesense-project.eu/>, CHIST-ERA project.

**Received** 2025-07-15 **Accepted** 2026-04-03 **Published** 2026-09-03

**Editors** Stefania Dumbrava, George Fletcher, Matteo Lissandrini, and Riccardo Tommasini

**Special Issue** Data Management for (Knowledge) Graphs

**Generative AI Declaration** Generative AI was not used in the preparation of this article.

## 1 Introduction

Knowledge graphs use a graph-based model to integrate, manage, and extract value from diverse data sources at scale [26]. Thus, data integration is the primary application of Knowledge Graphs (KG). The W3C standard SPARQL 1.1 [23] serves as the authoritative language for engaging

<sup>1</sup> Corresponding author



© Luigi Asprino and Enrico Daga;  
licensed under Creative Commons License CC-BY 4.0

Transactions on Graph Data and Knowledge, Vol. 4, Issue 2, Article No. 5, pp. 5:1–5:46



Transactions on Graph Data and Knowledge

TGDK Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 5:2 Towards a Theory of Façade-X Data Access

with knowledge graphs serialised in RDF [32]. SPARQL provides functionalities for selecting, filtering, and aggregating data into a table format. Additionally, SPARQL allows the projection of results into an RDF structure using the CONSTRUCT query type. Consequently, research has investigated the use of SPARQL across a variety of applications beyond mere querying, particularly in the integration of diverse data sources [13, 33, 36]. In practice, methods often depend on enhancing SPARQL to retrieve data from non-RDF formats.

Recent studies [16] suggest utilizing an intermediate RDF model called Façade-X, which allows its components to be seamlessly converted into RDF Knowledge Graphs. This approach enables the development of software that offers *direct access* to source data as RDF, freeing knowledge engineers from the burden of managing diverse formats and associated languages they depend on. Façade-X uniquely addresses *re-engineering*, which means transforming resources by minimizing domain-specific concerns and concentrating on the syntactical meta-model, and allows users to focus on *remodelling*, that is, transforming the original domain model into a new representation [16]. In theory, Façade-X can represent any format defined by a context-free grammar, as well as the relational data model [8].

The Façade-X method informs the SPARQL Anything software and open source project, which also serves as its reference implementation<sup>2</sup>.

```
email,name,surname
laura@example.com,Laura,Grey
craig@example.com,Craig,Johnson
mary@example.com,Mary,Jenkins
jamie@example.com,Jamie,Smith
```

■ **Figure 1** Example CSV file.

■ **Listing 1** A query for retrieving the surname of people whose first name is “Laura”.

```
PREFIX xyz: <http://sparql.xyz/facade-x/data/>

SELECT ?surname WHERE {
  SERVICE <x-sparql-anything:https://sparql-anything.cc/example1.csv> {
    _:person xyz:surname ?surname .
    _:person xyz:name "Laura" .
  }
}
```

► **Example 1.** Suppose having the CSV file shown in Figure 1 as input<sup>3</sup>. Using SPARQL Anything, a SPARQL user can retrieve the surnames of people whose first name is “Laura” by running the query in Listing 1 obtaining the following result:

| surname |
|---------|
| “Grey”  |

---

<sup>2</sup> <http://sparql-anything.cc>

<sup>3</sup> Also available at <https://sparql-anything.cc/example1.csv>

This system (accessible through a web server, CLI, Java, or Python API [40]) has been utilized in numerous practical applications and is gaining recognition from the knowledge graph community in both research and industry [7]<sup>4</sup>. The W3C Data Façades Community Group has been recently founded to develop Façade-X as a vendor-neutral, interoperable technology that can be implemented by various developers and organisations, including graph database vendors<sup>5</sup>.

Currently, the execution of SPARQL queries on Façade-X data sources requires the materialisation of the source (in full or via a slicing approach [6]). An open question is how to support the implementation of more efficient query engines. It is noteworthy that Façade-X utilises a fraction of RDF, and, consequently, not all SPARQL queries yield a solution (i.e. are satisfiable) when evaluated over a Façade-X graph. For example, the query described in the Example 1 is satisfiable, whereas the query in the Listing 2 is not. This is because the Façade-X model does not allow for such a pattern: CSV data sources don't have any `rdf:type` triples; therefore, no results would be produced from its evaluation. These considerations can be made simply by looking at the query itself (rather than the input) and could save computational resources.

■ **Listing 2** A query for retrieving the surname of people whose first name is “Laura”.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?s WHERE {
  SERVICE <x-sparql-anything:https://sparql-anything.cc/example1.csv> {
    ?s rdf:_1 ?o . ?x rdf:type ?s .
  }
}
```

In this article, we study the satisfiability of basic graph patterns under Façade-X. In addition, we contribute an algorithm to annotate any SPARQL BGP with possible solution patterns, to determine whether a given BGP can have a solution within Façade-X. We contribute:

- A consolidated formal definition of Façade-X and associated mappings to RDF;
- Characterisation of the satisfiability of SPARQL basic graph patterns queries within Façade-X;
- Algorithm for assessing whether a SPARQL basic graph patterns is satisfiable under Façade-X.

The rest of the article is structured as follows. The next section is dedicated to surveying related literature and contextualise our work. Section 3 introduces the formalisms and languages adopted in this article. Section 4 presents a consolidated version of Façade-X whose mapping to RDF is formalised in Section 5.1. Finally, Section 6 studies the conditions under which a BGP is satisfiable in Façade-X which are exploited in the algorithm presented in Section 7. Finally, Section 8 provides experiments with a proof-of-concept implementation, which demonstrates the practical feasibility and advantages of our theory, including an evaluation with real world queries. We conclude the article with Section 9.

## 2 Related work

We survey literature on Knowledge Graph Construction (KGC), i.e. systems and methods to onboard non-RDF data into RDF databases. We then look into approaches to extend SPARQL. Next, we focus on the evaluation of SPARQL queries, particularly basic graph patterns. Furthermore, we review the problem of satisfiability of graph patterns outside the RDF area. We conclude with a look at various approaches to query documents and provide a unifying view over heterogeneous resources.

<sup>4</sup> See also the activity on the open-source project page on GitHub: <http://github.com/sparql-anything/sparql.anything>

<sup>5</sup> W3C Data Façades Community Group: <https://www.w3.org/community/facade-x/>.

## 2.1 Knowledge graph construction

Mapping languages for transforming heterogeneous files into RDF are represented by RML [29, 47], also specialised to support data cleaning operations [44], and specific forms of data: relational [41], geospatial data [30], etc. RML has been adopted as reference mapping language in most of the KGC systems, such as SDM-RDFizer [27], Morph-CSV [11], and Morph-KGC [4]. Planning techniques have been studied to improve the efficiency of RML engines [28]. A recent work provides an algebraic foundation for RML based approaches [37], which establish a *query* operator as a primitive component in the definition of a mapping. Façade-X could be used as a common language to express queries over heterogeneous data sources *formally*.

Authors of an alternative to RML, based on the RDF constraining language ShEx [21], stress the importance of making mappings usable by end users. Indeed, literature acknowledges how these languages are built with machine-processability in mind [25], and how defining or even understanding the rules is not trivial. Façade-X provides a common abstraction allowing to express queries to non-RDF sources using basic graph patterns. It is worth noting that, although abstractions targeting individual data formats are already available in the literature, Façade-X keeps a single formalism and a single data model (i.e. RDF) for all data formats. Therefore, it paved the way for querying non-RDF data in SPARQL and building KGs, by making plain CONSTRUCT queries with a SPARQL engine supporting Façade-X data access. Moreover, since Façade-X is an abstract model that has been defined independently of RDF (as shown in Section 4, its specification is in predicate logic), it can potentially be applied to other languages (e.g. Datalog). Indeed, other query languages could benefit from the homogeneous overview of heterogeneous sources provided by Façade-X. Although this analysis could be extended to other frameworks for representing and querying data, our focus is on SPARQL. However, it is an open question *how to efficiently execute SPARQL queries on Façade-X data sources* that cannot be fully loaded in memory via a materialisation approach [6]. This problem motivates the present work.

## 2.2 Approaches to extend SPARQL

We examine methods for enhancing SPARQL. A common technique for augmenting SPARQL is to introduce custom functions that can be utilized in FILTER or BIND operators<sup>6</sup>. Query processing engines can enhance SPARQL through the use of so-called magic properties. This method involves defining custom predicates to guide specific actions during query execution<sup>7</sup>. A comparable method is employed to incorporate XPath and XQuery in Virtuoso, which also enables the embedding of SQL functions within SPARQL to directly access the underlying relational model<sup>8</sup>. SPARQL Generate [33] presents a technique for transforming data from varied sources into RDF by enhancing the SPARQL syntax with a new GENERATE operator [33]. This method introduces two additional operators, SOURCE and ITERATOR. Custom functions execute ad-hoc operations on supported formats, utilizing XPath or JSONPath. Nevertheless, there are methods to augment SPARQL without modifying the standard syntax. For instance, BASIL variables permit the formulation of parametric queries by imposing a convention on SPARQL variable names [17, 35]. SPARQL Anything leverages BASIL variables to facilitate parametric queries and file names. SPARQL Micro service [36] offers a framework that, based on an API mapping

<sup>6</sup> ARQ includes a library of custom functions that facilitate aggregates, such as calculating the standard deviation of a set of values. ARQ functions: <https://jena.apache.org/documentation/query/extension.html> (accessed 15/12/2020).

<sup>7</sup> For instance, this allows for the specification of intricate fulltext searches over literal values. Query processors can outsource execution to a fulltext engine (e.g., Lucene) and produce a set of query results as triple patterns

<sup>8</sup> Virtuoso Open Source. SPARQL inline in SQL: <http://docs.openlinksw.com/virtuoso/rdfsparqlinline/>.

specification, encapsulates web APIs as SPARQL endpoints and employs a JSON-LD profile to convert the API's JSON responses into RDF. Façade-X adopts a similar minimalist strategy and extends SPARQL by *overriding* the functionality of the SERVICE operator.

## 2.3 Evaluation of SPARQL graph patterns

Foundations of SPARQL query executions can be found in [26]. Seminal work in the area include studying the semantics of the query language [39, 3] and optimise the execution of graph pattern matching [46, 24, 19, 49]. Recent work tackles the efficient execution of SPARQL queries via translation in Datalog [2] or via combining logical and physical query plans [38].

The satisfiability problem is undecidable in SPARQL 1.0 (and, therefore, in 1.1 that is its extension) but becomes decidable for basic graph patterns [52]. Although decidability is NP-complete in theory, decidability of SPARQL BGPs is tractable in practice as the SPARQL BGPs are typically small [52]. Our work starts from the observation that Façade-X RDF graphs are less expressive than general RDF data. This motivates the study of evaluating basic graph patterns in the context of Façade-X graphs. However, it is certainly possible that optimization techniques for RDF pattern matching can be also applied to Façade-X. Hence, in the present work, we investigate the nature of Façade-X graphs and their corresponding basic graph patterns in SPARQL.

Another line of research analyses the satisfiability of BGPs in the context of conjunctive query (CQ) theory. Indeed, a SPARQL BGP can be viewed as a CQ over an RDF graph, thus inheriting the properties of CQs. A well-known result of Yannakakis [51] demonstrates that the evaluation of acyclic CQs is solvable in polynomial time. This result shows that the shape of a query fundamentally impacts the complexity of its evaluation. This perspective is relevant to Façade-X. Since the Façade-X data model only allows for rooted, acyclic structures, thus enforcing the tree-like nature of satisfiable BGPs.

## 2.4 Satisfiability of patterns in graphs

The problem of efficiently performing graph matching operations is significant beyond SPARQL [20, 34, 50, 9, 18]. While the general problem of subgraph pattern matching is known to be NP-complete [9], it seems that for a wide variety of statistics concerning trees, the expected time complexity for pattern matching is linear [45]. The satisfiability problem of tree patterns queries (within XPath and XQuery) is investigated in [31], where it is observed that some of the queries are NP-complete, while others can be resolved in polynomial time.

While Façade-X graphs can be seen as trees, when referring to static data sources, such as JSON documents or XML files (where two equal strings have different identity, because it derives from their position), their representation in RDF are not, because identical types and literals will be projected to unique RDF nodes. Instead, Façade-X/RDF graphs are *directed, acyclic, and rooted path graphs*. In the present work, we focus on satisfiability of SPARQL basic graph patterns on Façade-X RDF sources, and contribute a theory and related algorithm for deciding whether a graph pattern can have a solution in a possible Façade-X RDF graph.

## 2.5 Other approaches

Tools are available for automatically transforming data sources of several formats into RDF (Any23<sup>9</sup>, JSON2RDF<sup>10</sup>, CSV2RDF<sup>11</sup> to name a few). While these tools have a similar goal (i.e. enabling the user to access the content of a data source as if it was in RDF), the (meta)model used

<sup>9</sup> <http://any23.apache.org/> (now retired).

<sup>10</sup> <https://github.com/AtomGraph/JSON2RDF>

<sup>11</sup> <http://clarkparsia.github.io/csv2rdf/>



for generating the RDF data highly depends on the input format, thus limiting the homogeneity of data generated from heterogeneous data formats. In addition, none of those approaches are based on a common abstraction from heterogeneous formats.

A way to provide generalised data access is the list-of-lists approach. This is used by software libraries to onboard heterogeneous data structures, see, for example, Pandas<sup>12</sup> or the Visidata tool<sup>13</sup>. Façade-X follows a similar intuition, while targeting *graphs* as the general meta-model, rather than tabular representations.

Façade-X use cases typically involve documents. The approaches to querying documents have a tradition in computer science [12, 1, 48], and many format-specific languages have been developed for that (XPath, JsonPath, and CSS selectors are the most popular). Façade-X can be seen as a general approach to express queries over heterogeneous files. Here, we focus on the satisfiability of basic graph patterns against Façade-X RDF graphs. This analysis is the foundation for studying efficient execution strategies of SPARQL / Façade-X queries against serialized resources such as JSON or XML files.

### 3 Preliminaries

We introduce the reader to the formalisms on which our theory is based, namely RDF, SPARQL and Predicate Logic.

#### 3.1 Predicate logic

As the façade-x model is defined in predicate logic, we provide the reader with a brief introduction to the main ingredients of this formalism. However, providing a full description of the syntax and semantic of PL is not the purpose of this section. For a complete definition of the syntax and semantic of PL we defer the reader to reference books, such as [42]. Predicate Logic (PL) is the logic to speak about objects, which are the domain of discourse (universe  $\mathcal{U}$ ). PL is concerned about properties and relations about the objects of the universe which are expressed in form of predicates. Intuitively, a PL formula is combination of PL terms (variables and constants) with logical operators  $\neg, \wedge, \vee, \Rightarrow$  and quantifiers  $\exists, \forall$ . An interpretation  $\mathcal{J}$  of one or multiple PL formulas specifies what each predicate means, and the entities that can instantiate the variables. Given a set of predicates  $P_1 \dots P_n$  and function  $f_1 \dots f_m$  a PL interpretation is

$$\mathcal{J} = (\Delta^{\mathcal{J}}, P_1^{\mathcal{J}} \dots P_n^{\mathcal{J}}, f_1^{\mathcal{J}} \dots f_m^{\mathcal{J}})$$

where  $\Delta^{\mathcal{J}} \subseteq \mathcal{U}$  is the domain,  $P_i^{\mathcal{J}} \subseteq \Delta^{\mathcal{J}} \times \dots \times \Delta^{\mathcal{J}}$  (k-times where k is the arity of  $P_i$ ) is the interpretation of the predicate  $P_i$ ,  $f_i^{\mathcal{J}} : \Delta^{\mathcal{J}} \times \dots \times \Delta^{\mathcal{J}} \rightarrow \Delta^{\mathcal{J}}$  (k-times where k is the arity of  $f_i$ ) is the interpretation of the function  $f_i$ .

A PL open formula, also called *query*, is a PL formula with free variables (i.e. not quantified). Let **Vars** be the set of variables, given an interpretation  $\mathcal{J}$ , an assignment is a function  $\alpha : \mathbf{Vars} \rightarrow \Delta^{\mathcal{J}}$ .

Given a formula  $\phi$  with free variables  $(x_1 \dots x_n)$  and an interpretation  $\mathcal{J}$ , the answer to the query is the tuple  $(a_1 \dots a_n)$ , where  $a_i$  is an assignment of a free variable to an object of the domain of discourse  $\Delta^{\mathcal{J}}$ , such that  $\mathcal{J}, (a_1 \dots a_n) \models \phi$ . We introduce  $\mathcal{L}$  as the set of all possible queries, i.e. the query language, and the function **eval** which given a query  $\phi$  and an interpretation  $\mathcal{J}$  returns the list of assignments that satisfy  $\phi$  over  $\mathcal{J}$ .

<sup>12</sup><https://hevodata.com/learn/pandas-load-json/>

<sup>13</sup><https://www.visidata.org/>

### 3.2 RDF and SPARQL

In this section, we use relational algebra notation to define the elements of the universe under consideration, indicated by  $\mathcal{U}$ , namely RDF graphs, SPARQL queries, and resource, data sources and values they contain that will be introduced in Section 6. We also use predicate logic to define the conditions that apply to the elements of the universe.

We start from the basic notions of RDF [32] and SPARQL [23]. We defer the reader to [26] and to the corresponding documentation for a complete description of these standards. Formally, let  $\mathcal{I} \subseteq \mathcal{U}$ ,  $\mathcal{B} \subseteq \mathcal{U}$ , and  $\mathcal{L} \subseteq \mathcal{U}$  be infinite sets of IRIs, blank nodes, and literals. The sets are assumed to be pairwise disjoint and we will collectively refer to them as *RDF terms*. A tuple  $(s, p, o) \in \mathcal{T}$  (with  $\mathcal{T}$  is the set  $(\mathcal{I} \cup \mathcal{B}) \times (\mathcal{I}) \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$ ) is called (*RDF*) *triple* and we say  $s$  is the *subject* of the triple,  $p$  the *predicate*, and  $o$  the *object*. An *RDF graph* is a set of RDF triples, whereas an *RDF dataset* is a collection of named RDF graphs, each one identified by IRI, and a default RDF graph. *RDF graphs* can be expressed as a set of quads, i.e. elements of the set  $\mathcal{I} \times \mathcal{T}$ .

SPARQL is based on the idea of defining patterns to be matched against an input RDF dataset. Formally, considering the set of variables  $\mathcal{V} \subseteq \mathcal{U}$ , disjoint from the previously defined  $\mathcal{I}$ ,  $\mathcal{B}$ , and  $\mathcal{L}$ , a *triple pattern* is a tuple of the form  $(s, p, o) \in (\mathcal{I} \cup \mathcal{B} \cup \mathcal{V}) \times (\mathcal{I} \cup \mathcal{V}) \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V})$ <sup>14</sup>. All the possible SPARQL triple patterns are enumerated in Table 1. A *basic graph pattern (BGP)* is a set of triple patterns.

A SPARQL query  $Q$  is composed of the following components: **(i)** the query type (i.e. SELECT, ASK, DESCRIBE, CONSTRUCT); **(ii)** the dataset clause; **(iii)** the graph pattern (recursively defined as being a BGP or the result of the composition of one or more graph patterns through one of several SPARQL operators that modify and combine the obtained results); **(iv)** the solution modifiers (i.e. LIMIT, GROUP BY, OFFSET).

This work focuses solely on the graph pattern itself, since the other components can be evaluated as defined in SPARQL 1.1. Consider a graph pattern consisting of two joined unordered triple patterns with a join condition defined on a single node. There are 6 possible joins matching nodes of the two triple patterns. The joins are listed in Table 2.

More complex join conditions involving multiple nodes can be derived by combining these joins. Most common graph structures can be classified according to a set of graph *motifs* [5, 43]. These are reported in Table 3. For brevity, the motifs only contain nodes as variables, but these can be any valid combination of node types.

## 4 A consolidated version of Façade-X

Façade-X is the meta-model that results from the *abstraction* of all the basic data structures used to represent the source data formats and the *combination* of these data structures into a unified model. In this section, we present a consolidated version of Façade-X.

### 4.1 Resources and data sources

Before detailing the Façade-X, it is useful to formalise elements of its application context, i.e. the fact that data is serviced and shaped within a digital artifact of sort. We rely on the generic notion of *resource* to refer to a digital artifact that wraps some data<sup>15</sup>. In the Example 1, the Resource is the file identified by the URL <https://sparql-anything.cc/examples/simple.csv>, while the

<sup>14</sup>We are aware that the BGP specification in [23] includes literals in subject position. However, such patterns won't have a solution in RDF graphs, as defined in [32].

<sup>15</sup>To not be confused with the concept of Resource in RDFS (that is the most general type of RDF term) [22]

■ **Table 1** The list of all possible SPARQL triple patterns.

| Id    | Pattern     | Example                                      | Id     | Pattern     | Example                                                     |
|-------|-------------|----------------------------------------------|--------|-------------|-------------------------------------------------------------|
| $VVV$ | $V \ V \ V$ | $?s \ ?p \ ?o$                               | $ITV$  | $I \ I \ V$ | $\langle s \rangle \ \langle p \rangle \ ?o$                |
| $VVI$ | $V \ V \ I$ | $?s \ ?p \ \langle o \rangle$                | $III$  | $I \ I \ I$ | $\langle s \rangle \ \langle p \rangle \ \langle o \rangle$ |
| $VVB$ | $V \ V \ B$ | $?s \ ?p \ _:o$                              | $IIB$  | $I \ I \ B$ | $\langle s \rangle \ \langle p \rangle \ _:o$               |
| $VVL$ | $V \ V \ L$ | $?s \ ?p \ "o"$                              | $IIL$  | $I \ I \ L$ | $\langle s \rangle \ \langle p \rangle \ "o"$               |
| $IVV$ | $V \ I \ V$ | $?s \ \langle p \rangle \ ?o$                | $BVV$  | $B \ V \ V$ | $_:s \ ?p \ ?o$                                             |
| $VII$ | $V \ I \ I$ | $?s \ \langle p \rangle \ \langle o \rangle$ | $BVI$  | $B \ V \ I$ | $_:s \ ?p \ \langle o \rangle$                              |
| $VIB$ | $V \ I \ B$ | $?s \ \langle p \rangle \ _:o$               | $BVB$  | $B \ V \ B$ | $_:s \ ?p \ _:o$                                            |
| $VIL$ | $V \ I \ L$ | $?s \ \langle p \rangle \ "o"$               | $BVL$  | $B \ V \ L$ | $_:s \ ?p \ "o"$                                            |
| $IVV$ | $I \ V \ V$ | $\langle s \rangle \ ?p \ ?o$                | $BITV$ | $B \ I \ V$ | $_:s \ \langle p \rangle \ ?o$                              |
| $IVI$ | $I \ V \ I$ | $\langle s \rangle \ ?p \ \langle o \rangle$ | $BII$  | $B \ I \ I$ | $_:s \ \langle p \rangle \ \langle o \rangle$               |
| $IVB$ | $I \ V \ B$ | $\langle s \rangle \ ?p \ _:o$               | $BITB$ | $B \ I \ B$ | $_:s \ \langle p \rangle \ _:o$                             |
| $IVL$ | $I \ V \ L$ | $\langle s \rangle \ ?p \ "o"$               | $BITL$ | $B \ I \ L$ | $_:s \ \langle p \rangle \ "o"$                             |

■ **Table 2** The list of all possible single-node joins involving two SPARQL triple patterns.

| Id            | Join              | Example                               |
|---------------|-------------------|---------------------------------------|
| $S \bowtie P$ | $subj_1 = pred_2$ | $?j \ ?p1 \ ?o1 \ . \ ?s2 \ ?j \ ?o2$ |
| $P \bowtie O$ | $pred_1 = obj_2$  | $?s1 \ ?j \ ?o1 \ . \ ?s2 \ ?p1 \ ?j$ |
| $S \bowtie S$ | $subj_1 = subj_2$ | $?j \ ?p1 \ ?o1 \ . \ ?j \ ?p1 \ ?o2$ |
| $P \bowtie P$ | $pred_1 = pred_2$ | $?s1 \ ?j \ ?o1 \ . \ ?s2 \ ?j \ ?o2$ |
| $S \bowtie O$ | $subj_1 = obj_2$  | $?j \ ?p1 \ ?o1 \ . \ ?s2 \ ?p1 \ ?j$ |
| $O \bowtie O$ | $obj_1 = obj_2$   | $?s1 \ ?p1 \ ?j \ . \ ?s2 \ ?p1 \ ?j$ |

content of the CSV file is the DataSource. We say that a resource contains one or more *data sources*, to refer to actual data contained in them. Intuitively, a CSV is a file (then, a resource) that includes some tabular data (hence, a single data source). A Spreadsheet is a file (then, a resource) containing many tabs, each one of them including tabular data (hence, possibly multiple data sources).

We introduce the unary predicates `Resource` and `DataSource` and the binary predicate `includes`. These predicates must comply with the following conditions: **(i)** Resources includes at least one data source; **(ii)** A data source must be included in one and only one resource. See Listing 3.

■ **Listing 3** Formal definition of resources and data sources.

```

1  $\forall r, ds. includes(r, ds) \Rightarrow Resource(r) \wedge DataSource(ds)$ 
2  $\forall r. \exists ds. Resource(r) \Rightarrow includes(r, ds)$ 
3  $\forall ds. \exists r. DataSource(ds) \Rightarrow includes(r, ds)$ 
4  $\forall ds, r_1, r_2. includes(r_1, ds) \wedge includes(r_2, ds) \Rightarrow r_1 = r_2$ 

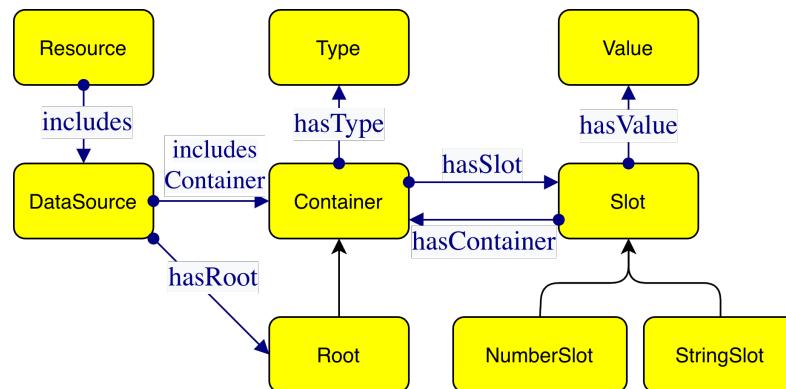
```

■ **Table 3** A list of possible SPARQL BGP motifs.

| BGP Motif Name | Path                     | Cycle/Clique            | Star                                        | Snowflake                                                                   |
|----------------|--------------------------|-------------------------|---------------------------------------------|-----------------------------------------------------------------------------|
| Motif          | ?s ?p ?o .<br>?o ?p1 ?o1 | ?s ?p ?o .<br>?o ?p1 ?s | ?s ?p ?o .<br>?s ?p1 ?o1<br>. ?s ?p2<br>?o2 | ?s ?p ?o .<br>?s ?p1 ?o1<br>. ?s ?p2<br>?o2 . ?o<br>?p3 ?o3 .<br>?o ?p4 ?o4 |

## 4.2 Façade-X

We overview the basic data structures needed for representing the content of common resources including CSV, JSON, YAML, XML, HTML, Text, Markdown, Relational Databases, and Spreadsheets (e.g. XLS and XLSx)<sup>16</sup>. The framework proposed in [8] represents any data source as a combination of only three knowledge representation primitives, namely: sets of key-data pairs (*maps*), ordered sequence of elements (*lists*), and unary predicates (*types*). In fact, this framework interprets any data source as a sequence of values, regardless of its format, and then, arranges the values of the sequence in multiple structures according to their serialisation format. Crucially, all these structures can be captured using the three knowledge representation primitives mentioned above. In addition, the framework introduces the notion of *Container* as an abstraction of both lists and maps through the notion of *Slots*. A *Slot* is an allotted place for an object, namely a primitive value or another container, “contained” in a *Container*. *Container* may have a type. Façade-X is a representation of a data source in terms of Containers, Types, Slots, and Values. In this Section we provide a formal definition of the Façade-X. Figure 2 shows a diagrammatic representation of the Façade-X that provides an informal overview of the model.



■ **Figure 2** A diagrammatic representation of the Façade-X model. Unlabelled arcs represent subsumption (e.g. Root is a type of Container).

We introduce the unary predicates: `Container`, `Slot`, `StringSlot`, `NumberSlot` and `Value`. We also introduce the binary predicates `hasSlot`, `hasValue`, `includesContainer` and `hasContainer`. Containers are collections of unique key/value pairs (slots) where: the key of the slot is unique in the collection and can be either a number (i.e. `NumberSlot`) or a sequence of alphanumeric characters (i.e. `StringSlot`); the value can be either a primitive value or another container. See Listing 4.

<sup>16</sup>See also <https://sparql-anything.readthedocs.io/stable/#supported-formats>.

■ **Listing 4** Formal definition of containers, slots, and values.

```

1  $\forall c, s. \text{hasSlot}(c, s) \Rightarrow \text{Container}(c) \wedge \text{Slot}(s)$ 
2  $\forall s, v. \text{hasValue}(s, v) \Rightarrow \text{Slot}(s) \wedge \text{Value}(v)$ 
3  $\forall s, c. \text{hasContainer}(s, c) \Rightarrow \text{Slot}(s) \wedge \text{Container}(c)$ 
4  $\forall s. \text{NumberSlot}(s) \Rightarrow \text{Slot}(s)$ 
5  $\forall s. \text{StringSlot}(s) \Rightarrow \text{Slot}(s)$ 
6  $\nexists s. \text{StringSlot}(s) \wedge \text{NumberSlot}(s)$ 
7  $\forall s. \text{Slot}(s) \Rightarrow \text{NumberSlot}(s) \vee \text{StringSlot}(s)$ 
8  $\forall ds, c. \text{includesContainer}(ds, c) \Rightarrow \text{DataSource}(ds) \wedge \text{Container}(c)$ 
9  $\forall c. \exists ds. \text{Container}(c) \Rightarrow \text{includesContainer}(ds, c)$ 
10  $\forall ds_1, ds_2, c. \text{includesContainer}(ds_1, c) \wedge \text{includesContainer}(ds_2, c) \Rightarrow ds_1 = ds_2$ 

```

■ **Listing 5** Formal definition of the properties of slots.

```

1  $\forall v \exists s. \text{Value}(v) \Rightarrow \text{hasValue}(s, v)$ 
2  $\forall s \exists c. \text{Slot}(s) \Rightarrow \text{hasSlot}(c, s)$ 
3  $\forall c_1, c_2, s. \text{hasSlot}(c_1, s) \wedge \text{hasSlot}(c_2, s) \Rightarrow c_1 = c_2$ 
4  $\forall s \exists x. \text{Slot}(s) \Rightarrow \text{hasContainer}(s, x) \vee \text{hasValue}(s, x)$ 
5  $\nexists s, x. \text{hasContainer}(s, x) \wedge \text{hasValue}(s, x)$ 
6  $\forall c_1, c_2, s. \text{hasContainer}(s, c_1) \wedge \text{hasContainer}(s, c_2) \Rightarrow c_1 = c_2$ 
7  $\forall v_1, v_2, s. \text{hasValue}(s, v_1) \wedge \text{hasValue}(s, v_2) \Rightarrow v_1 = v_2$ 
8  $\nexists c, s. \text{hasSlot}(c, s) \wedge \text{hasContainer}(s, c)$ 
9  $\forall c \nexists s_1 \dots s_n, c_1 \dots c_n. \text{hasSlot}(c, s_1) \wedge \text{hasContainer}(s_1, c_1) \wedge \dots$ 
    $\dots \wedge \text{hasSlot}(c_{n-1}, s_n) \wedge \text{hasContainer}(s_n, c)$ 
10  $\forall c, s_1, s_2. \text{hasSlot}(s_1, c) \wedge \text{hasSlot}(s_2, c) \Rightarrow s_1 = s_2$ 

```

**Terminology.** We say that a container  $c$  *contains* a slot  $s$  if  $(c, s)$  is in the interpretation of the predicate `hasSlot`. Moreover, we say that a slot  $s$  *holds* a value  $v$  (or a container  $c$ ) if  $(s, v)$  (resp.  $(s, c)$ ) is in the interpretation of `hasValue` (resp. `hasContainer`). In this case we can also say that  $v$  (resp.  $c$ ) is assigned to a slot. Finally, we say that a container  $c$  *recursively contains* a container  $c'$  if there exists a sequence  $s_0, \dots, s_n$  and containers  $c_1, \dots, c_n$  such that  $(c, s_0), (c_1, s_1) \dots (c_n, s_n)$  is in the interpretation of `hasSlot` and  $(s_0, c_1), (s_1, c_2), \dots, (s_n, c')$  is in the interpretation of `hasContainer`.

A formal definition of the properties of slots is provided in Listing 5. Values must be assigned to a slot (cf. 5.1). Slots must be contained by a single container (cf. 5.2, 5.3). Slots must hold either a single container or a single value (cf. 5.4, 5.5, 5.7, 5.6). There cannot exist a container  $c$  and a slot  $s$  such that  $c$  contains  $s$  and  $s$  holds  $c$ . (cf. 5.8). Each container cannot recursively contain itself through a sequence of slots and containers (cf. 5.9). Each container can only be held by a single slot (it is not possible for two slots to hold the same container)(cf. 5.10).

Containers can have types (cf. 6.1). Every type must be assigned to a container (cf. 6.2). To this end we introduce the unary predicate `Type` and the binary predicate `hasType`. A formal definition of types is provided in Listing 6.

■ **Listing 6** Formal definition of types.

```
1  $\forall c, t. \text{hasType}(c, t) \Rightarrow \text{Container}(c) \wedge \text{Type}(t)$ 
2  $\forall t. \exists c. \text{Type}(t) \Rightarrow \text{hasType}(c, t)$ 
```

The unary predicates `Container`, `Slot`, `Value`, and `Type` are pairwise disjoint. Disjointness axioms are provided in Listing 7.

■ **Listing 7** Definition of the disjointness relation.

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>1 <math>\forall x. \text{Container}(x) \Rightarrow \neg \text{Slot}(x)</math> 2 <math>\forall x. \text{Container}(x) \Rightarrow \neg \text{Value}(x)</math> 3 <math>\forall x. \text{Container}(x) \Rightarrow \neg \text{Type}(x)</math> 4 <math>\forall x. \text{Slot}(x) \Rightarrow \neg \text{Container}(x)</math> 5 <math>\forall x. \text{Slot}(x) \Rightarrow \neg \text{Value}(x)</math> 6 <math>\forall x. \text{Slot}(x) \Rightarrow \neg \text{Type}(x)</math></pre> | <pre>7 <math>\forall x. \text{Value}(x) \Rightarrow \neg \text{Container}(x)</math> 8 <math>\forall x. \text{Value}(x) \Rightarrow \neg \text{Slot}(x)</math> 9 <math>\forall x. \text{Value}(x) \Rightarrow \neg \text{Type}(x)</math> 10 <math>\forall x. \text{Type}(x) \Rightarrow \neg \text{Container}(x)</math> 11 <math>\forall x. \text{Type}(x) \Rightarrow \neg \text{Slot}(x)</math> 12 <math>\forall x. \text{Type}(x) \Rightarrow \neg \text{Value}(x)</math></pre> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

For each data source, there exists one and only one Root container (cf. 8.1, 8.7, 8.3). The Root container cannot be assigned to a slot (cf. 8.6). Containers that are not root must be assigned to a slot (cf. 8.8). To identify the root container we introduce the unary predicate `Root` as specialisation of `Container` (8.5) and the binary predicate `hasRoot` that connects a data source to its root container (cf. 8.2, 8.4). A formal definition of the Root container is provided in Listing 8.

■ **Listing 8** Formal definition of Root container.

```
1  $\forall ds. \exists r. \text{DataSource}(ds) \Rightarrow \text{hasRoot}(ds, r)$ 
2  $\forall ds, r. \text{hasRoot}(ds, r) \Rightarrow \text{DataSource}(ds) \wedge \text{Root}(r)$ 
3  $\forall r. \exists ds. \text{Root}(r) \Rightarrow \text{hasRoot}(ds, r)$ 
4  $\forall ds, r. \text{hasRoot}(ds, r) \Rightarrow \text{includesContainer}(ds, r)$ 
5  $\forall x. \text{Root}(x) \Rightarrow \text{Container}(x)$ 
6  $\nexists r, s. \text{Root}(r) \wedge \text{hasContainer}(s, r)$ 
7  $\forall ds, r_1, r_2. \text{hasRoot}(ds, r_1) \wedge \text{hasRoot}(ds, r_2) \Rightarrow r_1 = r_2$ 
8  $\forall c \exists s. \text{Container}(c) \wedge \neg \text{Root}(c) \Rightarrow \text{hasContainer}(s, c)$ 
```

In Example 1, a container is generated for each row of the file. These containers are held by slots in a root container, which represents the entire table. A value is generated for each cell and associated with the corresponding row via a slot. An excerpt of this is shown in the Example 4.

### 4.3 Properties of the Façade-X

This section outlines some properties of the Façade-X model that will be used later to characterise the RDF knowledge graph constructed from the instances of the model.

► **Proposition 1** (Connectedness of containers). *Every non-root container is recursively contained by the root container.*

**Proof.** For each non-root container  $c$ , either (i)  $c$  is not assigned to any slot; (ii) or  $c$  is assigned to a slot. We can exclude (i) as by definition (8.8<sup>17</sup>) a non-root container must be assigned to a slot. As for (ii), we show that for each non-root container  $c$  there always exist a sequence of slots  $s_r, s_1, \dots, s_n$  and containers  $r, c_1, \dots, c_n$  such that  $(r, s_r), (c_1, s_1) \dots (c_n, s_n)$  is in the interpretation of **hasSlot** and  $(s_r, c_1), (s_1, c_2), \dots, (s_n, c)$  is in the interpretation of **hasContainer**, i.e.  $r$  recursively contains  $c$ . The following conditions imply that each non-root container  $c$  must be in a sequence as above: (a) each non-root container must be assigned to a slot (8.8); (b) each slot is always contained by a container (5.2); (c) containers cannot be recursively contain themselves (5.9). It is worth noting that (a) and (b) imply that for each non-root there is always a sequence of containers like that above which it belongs to. (c) implies that the sequence does not form a cycle. Therefore, the first container in the sequence must be the root container, since it cannot be assigned to any slot. ◀

► **Proposition 2.** *For each data source  $ds$ , there is one and only one sequence of containers and slots that connects the root of  $ds$  to its containers.*

**Proof.** This follows from the following facts: (i) There is always a single root for each data source (8.7); (ii) Every non-root container is recursively contained by the root container (1); (iii) Each slot is assigned to one and only one container (5.3); (iv) Each non-root container is assigned to one and only one slot (5.6). Therefore, (ii) implies that there is always a sequence of slots and containers that connects a container to a root, (ii) that there is always a single root, (ii) and (iv) guarantee that the sequence is one and only one (it is convenient to pass through the sequence backward, the last container of the sequence can be assigned to one and only one slot, which in turn can be assigned to one and only container and so on). ◀

Now that we have described the Façade-X model and demonstrated its properties, we introduce a function  $fx$ , called *façadify*. This function is required to convert a resource into an instance of the Façade-X model. Let  $\mathcal{I}$  be the set all possible interpretations and  $\mathcal{R}$  the set of all possible resources, the function  $fx$ , called *façadify*, given a resource  $r$  returns an interpretation  $\mathcal{I}$  that complies with the Façade-X model, i.e.  $fx : \mathcal{R} \rightarrow \mathcal{I}$ .

Summary of changes from [8]:

- We remove the predicates **Key(k)**, **hasKey(s,k)**, **StringKey(k)**, and **NumberKey(k)**, i.e. collapse them with the definition of slots. We add **StringSlot** and **NumberSlot** as a replacement. Without loss of generality, we imagine that the key of the slot is included in the identifier of the individual.
- We rename **Class** into **Type**.
- We solve an ambiguity regarding the distinction between data sources and root containers, that are now distinct. We add the predicate **hasRoot(ds, r)**.
- We remove **Name** and **hasName** that were used to assign a name to a **DataSource** (we assume that the identifier of the data sources are unique).
- We clarify disjointness and connectedness.
- We provide formal definitions for the predicates **DataSource**, **Resource** and for integrity constraints that were previously only defined informally.

► **Example 2.** Table 4 show an example of usage of the introduced predicates for interpreting a CSV file.

---

<sup>17</sup>With this notation we refer to the line 8 of the definition 8.



■ **Table 4** An example of usage of the introduced predicates for interpreting a CSV file<sup>18</sup>.

| Source: ex.csv | Façade-X Interpretation                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                | <pre>includes(file:///ex.csv,ex.csv), hasRoot(ex.csv,root),hasSlot(root,rs.1), hasSlot(root,rs.2),hasSlot(root,rs.3), NumberSlot(rs.1),NumberSlot(rs.2),NumberSlot(rs.3), hasContainer(rs.1,row:1),hasContainer(rs.2,row.2) name,surname Laura,Grey Craig,Johnson hasContainer(rs.3,row.3), hasSlot(row.1,rsr:11),hasSlot(row.1,rsr.12), hasSlot(row.2,rsr.21),hasSlot(row.2,rsr.22), hasSlot(row.3,rsr.31),hasSlot(row.3,rsr.32), hasValue(rsr.11,"name"),hasValue(rsr.12,"surname"), hasValue(rsr.21,"Laura"),hasValue(rsr.22,"Grey"), hasValue(rsr.31,"Craig"),hasValue(rsr.32,"Jonhnsn") NumberSlot(rsr.11),NumberSlot(rsr.12),NumberSlot(rsr.21), NumberSlot(rsr.22),NumberSlot(rsr.31),NumberSlot(rsr.32)</pre> |

► **Example 3.** Table 5 show an example of usage of the introduced predicates for interpreting a JSON document.

■ **Table 5** An example of usage of the introduced predicates for interpreting a JSON file.

| Source: ex.json                       | Façade-X Interpretation                                                                                                                                                                                                                                                                                                                                                        |
|---------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>{   "a":1,   "b":[1,2,3] }</pre> | <pre>includes(file:///ex.json,ex.json), hasRoot(ex.json,root), hasSlot(root,rs.a),StringSlot(rs.a),hasValue(rs.a,1), hasSlot(root,rs.b),StringSlot(rs.b),hasContainer(rs.b,rsb.c), hasSlot(rsb.c,rsbc.1),hasSlot(rsb.c,rsbc.2),hasSlot(rsb.c,rsbc.3), NumberSlot(rsbc.1),NumberSlot(rsbc.2),NumberSlot(rsbc.3), hasValue(rsbc.1,1),hasValue(rsbc.2,2),hasValue(rsbc.3,3)</pre> |

► **Example 4.** Table 6 show an example of usage of the introduced predicates for interpreting a XML file.

## 5 Façade-X RDF graphs

### 5.1 Mapping Façade-X to RDF

We define a mapping to RDF as a function  $\mathcal{F} : \mathcal{L} \times \mathcal{I} \rightarrow 2^{\mathcal{I} \times \mathcal{T}}$  which takes as input a query  $q$  over Façade-X and an interpretation  $\mathcal{I}$  of the model, it evaluates  $q$  over  $\mathcal{I}$  and it returns a set of quads<sup>19</sup>. The function  $\mathcal{F}$  uses the assignments of the free variables of  $q$  to instantiate the quads.

<sup>18</sup>The example shows the CSV as a single container including a list of lists. SPARQL Anything allows to interpret the first line as a schema and generate an alternative representation, using column names as `StringSlots`. However, in this work, we do not consider format-specific configurations. More details at: <https://sparql-anything.readthedocs.io/stable/formats/CSV/>.

<sup>19</sup> $2^A$  denotes the powerset of a set  $A$ , i.e. the set of all possible subsets of  $A$ .

■ **Table 6** An example of usage of the introduced predicates for interpreting a XML file.

| Source: <code>ex.xml</code>                                                                        | Façade-X Interpretation                                                                                                                                                                                                                                                                                                     |
|----------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>&lt;TEAM name="Chicago Bulls"&gt;   &lt;PLAYER name="Micheal Jordan"/&gt; &lt;/TEAM&gt;</pre> | <pre>includes(file:///ex.xml,ex.xml), hasRoot(ex.xml,root),hasType(root,TEAM), hasSlot(root,r.name),StringSlot(rname), hasValue(rname,"Chicago Bulls"), hasSlot(root,r.1),NumberSlot(r.1), hasContainer(r.1,r1c),hasType(r1c,PLAYER), hasSlot(r1c,r1c.name),StringSlot(r1c.name), hasValue(r1c.name,"Micheal Jordan")</pre> |

■ **Listing 9** Definition of Façade-X namespaces.

```
1 prefix fx: <http://sparql.xyz/facade-x/ns/>
2 prefix xyz: <http://sparql.xyz/facade-x/data/>
```

The input of  $\mathcal{F}$  is specified as a PL formula and the output is specified as a RDF graph template that include the free variable to instantiate the RDF terms. We use the symbol  $\Rightarrow$  to indicate an input-output pair of  $\mathcal{F}$ . We note that such mappings are bidirectional and conversion is loss-less, meaning that one can go back and forth without losing information.

It is worth noting that even though the purpose of mapping is different, formalising the mapping as a function that associates queries and data sources with RDF graphs is compatible with the framework proposed in [37] which associates queries with the positions of graphs to be generated. However, demonstrating the compatibility of the two theories is beyond the scope of this article.

Before providing the details of the function  $\mathcal{F}$  we introduce elements that will be used in the transformation. We define two namespaces and preferred prefix names: **fx** and **xyz** (see Listing 9).

We define the mappings (see Listing 10) between Façade-X terms and RDF. These mappings are the input-output (i.e. query, RDF template) pairs of  $\mathcal{F}$ . We use the functions **IRI**, **Entity** and **Literal** to construct RDF terms on the basis of the arguments passed to the function. Depending from the configuration given by the KG engineer, the **Entity** constructs either an IRI or a Blank Node. Moreover, we introduce the function **ContainerMembershipProperty** that given  $n \in \mathbb{N}^+$  constructs the IRI of the corresponding container membership property, i.e. **rdf:\_n**. We introduce the primitive IRI **fx:root** to represent the unary predicate **Root**. Without loss of generality, we assume that (i) **ContainerMembershipProperty** can extract the cardinal order of the slot from its identifier; (ii) **IRI** extract the name of the slot from its identifier.

► **Corollary 3.** *The mapping rules of Definition 10 are bidirectional.*

**Proof.** This is can be demonstrated by showing that the mapping of Definition 10 is a bijective, i.e. a one-to-one correspondence of distinct elements. By definition the mapping rules define a correspondence of queries and generated quads. Therefore, we have only to show that the corresponding elements are distinct. Firstly, we note that queries and generated quads are distinct entities (i.e. it is not possible to convert one into the other using implications). As queries, we note that in each query (implicitly) involves distinct elements of the model: 1: **Root**; 2: **Container** and **StringSlot**; 3: **Container** and **NumberSlot**; 4: **Value** and **StringSlot**; 5: **Value** and **NumberSlot**;

■ **Listing 10** Definition of mappings of the Façade-X to RDF.

```

1 hasRoot(ds, c)  $\equiv$  {(IRI(ds), Entity(ds, c), rdf:type, fx:root)}
2 includesContainer(ds, c1)  $\wedge$  includesContainer(ds, c2)
    $\wedge$  StringSlot(s)  $\wedge$  hasSlot(c1, s)  $\wedge$  hasContainer(s, c2)  $\equiv$ 
   {(IRI(ds), Entity(ds, c1), IRI(xyz:, s), Entity(ds, c2))}
3 includesContainer(ds, c1)  $\wedge$  includesContainer(ds, c2)
    $\wedge$  NumberSlot(s)  $\wedge$  hasSlot(c1, s)  $\wedge$  hasContainer(s, c2)  $\equiv$ 
   {(IRI(ds), Entity(ds, c1), ContainerMembershipProperty(s), Entity(ds, c2))}
4 includesContainer(ds, c)  $\wedge$  NumberSlot(s)  $\wedge$  hasSlot(c, s)  $\wedge$  hasValue(s, v)  $\equiv$ 
   {(IRI(ds), Entity(ds, c), ContainerMembershipProperty(s), Literal(v))}
5 includesContainer(ds, c)  $\wedge$  StringSlot(s)  $\wedge$  hasSlot(c, s)  $\wedge$  hasValue(s, v)  $\equiv$ 
   {(IRI(ds), Entity(ds, c), IRI(xyz:, s), Literal(v))}
6  $\wedge$  includesContainer(ds, c)  $\wedge$  hasType(c, t)  $\equiv$ 
   {(IRI(ds), Entity(ds, c), rdf:type, IRI(xyz:, t))}

```

■ **Listing 11** The result of the application of the mapping to the input of Example 1.

```

PREFIX fx:      <http://sparql.xyz/facade-x/ns/>
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

[ rdf:type      fx:root;
  rdf:_1       [ rdf:_1  "email"; rdf:_2  "name"; rdf:_3  "surname" ];
  rdf:_2       [ rdf:_1  "laura@example.com"; rdf:_2  "Laura";
                 rdf:_3  "Grey" ];
  rdf:_3       [ rdf:_1  "craig@example.com"; rdf:_2  "Craig";
                 rdf:_3  "Johnson" ];
  rdf:_4       [ rdf:_1  "mary@example.com"; rdf:_2  "Mary";
                 rdf:_3  "Jenkins" ];
  rdf:_5       [ rdf:_1  "jamie@example.com"; rdf:_2  "Jamie";
                 rdf:_3  "Smith" ]
] .

```

and 6: Type. In a similar fashion, this is also the case for the generated quads. While the first two elements are always an IRI derived from the DataSource and a Container, the property and the object are distinct: 1: `rdf:type`, `fx:root`; 2: `xyz:` property and Entity; 3: container membership property and Entity; 4: container membership property and Literal; 5: `xyz:` property and Literal; and 6: `rdf:type` and `xyz:` IRI. ◀

Listing 11 shows the result of applying the mapping rules in Listing 10 to the input of Example 1. Considering the mapping to RDF defined above, we can show that the constructed quads meet a series of conditions.

► **Corollary 4.** *The following statements apply to the constructed RDF quads:*

- (i) *The subject is<sup>20</sup> a **Container**;*
- (ii) *If the property is not **rdf:type**, then it is a **Slot**;*

<sup>20</sup>“is” in this context means “is constructed on the basis of”.

- (iii) If the property is **rdf:type** and the object is not **fx:root**, then the object is a **Type**;
- (iv) If the property is a container membership property, then the property is a **NumberSlot**;
- (v) If the object is a **Literal**, then the property is a **Slot** and the object from a **Value**.

**Proof.** The statement (i) is demonstrated by observing that the subjects in the mapping rules of the Definition 10 is always an Entity constructed on the basis of a variable bound to a Container.

The statement (ii) is demonstrated by observing that all the rules that do not produce quads having **rdf:type** as property, namely 2, 3, 4 and 5, have as property an Entity or a container membership property derived from a slot.

The statement (iii) is demonstrated by observing that the only mapping rule that produces a quad having **rdf:type** as property and an object different from **fx:root** is the 6. The quad generated by this rule has as object an Entity derived from a type *t*.

The statement (iv) is demonstrated by observing that the only mappings comply with this condition are the 3 and 4 and in both rules the predicate is derived from a **NumberSlot**.

The statement (v) is demonstrated by observing that the only mappings comply with this condition are the 4 and 5 and in both rules the predicate is derived from a **Slot** and the object from a **Value**. ◀

Considering the implications demonstrated in the Corollary 4, the mapping to RDF provided in the Definition 10 and the Façade-X described in Section 4.2, we can show that further implications hold. Note that these implications apply to the constructed quads as well as to the facts implied by the statements in Corollary 4 (this is why they are provided separately).

► **Corollary 5.** *The following statements hold:*

- (i) If the object is an IRI and the property is a **Slot**, then the object is a **Container**;
- (ii) If the object is a **Container**, then the property is a **Slot**;
- (iii) If the object is a **Value**, then the property is a **Slot**.

**Proof.** The statement (i) is demonstrated by observing that the only mapping rules that generate a IRI as object (via the function Entity) are the 2 and 4. In both, the predicate of the generated quad is a Slot.

The statement (ii) is demonstrated by observing that the only mapping rules that generate a Container as object are the 2 and 3. In both, the predicate of the generated quad is a Slot.

The statement (iii) is demonstrated by observing that the only mapping rules that generate a Value as object are the 3 and 4. In both, the predicate of the generated quad is a Slot. ◀

Finally, on the basis of the corollaries and the definitions above, we can show that further implications can demonstrated on the triple patterns satisfiable on the Façade-X graphs.

► **Corollary 6.** *The following conditions hold:*

- (i) If the object is **fx:root**, then the subject is derived from the **Root** and the property is either **rdf:type** or a variable that can match only with **rdf:type**, we call such a term **TypeProperty**;
- (ii) If the object is a **Type**, then the property is a **TypeProperty**;
- (iii) If the property of a triple pattern is neither a container membership property, nor a variable, nor a **TypeProperty**, then it must be a **StringSlot**.

**Proof.** The statement (i) is demonstrated by observing that the only mapping rule in which **fx:root** appears is at line 1 and the subject of this constructed quad is a Container and the property is **rdf:type**. Therefore, any triple pattern that has **fx:root** as object, graph must have a **TypeProperty** as property to be satisfiable on a Façade-X graph.

Similarly, regarding the statement (ii) we can observe that a Type is generated only by the mapping 6 which generates a quad having `rdf:type` as property.

The statement (iii) is demonstrated by observing that if we exclude the mapping rules that generate a container membership property are the 3 and 4 and those generating a quad having `rdf:type` as property are the 1 and 6. Therefore, if we exclude also that the property of the triple is a variable, then the property must be a **StringSlot**. ◀

## 5.2 Properties of Façade-X RDF graphs

RDF graphs are usually interpreted as *directed, labelled* (mathematical) graphs, which we refer to as “m-graphs” to avoid confusion. An m-graph can be constructed from an RDF graph by treating the subjects and objects of the triples as nodes and representing the properties as labelled edges. Considering the properties of the Façade-X model demonstrated in the previous Sections we can derive some topological properties of the corresponding m-graph.

► **Theorem 7.** *Façade-X m-graphs built from a Façade-X RDF graphs are connected.*

**Proof.** We can observe that nodes of a Façade-X m-graph can be either Containers, Values, Types or `fx:root`. Proposition 1 demonstrates that for each container there is always a sequence of slots and containers that connect it the root. Moreover, Containers can be object only of quads having a Slot as property (see 5.(ii)), hence a sequence of Containers and Slots in the Façade-X model corresponds a path of Container nodes and Slot edges. Therefore, there is always a path that connects the Root node to any Container node. Finally, we can observe that Types, Values and `fx:root` are always directly connected to the Container they belong to (see 10.1, 10.4, 10.5 and 10.6). Hence, the Façade-X m-graphs are connected. ◀

► **Theorem 8.** *Façade-X m-graphs built from a Façade-X RDF graphs are acyclic.*

**Proof.** A cycle involving a container node is not possible, as this would imply that a container contains itself recursively, which would invalidate condition 5.9. Types, `fx:root` and Values are directly connected to the container they belong to only and can not have outgoing edges. Therefore, Façade-X m-graphs are acyclic. ◀

► **Theorem 9.** *Façade-X m-graphs have a single root.*

**Proof.** The only mapping rule matching the root container is the 10.1. For each DataSource there is a single solution for the query of this rule since there is one and only one root for each DataSource, see 8.3 and 8.7. ◀

► **Theorem 10.** *In Façade-X m-graphs, there always exist a single directed path connecting the root node to any container node.*

**Proof.** Proposition 2 demonstrates that for each Container *c* there is always a single sequence of containers and slots that connects the root of the data source to *c*. Proof of Theorem 7 demonstrates that a sequence of Containers and Slots in the Façade-X model corresponds a directed path of Container nodes and Slot edges. Therefore, each container is connected to the root of the its data source by one and only one path. ◀

► **Corollary 11.** *In Façade-X m-graphs, there is either a single directed path connecting any pair of container nodes, or there is none.*

**Proof.** This is a consequence of the Theorem 10. We observe that for a given pair of nodes *x* and *y*, either *y* lies on the path from the root to *x*, or it does not. ◀

## 6 Satisfiability of Basic Graph Patterns under Façade-X

We characterise the Basic Graph Patterns that admits a solution on a Façade-X RDF graph ( $BGP^{FX}$ ). Although the number of satisfiable  $BGP^{FX}$  is infinite, we can characterise a  $BGP^{FX}$  from the Triple Patterns (TPs) that admit a solution on a Façade-X RDF Graph and the possible joins among them. Moreover, although the Façade-X model include their definition, Resource and DataSource are not involved in BGP expressions. In fact, a DataSource is mapped to a named graph (see Definition 10). A Façade-X Resource is mapped to an RDF Dataset, as intended by [53]. In what follows, we study basic graph patterns.

### 6.1 Satisfiable Triple Patterns

We remind some properties of a Façade-X RDF graph that can be derived by the theory outlined in the previous Sections.

- The subject must be a Container;
- The set of all possible predicates is the union of TypeProperty, SlotString and SlotNumber;
- The set of all possible objects is the union of Value, Container, FXRoot and Type;
- Container, TypeProperty, SlotString, SlotNumber, Value, Root and Type, of these sets are mutually disjoint by definition.

Given these conditions, we can have only 6 TPs compatible with a Façade-X graph (out of the 12 possible combinations). Table 7 shows the TPs resulting all the 12 possible combinations of Façade-X components, together with an indication of their satisfiability. Consider the Façade-X triple pattern “Container SlotNumber Value” (first row of the table). This is satisfiable, since the Façade-X graph can contain such a triple. The last column of the table indicates the SPARQL triple patterns that can match the corresponding Façade-X triple pattern. For example, the SPARQL triple pattern  $\mathcal{VIL}$  (e.g. `?s rdf:_1 "some value".`) can match “Container SlotNumber Value” as the variable `?s` can be resolved to a container, `rdf:_1` is a SlotNumber and “some value” is a Literal. The Façade-X triple pattern “Container SlotNumber Type” (second row of the table) is not satisfiable as types in Façade-X occur only in triples with a TypeProperty as predicate.

### 6.2 Satisfiable Join conditions

Without loss of generality, we can study the satisfiability of  $BGP^{FX}$  consisting of two TPs. In fact, more complex BGPs can be derived by joining BPGs of one or two TPs. Since properties never occur as the subject or object in Façade-X, joins that equate the subject or object of one TP with the predicate of another TP are not permitted. Therefore, the joins  $S \bowtie P$  and  $P \bowtie O$  will always result in a empty solution. As a result, Façade-X TPs can be involved in four types of joins, namely:  $S \bowtie S$ ,  $P \bowtie P$ ,  $S \bowtie O$ , and  $O \bowtie O$ .

However, not all Façade-X TPs can be involved in all of these joins. Consider the TPs C.SN.V and C.SN.C. The only joins allowed are  $S \bowtie S$ ,  $P \bowtie P$ , and  $S \bowtie O$ , but not  $O \bowtie O$  since Values and Containers are disjoint.

For example, the following SPARQL BGPs are satisfiable:

- `?s rdf:_1 ?v . ?s rdf:_2 ?c .` which is example of  $S \bowtie S$  join;
- `<i1> ?p "some value" . <i2> ?p <i3> .` which is example of  $P \bowtie P$  join;
- `<i> rdf:_1 ?c . ?c rdf:_2 "some value" .` which is example of  $S \bowtie O$  join;

Moreover, consider the following BGP.

```
?s1 rdf:_1 ?o .
?s2 rdf:_2 ?o .
```

■ **Table 7** Triple Patterns result in all possible combinations of Façade-X components, together with an indication of their satisfiability on a Façade-X RDF graph.

| Id     | Façade-X Triple Pattern          | Satisfiability | Corresponding SPARQL Triple Patterns                                                       |
|--------|----------------------------------|----------------|--------------------------------------------------------------------------------------------|
| C.SN.V | Container SlotNumber Value       | ✓              | $VVV, VVB, VVL, VIV, VIB, VIL, IVV, IVB, IVL, IIV, IIB, IIL, BVV, BVB, BVL, BIT, BIB, BIL$ |
| C.SN.T | Container SlotNumber Type        | –              | –                                                                                          |
| C.SN.R | Container SlotNumber FXRoot      | –              | –                                                                                          |
| C.SN.C | Container SlotNumber Container   | ✓              | $VVV, VVI, VVB, VIV, VII, VIB, IVV, IVI, IVB, IIV, IIL, IIB, BVV, BVI, BVB, BIT, BIL, BIB$ |
| C.SS.V | Container SlotString Value       | ✓              | $VVV, VVB, VVL, VIV, VIB, VIL, IVV, IVB, IVL, IIV, IIB, IIL, BVV, BVB, BVL, BIT, BIB, BIL$ |
| C.SS.T | Container SlotString Type        | –              | –                                                                                          |
| C.SS.R | Container SlotString FXRoot      | –              | –                                                                                          |
| C.SS.C | Container SlotString Container   | ✓              | $VVV, VVI, VVB, VIV, VII, VIB, IVV, IVI, IVB, IIV, IIL, IIB, BVV, BVI, BVB, BIT, BIL, BIB$ |
| C.TP.V | Container TypeProperty Value     | –              | –                                                                                          |
| C.TP.T | Container TypeProperty Type      | ✓              | $VVV, VVI, VVB, VIV, VII, VIB, IVV, IVI, IVB, IIV, IIL, IIB, BVV, BVI, BVB, BIT, BIL, BIB$ |
| C.TP.R | Container TypeProperty FXRoot    | ✓              | $VVV, VVI, VVB, VIV, VII, VIB, IVV, IVI, IVB, IIV, IIL, IIB, BVV, BVI, BVB, BIT, BIL, BIB$ |
| C.TP.C | Container TypeProperty Container | –              | –                                                                                          |

We have that either  $?o$  is a value or it is a container; it cannot be both, as these are mutually exclusive. Therefore, if other constraints from different TPs require that  $?o$  be both a container and a value, the pattern is not satisfiable. Moreover, we will see that  $?o$  can only be a **Value**.

In general, assuming the order of the triple pattern is meaningful, we have 6 valid Façade-X TPs, hence 36 Façade-X basic graph patterns of two joined Façade-X TPs. If we do not consider the order of the TPs, there are 21 possible Façade-X graph patterns<sup>21</sup>. The join condition among these patterns can be one of the 4 above, resulting in 84 combinations.

To narrow down the number of satisfiable combinations, we can make the following considerations.

- $S \bowtie S$  is always satisfiable as it equates containers (see 4.(i));
- $S \bowtie O$  is satisfiable when the joined node of the first and the joined node of the second Façade-X triple pattern are of the same Façade-X type (otherwise the graph would not have a solution as the Façade-X types are pairwise disjoint, see Definition 7).

As a result, we have 42 possible Façade-X graph patterns, which are showed in Table 8.

<sup>21</sup>This is the number of all possible combinations without repetition of 2 elements from a set of 6, i.e.  $\binom{6}{2}$ , plus the number of Façade-X graph patterns where the same pattern is repeated twice, i.e. 6.



■ **Table 8** The list of all possible joins of two Façade-X triple patterns with their corresponding admissible joining condition.

| Façade-X Graph Pattern  | Satisfiable Joins                                       |
|-------------------------|---------------------------------------------------------|
| $C.SN.V \bowtie C.SN.V$ | $S \bowtie S \ P \bowtie P \ O \bowtie O$               |
| $C.SN.V \bowtie C.SS.V$ | $S \bowtie S \ O \bowtie O$                             |
| $C.SN.V \bowtie C.SS.C$ | $S \bowtie S \ S \bowtie O$                             |
| $C.SN.V \bowtie C.TP.T$ | $S \bowtie S$                                           |
| $C.SN.V \bowtie C.TP.R$ | $S \bowtie S$                                           |
| $C.SN.C \bowtie C.SN.V$ | $S \bowtie S \ P \bowtie P$                             |
| $C.SN.C \bowtie C.SN.C$ | $S \bowtie S \ P \bowtie P \ S \bowtie O \ O \bowtie O$ |
| $C.SN.C \bowtie C.SS.V$ | $S \bowtie S$                                           |
| $C.SN.C \bowtie C.SS.C$ | $S \bowtie S \ S \bowtie O \ O \bowtie O$               |
| $C.SN.C \bowtie C.TP.T$ | $S \bowtie S$                                           |
| $C.SN.C \bowtie C.TP.R$ | $S \bowtie S$                                           |
| $C.SS.V \bowtie C.SS.V$ | $S \bowtie S \ P \bowtie P \ O \bowtie O$               |
| $C.SS.V \bowtie C.TP.T$ | $S \bowtie S$                                           |
| $C.SS.V \bowtie C.TP.R$ | $S \bowtie S$                                           |
| $C.SS.C \bowtie C.SS.V$ | $S \bowtie S \ P \bowtie P$                             |
| $C.SS.C \bowtie C.SS.C$ | $S \bowtie S \ P \bowtie P \ S \bowtie O \ O \bowtie O$ |
| $C.SS.C \bowtie C.TP.T$ | $S \bowtie S$                                           |
| $C.SS.C \bowtie C.TP.R$ | $S \bowtie S$                                           |
| $C.TP.T \bowtie C.TP.T$ | $S \bowtie S \ P \bowtie P \ O \bowtie O$               |
| $C.TP.R \bowtie C.TP.T$ | $S \bowtie S \ P \bowtie P$                             |
| $C.TP.R \bowtie C.TP.R$ | $S \bowtie S \ P \bowtie P \ O \bowtie O$               |

### 6.3 Satisfiable Basic Graph Patterns

Combining these patterns yields to more complex graph structures as those reported in Table 3. It is worth noting that we can consider a BGP as a conjunction of its sub-BGPs, if a sub-BGP violates the Façade-X model, then the BGP is unsatisfiable. Therefore, given a BGP, we could extract every possible sub-BGP from it and check whether any of these violate the Façade-X model. If we find a sub-BGP that violates Façade-X, then the BGP is unsatisfiable.

We observe that there are formulas of the model that can be violated without compromising the satisfiability of the BGP. In fact, a BGP is satisfiable if there is a *partial* mapping for its variables under which the BGP is isomorphic with a *portion* of the Façade-X RDF graph. Therefore, formulas requiring the existence of triples can be violated without compromising the satisfiability of the BGP, because the mandatory triples could be in a different portion of the graph. For example, the formula 5.1 implies that every value must be contained in a slot. However, BGPs that do not involve slots or values (e.g.  $?s \ a \ ?t$ ) can violate formula 5.1 while still being satisfiable.

This approach, i.e. analysing the BGP by its sub components to assess the satisfiability, raises three questions:

- (i) Which formulas, if violated, would make a BGP unsatisfiable?
- (ii) How can we check whether a (sub-)BGP violates a model formula?
- (iii) Which (sub-)BGPs do we need to extract?

**Answer (i).** We identify the categories for the model formulas: *type/disjoint implications* which are the formulas that allow us to reason about the types of the individuals; *min-cardinality constraints* (resp. *max-cardinality constraints*) which are the formulas that require that there

■ **Table 9** Classification of the Façade-X model formulas by the categories identified: Min-Cardinality Constraint (Min), Max-Cardinality Constraint (Max), Type Implications (Type), Disjointness (Dis), Integrity Constraints (Int), BGP Inviolable (Inv) and BGP Inviolable By Construction of the Mapping Rules (Inv BC).

| Form       | Definition 3 |   |   |   | Definition 4 |   |   |   |   |   |   |   |   |    | Definition 5 |   |   |   |   |   |   |   |   |    |
|------------|--------------|---|---|---|--------------|---|---|---|---|---|---|---|---|----|--------------|---|---|---|---|---|---|---|---|----|
|            | 1            | 2 | 3 | 4 | 1            | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 1            | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Min        |              | ✓ | ✓ |   |              |   |   |   |   |   |   |   | ✓ |    | ✓            | ✓ |   | ✓ |   |   |   |   |   |    |
| Max        |              |   |   | ✓ |              |   |   |   |   |   |   |   |   | ✓  |              |   | ✓ |   |   | ✓ | ✓ |   |   | ✓  |
| Type       | ✓            |   |   |   | ✓            | ✓ | ✓ | ✓ | ✓ |   | ✓ | ✓ |   |    |              |   |   |   |   |   |   |   |   |    |
| Disj       |              |   |   |   |              |   |   |   |   |   |   |   |   |    |              |   |   |   |   |   |   |   |   |    |
| Int        |              |   |   |   |              |   |   |   |   | ✓ |   |   |   |    |              |   |   |   | ✓ |   |   | ✓ | ✓ |    |
| BGP Inv    |              |   |   |   | ✓            |   |   |   |   |   |   |   |   |    |              |   |   |   | ✓ | ✓ | ✓ | ✓ | ✓ | ✓  |
| BGP Inv BC |              |   |   |   |              | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓  | ✓            | ✓ | ✓ | ✓ |   |   |   |   |   |    |

| Form       | Definition 6 |   | Definition 7 |   |   |   |   |   |   |   |   |    |    | Definition 8 |   |   |   |   |   |   |   |
|------------|--------------|---|--------------|---|---|---|---|---|---|---|---|----|----|--------------|---|---|---|---|---|---|---|
|            | 1            | 2 | 1            | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 1            | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Min        |              |   |              |   |   |   |   |   |   |   |   |    |    | ✓            |   | ✓ |   |   |   |   | ✓ |
| Max        |              |   |              |   |   |   |   |   |   |   |   |    |    |              |   |   |   |   |   | ✓ |   |
| Type       | ✓            | ✓ |              |   |   |   |   |   |   |   |   |    |    |              | ✓ |   | ✓ | ✓ |   |   |   |
| Disj       |              |   | ✓            | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓  | ✓  | ✓            |   |   |   |   |   |   |   |
| Int        |              |   |              |   |   |   |   |   |   |   |   |    |    |              |   |   |   |   | ✓ |   |   |
| BGP Inv    |              |   |              |   | ✓ |   |   |   |   |   |   | ✓  |    |              |   |   |   |   | ✓ | ✓ |   |
| BGP Inv BC | ✓            | ✓ | ✓            | ✓ |   | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |    | ✓  | ✓            | ✓ |   | ✓ | ✓ |   |   |   |

must be a minimum (resp. maximum) of individuals/pairs in the interpretation of a model; and *integrity constraints* which are the formulas that impose that one or multiple individuals/pairs do not exist.

It is worth noticing that min-cardinality constraints can be violated without compromising the satisfiability of the BGP. These formulas imply the (implicit) existence of triples that can exist in a different portion of the Façade-X RDF graph. Instead, violating a maximum cardinality constraint, an integrity constraint, a type/disjoint implications renders a BGP unsatisfiable, since these constraints prevent triples from existing, whereas BGP requires them to exist. These call these kind of formulas *inviolable*.

Moreover, we note that there are formulas that, by definition of the mapping rules *alone* (see 10), can not be violated. For example, formula 4.6 can not be violated as NumberSlot and StringSlot are always generated by distinct rules. Instead, for formula 4.1 to be violated, we need to know that StringSlots or NumberSlots are both Slots, which is knowledge that cannot be derived from the mappings alone. Finally, it is worth noting that type implications themselves can only lead to the unsatisfiability of the BGP when considered alongside a disjoint implication.

Table 9 classifies the model formulas according to the above-defined categories and indicates whether the formula is inviolable or inviolable by construction.

**Answer (ii).** A BGP violates a model formula if there is a mapping between its nodes and the formula's variables that renders the formula invalid (i.e. if it makes the negation of the formula true). For example, consider the formula 5.8, the triple pattern  $?x \text{ ?p } ?x$  violate it since we can assign to both  $c$  and  $s$  the variable  $?x$ . Then we can reformulate question (ii) as “how can we assign variables of the inviolable formulas to the nodes of the BGP?”

For each *inviolable* formula, we can construct, via the mapping rules in Definition 10, a *proxy graph* that violates the formula. This is the minimal graph that violates a formula and exemplifies the pattern that must be avoided in BGP to ensure satisfiability. Since mapping rules are bidirectional (see Theorem 3) assigning BGP nodes to formula variables is equivalent to assigning BGP nodes to proxy-graph nodes. Therefore, BGPs that are isomorphic to a proxy pattern violate an inviolable formula.

■ **Table 10** Inviolable formulas and corresponding proxy graph.

| Formula        | Proxy Graph                                                                                                                                                             |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 4.1, 7.3, 7.10 | $\langle i1 \rangle \text{ xyz:s } \langle i2 \rangle . \langle i3 \rangle \text{ rdf:type } \langle i1 \rangle$                                                        |
| 5.5            | $\langle i1 \rangle \text{ rdf:_1 "a" } . \langle i1 \rangle \text{ rdf:_1 } \langle i2 \rangle$                                                                        |
| 5.6            | $\langle i1 \rangle \text{ rdf:_1 } \langle i2 \rangle . \langle i1 \rangle \text{ rdf:_1 } \langle i3 \rangle$                                                         |
| 5.7            | $\langle i1 \rangle \text{ rdf:_1 "1" } . \langle i1 \rangle \text{ rdf:_1 "2"}$                                                                                        |
| 5.8            | $\langle i1 \rangle \text{ rdf:_1 } \langle i1 \rangle$                                                                                                                 |
| 5.9            | $\langle i1 \rangle \text{ rdf:_1 } \langle i2 \rangle . \langle i2 \rangle \text{ rdf:_1 } \langle i3 \rangle . \langle i3 \rangle \text{ rdf:_1 } \langle i1 \rangle$ |
| 5.10           | $\langle i1 \rangle \text{ rdf:_1 } \langle i2 \rangle . \langle i1 \rangle \text{ rdf:_2 } \langle i2 \rangle$                                                         |
| 8.6            | $\langle i1 \rangle \text{ rdf:type fx:root } . \langle i2 \rangle \text{ rdf:_1 } \langle i1 \rangle$                                                                  |
| 8.7            | $\langle i1 \rangle \text{ rdf:type fx:root } . \langle i2 \rangle \text{ rdf:type fx:root}$                                                                            |

Table 10 shows the inviolable formulas and the corresponding proxy graphs. For brevity, Table 10 omits disjoint formulas. Such formulas can be violated by forcing a node to be of two disjoint types. Most of the formulas cannot be violated due to the construction of the mapping rules, either because the two types of a disjoint occur in different roles (e.g. slots and containers are placed as properties and subjects) or because they are materialised using different RDF node types (e.g. values and containers are literals and IRIs). These conditions can be validated by checking if the triple patterns of the BGP match the valid triple pattern indicated in Table 7. Therefore, the only disjoint that can be violated is between Type and Container.

► **Remark 12.** In addition to the above conditions, there is an extra constraint that must be met to ensure the BGP is satisfiable. This constraint is treated separately from the others as it implicitly emerges from the structural properties of the Façade-X m-graph. Since there is a single path connecting the root to its containers (see 10) and there is either a single path connecting any pair of container nodes, or there is none (see 11), backward paths of containers to the root must be compatible. For example, the following BGP is not satisfiable since  $\langle iri \rangle$  which must be a container, and it is contained by two different containers:  $\langle j1 \rangle$  and  $\langle j2 \rangle$  (in case that  $?j3=?j5$ ,  $?p3=?p4$ ,  $?p2=?p4$ ), or,  $\langle j3 \rangle$  and  $\langle j5 \rangle$  (if  $j3 \neq j5$ ):

$\langle j1 \rangle ?p2 ?j3 . ?j3 ?p3 \langle iri \rangle .$   
 $\langle j4 \rangle ?p4 ?j5 . ?j5 ?p4 \langle iri \rangle . \langle iri \rangle ?p ?o .$

**Answer (iii).** It is worth noting that, in order to determine whether a BGP violates an inviolable formula, we must first establish whether the BGP contains an invalid FX triple pattern, and secondly, whether it is isomorphic to a proxy graph in Table 10. If we exclude proxy graph associated to formula 5.9 that imposes that Façade-X RDF graphs are acyclic, conditions that would make the BGP unsatisfiable would span across one or two triple patterns. Therefore, assuming that the BGP is acyclic, we can determine its validity by extracting sub-BGPs of one or two triple patterns and checking the above-defined conditions.

## 7 Detecting satisfiable Basic Graph Patterns under Façade-X

We apply the Façade-X theory defined in the previous sections to determine the satisfiability of basic graph patterns in Façade-X. We first summarize the components of the theory and their role in addressing the problem and then present two algorithms to annotate basic graph patterns and determine their satisfiability.

## 7.1 Implementing the theory

We will introduce some preliminary definitions and supporting functions that will be used in later algorithms.

**Satisfiable BGPs under Façade-X.** We define  $G$  as the set of all possible RDF graphs, and  $G^{FX}$  as the set of possible RDF graphs that derive from the mappings of definition 10. Furthermore, we define  $BGP^{22}$  as the set of SPARQL Basic Graph patterns and  $BGP^{FX} \subset BGP$  as the set of BGPs having at least one solution against at least one possible  $g \in G^{FX}$ . Note that there is a strict inclusion between  $BGP^{FX}$  and  $BGP$  as we already demonstrated that not all triple patterns/joins are satisfiable under Façade-X (see Section 6.3).

Any  $bgp \in BGP^{FX}$  must comply with the conditions determined by our theory, namely:

- C1** No cycles are allowed in BGPs (see Theorem 8);
- C2** Join conditions of the BGP must comply with those reported in Table 8;
- C3** BGPs cannot have two alternative paths that lead from any node to a container (see remark 12);
- C4** Mapping Rules (i.e. 10) and related corollaries 4, 5 and 6;
- C5** There must be *at most* one root (see 9);
- C6** Triple patterns of the BGP must comply with those reported in Table 7;
- C7** None of the sub-BGPs included in  $bgp$  must be isomorphic to the proxy graphs defined for the inviolable formulas as reported in the Table 10.

We ignore the following properties of Façade-X RDF graphs:

- Connectedness (cf. 7 and 2): because BGP don't need to, as non-joined triple patterns will be executed as their product;
- Façade-X graphs are rooted (cf. 9): because the BGP can safely match a portion of the Façade-X graph;
- Min-Cardinality constraints and conditions defined in formulas that are inviolable by construction of the mapping rules (cf. Section 6.3).

Next, we introduce functions to avoid unsupported joins, cycles and multiple paths to container nodes. These functions guarantee that the conditions C2, C1 and 12 are satisfied.

We proceed by describing the procedure to remove unsupported joins.

► **Algorithm 1** (Detection of unsupported joins).  $BGP^{FX}$  cannot have Subject/Property or Object/Property joins within or across triples. Listing 12 defines the function  $hasUnsupportedJoin(bgp) \rightarrow Bool$  to detect unsupported joins.

■ **Listing 12** Pseudocode of the  $hasUnsupportedJoin$  function.

```

1 function hasUnsupportedJoin(bgp): Bool
2   subjects = {}, properties = {}, objects = {}
3   for (s, p, o) ∈ bgp:
4     subjects = subjects ∪ {s}
5     properties = properties ∪ {p}
6     objects = objects ∪ {o}
7     if s ∈ properties ∨ o ∈ properties ∨ p ∈ subjects ∨ p ∈ objects :
8       return True
9   return False

```

<sup>22</sup>We remind that a BGP is a set of triple patterns, then  $BGP := 2^{(\mathcal{I} \cup \mathcal{B} \cup \mathcal{V}) \times (\mathcal{I} \cup \mathcal{V}) \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V})}$

**Time Complexity of `hasUnsupportedJoin`.** The time complexity depends on the number of triples in the bgp, hence  $O(|bgp|)$ .

We proceed to detect cycles in basic graph patterns.

► **Algorithm 2** (Detection of cycles). *Since Façade-X graphs are acyclic,  $BGP^{FX}$  also cannot have cycles. We define the function  $hasCycle(bgp) \rightarrow Bool$  to detect cycles in BGPs. This function explores the bgp by performing a depth-first search (DFS, see `detectCycle`) for each triple pattern of the bgp. If a node is visited two times by the search, the function returns True. `hasCycle` and `detectCycle` are defined in Listing 13.*

■ **Listing 13** Pseudocode of functions for detecting cycles.

```

1 function hasCycle(bgp): Bool
2   for (s, p, o) ∈ bgp:
3     if(detectCycle (bgp, {s}, s)):
4       return True
5   return False
6
7 function detectCycle(bgp, visited, last): Bool
8   for (last, p, next) ∈ bgp:
9     if next ∈ visited:
10      return True
11    else:
12      return detectCycle(bgp, visited ∪ {last}, next)
13  return False

```

**Time Complexity of cycle detection.** The time complexity of a depth-first search on a graph of  $n$  nodes and  $e$  edges is  $O(n + e)$ . We observe that the function `hasCycle` performs a DFS for each triple of the BGP (number of edges of the graph). Therefore in the worst case the time complexity is  $O(|bgp|^2 + |bgp| * n)$ .

Once the cycling detection is verified, we move to verify the constraints of our theory. To do that, we proceed to annotate the nodes in the pattern with the predicates resulting from the Façade-X mappings to RDF. We list the possible predicates in Table 11, which also summarises the following characteristics.

**Ground predicates.** Nodes in BGPs have roles related to the RDF triples (Subject, Predicate, Object) or roles resulting from mappings to the Façade-X model: Container, FXRoot (to represent the role of the term `fx:root`), Slot, SlotString, SlotNumber, Value, Type and TypeProperty. We organise these roles in a predicate hierarchy, where the top role is the RDF Node and the first layer Subject, Predicate, Object, which are in turn specialised by Façade-X predicates. Ultimately, the nodes in  $BGP^{FX}$  will necessarily be one of Container (Subject or Object), Type (Object), FXRoot (Object), TypeProperty (Property), SlotString (Slot which specialises Property), SlotNumber (Slot which specialises Property), or Value (Object). We name those *ground predicates* ( $GP$ ), i.e. the set of Façade-X predicates that cannot be specialised. These are marked with a \* in Table 11. Table 11 also summarises the disjointness between predicates (derived from Definition 7 and the analysis of satisfiable joins, see Table 8).

► **Definition 13** (Annotated Basic Graph Patterns). *We indicate the set of nodes of a BGP as  $\mathcal{N}$ , i.e.  $\mathcal{N} := \mathcal{V} \cup \mathcal{I} \cup \mathcal{B} \cup \mathcal{L}$ . We define the function `nodes` :  $BGP \rightarrow 2^{\mathcal{N}}$  which given a BGP returns the set of nodes it contains. We define a function `annotate` :  $BGP \rightarrow \text{Map}(\mathcal{N}, GP)^{23}$  which associates every bgp with a set of pairs (node, ground predicate).*

<sup>23</sup>  $\text{Map}(X, Y)$  indicates the set of all possible dictionaries having elements in  $X$  as keys and  $Y$  as values.

■ **Table 11** Façade-X RDF predicates hierarchy and disjointness. *Ground predicates* are marked with a \*.

| Predicate     | Specialises     | Disjoint with                                    |
|---------------|-----------------|--------------------------------------------------|
| Subject       |                 | Property                                         |
| Property      |                 | Subject, Object                                  |
| Object        |                 | Property                                         |
| TypeProperty* | Property        | Subject, Object, Slot, Type, Container           |
| Type*         | Object          | Slot, Container, Value, FXRoot                   |
| Container*    | Subject, Object | Property, Slot, Value, Type, FXRoot              |
| Slot          | Property        | Subject, Object, TypeProperty                    |
| Value*        | Object          | Property, Subject, Type, Container, FXRoot, Slot |
| FXRoot*       | Object          | Subject, Property, Container, Value, Type        |
| SlotNumber*   | Slot            | SlotString, Subject, Object                      |
| SlotString*   | Slot            | SlotNumber, Subject, Object                      |

**Relations between BGP and annotated BPGs.** Note that a BGP can be annotated in many ways, i.e.  $|annotate(bgp)| \geq 1, \forall bgp \in BGP$ . Note that the existence of an annotation for a BGP does not guarantee that it is satisfiable.

Later, we present two algorithms that, given a generic  $bgp \in BGP$ , can generate all possible annotations that satisfy the theory. In the remaining of this section, we are going to summarise the elements of the theory and introduce pseudo-code that will be later applied in the algorithms.

In what follows, we distinguish the BGP node types (Variable, IRIs, Literal) from the predicates, i.e. the *annotated roles* that each node may have in a Façade-X graph matching scenario. We omit blank nodes, as they operate as existential qualifiers and thus behave like variables do [23].

► **Definition 14.** *According to the above theory, the algorithm implements the following rule for annotating a BGP. Note that if a rule raises an inconsistency when verified, the NS term is raised, which stands for “no solution”.*

- R1** *If a node is a subject, then it is a Container;*
- R2** *If a node is the IRI  $fx:root$ , then it is FXRoot;*
- R3** *If a node is the IRI  $rdf:type$ , then it is TypeProperty;*
- R4** *If a Property is not a variable or the IRI  $rdf:type$ , then it is a Slot;*
- R5** *If the Object of a triple is not a Variable and not the IRI  $fx:root$  but Property is  $rdf:type$ , then the Object is a Type;*
- R6** *If the Object is FXRoot, then the Property is TypeProperty;*
- R7** *If the Object is Type, then Property is TypeProperty;*
- R8** *If the Property is a ContainerMembershipProperty, then the Property is SlotNumber;*
- R9** *If the Property is neither a Variable,  $rdf:type$ , or ContainerMembershipProperty, then is a SlotString;*
- R10** *If a node is a Literal, then it is a Value;*
- R11** *If the Object is a IRI and Property is Slot, then Object is a Container;*
- R12** *If the Object is a Value, then the Property is Slot;*
- R13** *If the Object is an IRI not equal to  $fx:root$  and it is annotated as FXRoot, then raise NS;*
- R14** *If the Object is a Type and is  $fx:root$ , then raise NS;*
- R15** *If the Object is an IRI and is a Value, then raise NS;*
- R16** *If the Object is a Container, then Predicate is Slot;*

- R17** *If two triples have the same subject and predicate, and the predicate is a Slot, they also need to have matching objects - where either or are variables or are the same node;*
- R18** *If there is a subject/object join when the object is FXRoot, then raise NS;*
- R19** *If a node is Object and Container or FXRoot and there are disjoint paths ( $S \bowtie O$  joins) leading to it, then raise NS.*

**Data structures used in the algorithms.** We introduce the data structures which will be referred to by both algorithms. These are: **(i)** the input Basic Graph Pattern, i.e. a sequence of triples denoted by the variable **bgp**; **(ii)** the set of predicates of Façade-X indicated as **terms**; **(iii)** the Façade-X RDF terms hierarchy, as presented in Table 11, i.e. a map, named **specialisations**, which associates each Façade-X predicate to its direct super type; **(iv)** the Façade-X RDF terms disjointness Table 11, a map, named **disjointness** which associates each Façade-X predicate to the set of predicates it is disjoint with; **(v)** ground terms, i.e. Façade-X set of terms that cannot be specialised, indicated as **groundTerms**; **(vi)** top terms, i.e. terms that do not specialise other terms, indicated as **topTerms**.

**Implementation of inference rules.** We assume that the consistency rules presented above (see R1-R19) are implemented in form of implications (**head**  $\leftarrow$  **body**), meaning that the head is applied if the body is true. These rules are accessible through the variable **rules**. We also introduce the functions: **(i)** **body(rule, node, map) : Bool** which takes a rule a node of the BGP and a  $map \in \text{Map}(\mathcal{N}, \text{terms})$ , it evaluates the body of the rule and returns True if the condition holds, False otherwise; **(ii)** **head(rule, node, map) : Bool** which applies the head of a rule (returns the implied FX term or NS).

► **Algorithm 3** (Checking constraints from inference rules). *We introduce the function **check** (Listing 14), which verifies the consistency of a bgp against the set of rules presented before. The function iterates through the nodes of the bgp, applying each rule in **rules** to each node. If the body of the rule is true, then it obtains the inferred term from the head of the rule. If the the inferred term is inconsistent with the other terms already associated with the node, it returns False; otherwise, it continues with the next rule. The function returns True if no inferred term raises an inconsistency.*

■ **Listing 14** Pseudocode of check function.

```

1 function check(map, bgp): Bool
2   for node → term in map:
3     for rule in rules:
4       if body (rule, node, map) then:
5         result ← head(rule, node, map)
6         if result = NS or result in disjointness[term]:
7           return False
8   return True

```

**Time complexity of check.** The time complexity of the function **check** is  $O(n * r)$  where  $n$  is the number of nodes in the bgp and  $r$  is the number of rules, i.e. R19. Since we can consider  $r$  a constant parameter, then we can conclude that **check** is linear in the number of nodes of the bgp.

**match(n1, n2).** We define a function **match(n1, n2) : Bool** that implements the SPARQL specification (SPARQL Basic Graph Pattern Matching) [23]. The function compares the nodes' types, if any of the two is a variable, returns True. If both aren't, it checks node equality (see Table 12).



■ **Table 12** The  $match(left, right)$  function compares two nodes.

| left          | right         | matches(left, right) |
|---------------|---------------|----------------------|
| $\mathcal{V}$ | $\mathcal{V}$ | True                 |
| $\mathcal{V}$ | $\mathcal{I}$ | True                 |
| $\mathcal{V}$ | $\mathcal{L}$ | True                 |
| $\mathcal{I}$ | $\mathcal{I}$ | =                    |
| $\mathcal{I}$ | $\mathcal{L}$ | False                |
| $\mathcal{L}$ | $\mathcal{L}$ | =                    |

► **Algorithm 4** (Function `satisfiesUniquePathToRoot`).

We present the function `satisfiesUniquePathToRoot(bgp, map, node):Bool` (see Listing 15 for the pseudocode) which guarantees that the constraint R19 is satisfied. The function takes as argument (i) `bgp` - a basic graph pattern; (ii) `map`  $\in \text{Dict}(\mathcal{N}, \text{terms})$  - nodes with annotations (e.g. ‘ $?x \rightarrow \text{Slot}$ ’); and (iii) a focus node  $n$  so that  $(n \rightarrow \text{Container}) \in \text{map}$  or  $(n \rightarrow \text{FXRoot}) \in \text{map}$ . The algorithm operates as follows:

1. From `bgp`, construct the set `triplePairs` containing pairs of distinct triples that have `node` as object (cf. line 2). This can be an empty set.
2. We collect backward triple paths. For each pair of triples so that  $\text{Triple}(\_, \_, \text{node})$ , and each triple in the pair, get the lists of triples having recursively  $\text{St}\bowtie\text{O}$  joins from the BGP (cf. function `walkBackwards`). `backwardsTriplePathPairs` is a list of tuples, holding two backwards paths for each triple pair.
3. Next, for each triple path pair, we transform each triple path in a sequence of backward node paths starting from node (cf. function `nodesPath`). `backwardsNodePathPairs` is a list of tuples, holding two backwards node paths for each pair.
4. We walk each paths’ pair.
5. Paths can have the same length or not. When there is a node in both paths, we check whether they are compatible, ie. we match the nodes in the same position, relying on the `match` function defined before. If two nodes don’t match, the `bgp` does not satisfy the constraint at R19.
6. If length does not match (lines 10-16), we verify that the shortest path of the two ends with a node that is not being annotated as `FXRoot`. Since both paths must be compatible for the `bgp` to be satisfiable, if the shortest path ends with a root container, the `bgp` is not satisfiable (cf. line 26).
7. If the algorithm proceeds, all conditions are satisfied: the solution graph does not need to have alternative unmatching paths to a container or root node.

**Time complexity of `satisfiesUniquePathToRoot`.** In the worst case, the function `satisfiesUniquePathToRoot` constructs a path pair for each node. Since the dominant part of the function involves going through pairs of paths and checking that each node does not invalidate any conditions, the time complexity of the function is  $O(n^2)$ .

## 7.2 Correctness and Completeness with respect to Façade-X

To guarantee the satisfiability of a BGP, it is necessary and sufficient to verify the conditions C1-C7. Here, we summarise how all the conditions are captured by our framework.

**C1, C2 and C3.** As discussed earlier, C1, C2 and C3 are verified by executing the functions `hasCycle`, `hasUnsupportedJoin`, and `satisfiesUniquePathToRoot`.

■ **Listing 15** Pseudocode of the `satisfiesUniquePathToRoot` function.

```

1 function satisfiesUniquePathToRoot(bgp, map, node): Bool
2   triplePairs ← {(s1, p1, node), (s2, p2, node)} | (s1, p1, node) ∈ bgp ∧ (s2, p2,
   node) ∈ bgp ∧ ((s1 ≠ s2) ∨ (p1 ≠ p2))}
3   backwardsTriplePathPairs ← {(walkBackwards([left], bgp), walkBackwards([right], bgp))
   | (left, right) ∈ triplePairs}
4   backwardsNodePathPairs ← {(nodesPath(left), nodesPath(right)) | (left, right) ∈
   backwardsTriplePathPairs}
5   for (leftList, rightList) in backwardsNodePathPairs:
6     for i ∈ [0..max(|leftList|, |rightList|)]:
7       if |leftList| = |rightList|:
8         if match(leftList[i], rightList[i]) = False:
9           return False
10        else:
11          if i < |leftList|:
12            shortestEndNode = rightList[|rightList|]
13          else:
14            shortestEndNode = leftList[|leftList|]
15          if (shortestEndNode, _, r) \in bgp ∧ map[r] = FXRoot:
16            return False
17   return True
18
19 function walkBackwards(list, bgp): List
20   (s, p, o) ← list[|list|]
21   if (s, p, s) ∈ bgp:
22     list ← list + [(s, p, s)]
23     return walkBackwards(list, bgp)
24   else:
25     return list
26
27 function nodesPath(triplePath): List
28   path = []
29   for (s, p, o) ∈ triplePath:
30     if path = []:
31       path = [o]
32     else: path =
33       path + [p, s]
34   return path

```

The other conditions (C4, C6, C5, C7) are verified by the `check` function via the inference rules R1-R19, as follows.

**C4 and C6.** R1-R12 guarantee that the BPG's triple patterns comply with the Façade-X triple patterns and mapping rule:

- R1 implies that all the subject must be a Container;
- R3, R5, R6, R7, R8 and R9 imply that the predicate of a BGP's TP is either a Slot or TypeProperty.
- R5, R10, R11 imply that the object of a TP is a Container or a Type or a Value or FXRoot. Note that if multiple incompatible predicates are associated with a node, the function `check` raises NS.

So far, satisfying the conditions guarantees that satisfiable Façade-X TPs are identified.

Next, we must demonstrate that unsatisfiable TPs are excluded. To this end, we go through the unsatisfiable patterns of Table 7, to demonstrate that they are excluded by the rules. **C.SN.T**, **C.SN.R**, **C.SS.T**, **C.SS.R** are excluded by the R11, which requires the Object of a Slot to be a Container that is disjoint with Type and FXRoot. **C.TP.V** is excluded by the R12, which

requires the property of a triple having Value as Object to be a Slot (and Slot is disjoint with TypeProperty).  $C.TP.C$  is excluded by the rule R16, which requires the property of a triple having Container as Object to be a Slot. Thus, we have shown how all unsatisfiable triple patterns are identified by the rules.

**C5.** Condition C5 is guaranteed by the R2 and R19. In fact, R2 guarantees that the IRI  $fx:root$  is annotated as FXRoot and R19 guarantees that there can not be multiple path leading to it.

**C7.** Last, we go through the Proxy Graph reported in Table 10 and we demonstrate that are all excluded by the rules. Proxy Graph for Formulas 4.1, 7.3, 7.10 raises a NS since  $\langle i1 \rangle$  is annotate as both Container (cf. R1) and Type (cf. R3 and R5) and they are disjoint predicates. 5.5, 5.6, and 5.7 are excluded by R17 since we have two triples have the same subject and predicate, the predicate is a Slot, but the objects do not match. 5.8 and 5.9 are cycles, then excluded by the function `hasCycle`. 5.10 is excluded by the function `satisfiesUniquePathToRoot`. 8.6 is excluded by R18 since it is a subject/object join when the object is FXRoot. Finally, 8.7 guarantees that the BGP has a single root and, as discussed earlier, this is enforced by R2 and R19.

### 7.3 Annotating graph patterns

We investigate two approaches to annotate basic graph patterns, ie. implementing the function of definition 13: (1) a top-down algorithm, addressing the annotation process as a search, and (2) a bottom-up algorithm, addressing the annotation process as constraint satisfaction one.

#### 7.3.1 Top-down algorithm

A top-down algorithm could address the annotation of the BGP problem as searching the space of annotated patterns for any (or all) annotations that comply with the theory. If none is found, the input *bgp* is not satisfiable. Clearly, it would be sufficient to find one viable solution for the BGP to be satisfiable. However, the implementation allows for finding all possible valid annotated BGPs (by setting the argument *complete* as `True`).

► **Algorithm 5** (Top-down algorithm: annotation as search). *The algorithm (whose pseudocode is provided in Listing 16) operates as follows:*

1. *It initialise an initial annotation  $start \in Map(\mathcal{N}, terms)$ , where each node is associated with its position in the triple it occurs in. For instance, given the *bgp*  $\{?a ?b ?c . ?c ?d ?e\}$ , it would generate the map  $\{(?a \rightarrow Subject) (?b \rightarrow Property) (?c \rightarrow Object) (?d \rightarrow Property) (?e \rightarrow Object)\}$ ;*
2. *Next, the recursive function `search` is called (cf. line 3);*
3. *For each node in the map, we first lookup for possible specialisations, and generate a copy of the map with a modified annotation (cf. line 9);*
4. *Next, we verify that the set of annotations is consistent with the constraints defined in 14 (cf. line 10);*
5. *If the *bgp* is consistent, and only ground predicates are mentioned in the annotation, the *bgp* is satisfiable. The current valid map is added to the solutions (cf. line 12). If *complete* is `False`, the solution is returned and the process interrupted, otherwise, we keep generating alternative solutions (cf. lines 6-7);*
6. *If the *bgp* is consistent but some annotations can still be specialised, we keep searching within the current space (cf. line 16).*

■ **Listing 16** Pseudocode of `annotate_topdown` function.

```

1 function annotate_topdown(bgp, complete) : List
2   map ← { node → Subject | (node,_,_) ∈ bgp } ∪ { node → Predicate | (_,node,_) ∈ bgp }
3     ∪ { node → Object | (_,_,node) ∈ bgp }
4   return search ( map , bgp, {}, complete)
5
6 function search (map , bgp, solutions, complete): List
7   for node → term in map:
8     for (subterm → term) in specialisations:
9       newMap ← copy(map)
10      newMap[node] ← subterm
11      if check(newMap, bgp):
12        if {term | node ← term ∈ newMap} ⊆ GP:
13          solutions ← solutions + [newMap]
14          if complete:
15            return solutions
16        else:
17          solutions ← solutions + search(newMap, bgp, solutions, complete)
18      return solutions

```

**Time complexity of `annotate_topdown`.** The dominant part of the algorithm is the search function. This function specialises multiple times the nodes of the bgp and checks the validity of the annotation every time a term is specialised. Therefore, the complexity of the `annotate_topdown` depends on the number of nodes of the bgp ( $n$ ), the number of specialisation relation in the predicate hierarchy  $|specialisations|$  and the complexity of `check`, i.e.  $O(n * r)$ . Furthermore, if the solution contains predicates other than ground ones, the function specialises the solution further. The process is repeated at most  $|terms| - |GP|$  times. Therefore, the complexity of `annotate_topdown` is quadratic in the number of nodes in the bgp and linear in the number of specialisations and inference rules, i.e.  $O(n^2 * r * |specialisations| * (|terms| - |GP|))$ .

### 7.3.2 Bottom-up algorithm

The second algorithm uses the same components (inference rules) of the previous one, but operates by first annotating bgp triples with all *grounded predicates* specialising Subject, Property, or Object, and then filters out inconsistent combinations.

► **Algorithm 6** (Bottom-up algorithm). *A generate function produces valid annotations of a bgp. When the parameter complete equals to True, it stops when one valid solution is found.*

1. We extract the list of unique nodes from the bgp (cf. line 2);
2. We populate a list with, for each node, the set of ground predicates that could match the nodes' position in any of the triples of the bgp (cf. lines 5-13). Note that  $|nodesTerms| = |nodes|$  and for any index  $i$  in the list,  $nodes[i] \rightarrow nodeTerms[i]$ . We assume that  $nodes[n]$  where  $n$  is a node of the BGP returns the set of ground predicates associated with  $n$ .
3. We refine the ground predicates by some of the inference rules (cf. Definition 14) whose application conditions are at the triple level (lines 15-22).
4. We generate possible annotations in line 16. *hypotheses* is a set of lists (tuples) of size  $|nodes|$ , each one with a possible combinations of ground predicates (line 24).
5. In the next steps, we verify each combination against our conditions with the check function (cf. lines 26-32).
6. If the argument complete is set to False, the process ends when the first valid solution is found (cf. lines 29-32).
7. Finally, all valid solutions are returned, or an empty set, if no solution was found (cf. line 33).

■ **Listing 17** Pseudocode of the bottom-up algorithm.

```

1 function generate_bottomup(bgp, complete):
2   nodes ← [{s|(s,_,_)∈bpg} ∪ {p|(_,p,_)∈bpg} ∪ {o|(_,_,o)∈bpg}]
3   nodesTerms ← []
4
5   for node in nodes:
6     t ← {}
7     if (node, p, o) ∈ bgp:
8       t ← { term | term ∈ groundTerms ∧ topTerm(term) = 'Subject' }
9     if (s, node, o) ∈ bgp:
10      t ← t + { term | term ∈ groundTerms ∧ topTerm(term) = 'Predicate' }
11     if (s, p, node) ∈ bgp:
12      t ← t + { term | term ∈ groundTerms ∧ topTerm(term) = 'Object' }
13     nodesTerms ← nodesTerms + t
14
15   for (s, p, o) ∈ bgp:
16     nodes[s] = {'Container'}
17     if p is an IRI:
18       if p ∈ TypeProperty:
19         nodes[p] = {'TypeProperty'}
20         nodes[o] = nodes[o] \ {'Container', 'Value'}
21       else:
22         nodes[o] = nodes[o] \ {'Type', 'Root'}
23
24   hypotheses ← nodeTerms[1] × nodeTerms[2] × ... × nodeTerms[|nodes|]
25
26   solutions = {}
27   for h ∈ hypotheses:
28     map = {nodes[i] → h[i] | 1 ≤ i ≤ |nodes|}
29     if check(map, bgp):
30       solutions ← solutions ∪ {map}
31     if complete = False:
32       return solutions
33   return solutions

```

**Time complexity of generate\_bottomup.** The dominant part of the algorithm is the iteration at lines 27-32. This goes through the hypotheses, which are  $n * |GP|$ . Then, for each hypothesis, it generates an annotation (map) and checks its validity. Therefore, the time complexity of generate\_bottomup is  $O(n^2 * |GP| * r)$ .

## 8 Experiments

We report on three experiments. The first compares the two algorithms described in Section 7 with a curated set of BGPs of increasing complexity, and provide empirical evidence of their performance. The second experiment focuses on real world queries, and has the objective of testing the relevance and practical feasibility of the approach. The third experiment shows the advantages of checking whether a query can be satisfied before evaluating it, considering the case of a knowledge graph construction benchmark.

### 8.1 Setting

We prepare a set of BGPs with different characteristics and consider two regimes: (a) find the first satisfiable annotation (ensure satisfiability); or (b) generate all satisfiable annotations (describe solution patterns).

■ **Table 13** List of BGPs used in experiments.

| BGP label | Satisfiable? | Triple patterns | Features                      |
|-----------|--------------|-----------------|-------------------------------|
| N_1T      | No           | 1               | only variables, no joins      |
| N_2J      | No           | 2               | only variables, with joins    |
| N_2P_R    | No           | 2               | multiple paths to fx:root     |
| N_2T      | No           | 2               | only variables, no joins      |
| N_3J      | No           | 3               | only variables, no joins      |
| N_3P_C    | No           | 3               | multiple paths to a container |
| N_3P_R    | No           | 3               | multiple paths to fx:root     |
| N_3T      | No           | 3               | only variables, no joins      |
| N_4J      | No           | 3               | only variables, with joins    |
| N_4P_C    | No           | 4               | multiple paths to a container |
| N_4T      | No           | 4               | only variables, no joins      |
| N_5J      | No           | 5               | only variables, with joins    |
| N_5P_C    | No           | 5               | multiple paths to a container |
| N_5T      | No           | 5               | only variables, no joins      |
| S_1T      | Yes          | 1               | only variables, no joins      |
| S_2J      | Yes          | 2               | only variables, with joins    |
| S_2P_R    | Yes          | 2               | multiple paths to fx:root     |
| S_2T      | Yes          | 2               | only variables, no joins      |
| S_3J      | Yes          | 3               | only variables, with joins    |
| S_3P_C    | Yes          | 3               | multiple paths to a container |
| S_3T      | Yes          | 3               | only variables, no joins      |
| S_4J      | Yes          | 4               | only variables, with joins    |
| S_4P_C    | Yes          | 4               | multiple paths to a container |
| S_4T      | Yes          | 4               | only variables, no joins      |
| S_5P_C    | Yes          | 5               | multiple paths to a container |
| S_5T      | Yes          | 5               | only variables, no joins      |

The algorithms are implemented in Java. Reported results were performed on a MacBook Pro with 8-Core Intel Core i9 (2.3 GHz) processor and Java 17 and Maven 3.9. We report on results of ten executions, aggregated with average duration (in milliseconds) and standard deviation. Since the satisfiability assessment will be part of a process performed at query time, we set the upper limit of 5 seconds, after which we stop the execution and consider the algorithm not viable. We leave possible optimisations of the algorithms and experimenting with a broader class of BGPs to future work.

The proof-of-concept implementation and the experimental code is published and experiments can be reproduced [15].

## 8.2 Comparing two algorithms

We report on experiments with the two algorithms described in Section 7. Both implementations perform all checks described in Section 7 (unsupported joins and cycles), before proceeding with the annotation of the BGPs. With these experiments, we want to demonstrate the practical feasibility of the approach to solving Façade-X BGP satisfiability problem.

■ **Table 14** Results for algorithm Top down (Search): only satisfiability.

| name   | satisfiable? | found | type | size | ms<br>(avg) | ms<br>(std) | tested<br>(avg) | tested<br>(std) |
|--------|--------------|-------|------|------|-------------|-------------|-----------------|-----------------|
| N_1T   | FALSE        | 0     | T    | 1    | 1.7         | 0.67        | 12              | 0               |
| N_2J   | FALSE        | 0     | J    | 2    | 0           | 0           | 0               | 0               |
| N_2P_R | FALSE        | 0     | P    | 2    | 71          | 20.11       | 1957            | 0               |
| N_2T   | FALSE        | 0     | T    | 2    | 478.8       | 84.63       | 16364           | 0               |
| N_3J   | FALSE        | 0     | J    | 3    | 0.1         | 0.32        | 0               | 0               |
| N_3P_C | FALSE        | 0     | P    | 3    | 2446.7      | 211.66      | 109601          | 0               |
| N_3P_R | FALSE        | 0     | P    | 3    | 266.7       | 16.77       | 12330           | 0               |
| N_3T   | FALSE        | -1    | T    | 3    | -1          | 0           | -1              | 0               |
| N_4J   | FALSE        | 0     | J    | 4    | 0           | 0           | 0.9             | 0.74            |
| N_4P_C | FALSE        | -1    | P    | 4    | -1          | 0           | -1              | 0               |
| N_4T   | FALSE        | -1    | T    | 4    | -1          | 0           | -1              | 0               |
| N_5J   | FALSE        | 0     | J    | 5    | 0           | 0           | 5.4             | 1.65            |
| N_5P_C | FALSE        | -1    | P    | 5    | -1          | 0           | -1              | 0               |
| N_5T   | FALSE        | -1    | T    | 5    | -1          | 0           | -1              | 0               |
| S_1T   | TRUE         | 1     | T    | 1    | 0.6         | 0.52        | 31              | 3.2             |
| S_2J   | TRUE         | 1     | J    | 2    | 0.5         | 0.53        | 16.3            | 1.49            |
| S_2P_R | TRUE         | 1     | P    | 2    | 17.9        | 1.73        | 636.3           | 15.81           |
| S_2T   | TRUE         | 1     | T    | 2    | 0.3         | 0.48        | 14.8            | 2.49            |
| S_3J   | TRUE         | 1     | J    | 3    | 1146.4      | 68.34       | 193106          | 1468.36         |
| S_3P_C | TRUE         | 1     | P    | 3    | 73.4        | 7.35        | 13808.7         | 250.37          |
| S_3T   | TRUE         | 1     | T    | 3    | 0.2         | 0.42        | 57.9            | 5.74            |
| S_4J   | TRUE         | 1     | J    | 4    | 2671.5      | 196.85      | 439371.1        | 3103.24         |
| S_4P_C | TRUE         | -1    | P    | 4    | -1          | 0           | -1              | 0               |
| S_4T   | TRUE         | 1     | T    | 4    | 0.3         | 0.48        | 90.4            | 9.58            |
| S_5P_C | TRUE         | -1    | P    | 5    | -1          | 0           | -1              | 0               |
| S_5T   | TRUE         | 1     | T    | 5    | 1           | 0           | 173.3           | 16.75           |

### 8.2.1 Data

We generated 27 test BGPs with different characteristics. These are listed in Table 13. BGP labels are informative:

- S: the bgp is satisfiable
- N: the bgp is not satisfiable
- *number*: the number of triples in the bgp
- T: the bgp contains only variables without joins
- J: the bgp contains one or more joins
- P: the bgp contains a join on an object and must satisfy the single-path-to-root condition
- R: the bgp contains a join on an object that is fx:root
- C: the bgp contains a join on an object that is a container

### 8.2.2 Results

Results are shown in tables 14-17. Table 14 show results for the Top down (Search) algorithm when doing satisfiability check, ie. the algorithm stops when the first satisfiable annotation is found. We report the speed in average milliseconds and the standard deviation. Furthermore, we show the



■ **Table 15** Results for algorithm Top down (Search): generate all satisfiable annotations.

| name   | satisfiable? | found | type | size | ms<br>(avg) | ms<br>(std) | tested<br>(avg) | tested<br>(std) |
|--------|--------------|-------|------|------|-------------|-------------|-----------------|-----------------|
| S_1T   | TRUE         | 6     | T    | 1    | 6           | 0.67        | 187.4           | 17.88           |
| S_2J   | TRUE         | 36    | J    | 2    | 541.9       | 62.94       | 72895.7         | 795.27          |
| S_2P_R | TRUE         | 1     | P    | 2    | 33.8        | 9.38        | 6445.1          | 291.25          |
| S_2T   | TRUE         | 32.3  | T    | 2    | 2917.4      | 1055.36     | 603611          | 212133.29       |
| S_3J   | TRUE         | -1    | J    | 3    | -1          | 0           | -1              | 0               |
| S_3P_C | TRUE         | -1    | P    | 3    | -1          | 0           | -1              | 0               |
| S_3T   | TRUE         | -1    | T    | 3    | -1          | 0           | -1              | 0               |
| S_4J   | TRUE         | -1    | J    | 4    | -1          | 0           | -1              | 0               |
| S_4P_C | TRUE         | -1    | P    | 4    | -1          | 0           | -1              | 0               |
| S_4T   | TRUE         | -1    | T    | 4    | -1          | 0           | -1              | 0               |
| S_5P_C | TRUE         | -1    | P    | 5    | -1          | 0           | -1              | 0               |
| S_5T   | TRUE         | -1    | T    | 5    | -1          | 0           | -1              | 0               |

■ **Table 16** Results for algorithm Bottom up (CSP): only satisfiability.

| name   | satisfiable? | found | type | size | ms<br>(avg) | ms<br>(std) | tested |
|--------|--------------|-------|------|------|-------------|-------------|--------|
| N_1T   | FALSE        | 0     | T    | 1    | 17.5        | 1.84        | 2      |
| N_2J   | FALSE        | 0     | J    | 2    | 0.3         | 0.48        | 2      |
| N_2P_R | FALSE        | 0     | P    | 2    | 6.2         | 0.79        | 36     |
| N_2T   | FALSE        | 0     | T    | 2    | 3.4         | 0.7         | 24     |
| N_3J   | FALSE        | 0     | J    | 3    | 0           | 0           | 24     |
| N_3P_C | FALSE        | 0     | P    | 3    | 2.9         | 0.88        | 18     |
| N_3P_R | FALSE        | 0     | P    | 3    | 8           | 3.16        | 36     |
| N_3T   | FALSE        | 0     | T    | 3    | 16.6        | 2.46        | 288    |
| N_4J   | FALSE        | 0     | J    | 4    | 0           | 0           | 288    |
| N_4P_C | FALSE        | 0     | P    | 4    | 2.8         | 0.92        | 54     |
| N_4T   | FALSE        | 0     | T    | 4    | 93.8        | 9.35        | 3456   |
| N_5J   | FALSE        | 0     | J    | 5    | 0.1         | 0.32        | 3456   |
| N_5P_C | FALSE        | 0     | P    | 5    | 7.4         | 1.58        | 162    |
| N_5T   | FALSE        | 0     | T    | 5    | 536.1       | 78.69       | 41472  |
| S_1T   | TRUE         | 1     | T    | 1    | 0.5         | 0.53        | 12     |
| S_2J   | TRUE         | 1     | J    | 2    | 0.1         | 0.32        | 144    |
| S_2P_R | TRUE         | 1     | P    | 2    | 0.6         | 0.52        | 36     |
| S_2T   | TRUE         | 1     | T    | 2    | 0.4         | 0.52        | 144    |
| S_3J   | TRUE         | 1     | J    | 3    | 1.4         | 0.7         | 432    |
| S_3P_C | TRUE         | 1     | P    | 3    | 0.8         | 0.42        | 18     |
| S_3T   | TRUE         | 1     | T    | 3    | 0.9         | 0.32        | 1728   |
| S_4J   | TRUE         | 1     | J    | 4    | 12.1        | 2.42        | 5184   |
| S_4P_C | TRUE         | 1     | P    | 4    | 0.5         | 0.53        | 54     |
| S_4T   | TRUE         | 1     | T    | 4    | 5.1         | 0.99        | 20736  |
| S_5P_C | TRUE         | 1     | P    | 5    | 0.3         | 0.48        | 162    |
| S_5T   | TRUE         | 1     | T    | 5    | 361.4       | 46.9        | 248832 |

■ **Table 17** Results for algorithm Bottom up (CSP): generate all satisfiable annotations.

| name   | satisfiable | found | type | size | ms<br>(avg) | ms<br>(std) | gen.   |
|--------|-------------|-------|------|------|-------------|-------------|--------|
| S_1T   | TRUE        | 6     | T    | 1    | 1           | 0.47        | 12     |
| S_2J   | TRUE        | 36    | J    | 2    | 1.1         | 0.32        | 144    |
| S_2P_R | TRUE        | 1     | P    | 2    | 0.4         | 0.52        | 36     |
| S_2T   | TRUE        | 36    | T    | 2    | 1.1         | 0.32        | 144    |
| S_3J   | TRUE        | 60    | J    | 3    | 3           | 0.47        | 432    |
| S_3P_C | TRUE        | 4     | P    | 3    | 0.5         | 0.53        | 18     |
| S_3T   | TRUE        | 216   | T    | 3    | 15.1        | 1.85        | 1728   |
| S_4J   | TRUE        | 300   | J    | 4    | 36.7        | 5.48        | 5184   |
| S_4P_C | TRUE        | 8     | P    | 4    | 0.9         | 0.32        | 54     |
| S_4T   | TRUE        | 1296  | T    | 4    | 171.9       | 31.8        | 20736  |
| S_5P_C | TRUE        | 16    | P    | 5    | 2.2         | 0.63        | 162    |
| S_5T   | TRUE        | 7776  | T    | 5    | 1754.5      | 270.5       | 248832 |

number of solution patterns that are evaluated. When reporting intermediate solutions evaluated, we include both average and standard deviation, as the algorithm is not fully deterministic, as the temporary search states are not ordered and the process stops when the first solution is found (without completing the search). It can be seen that the algorithm failed to return a result in five seconds in several cases. Table 15 shows results for the Top-down (Search) algorithm when generating all satisfiable annotations, ie. the algorithm proceeds to find all solution patterns. BGPs with more than two triples required more than five seconds. Results show how with just 2 triples (S\_2T), the algorithm generated 36 viable solutions after evaluating an average of  $585926.3 \pm 8903.79$  possible annotations. We can conclude that this algorithm is inefficient.

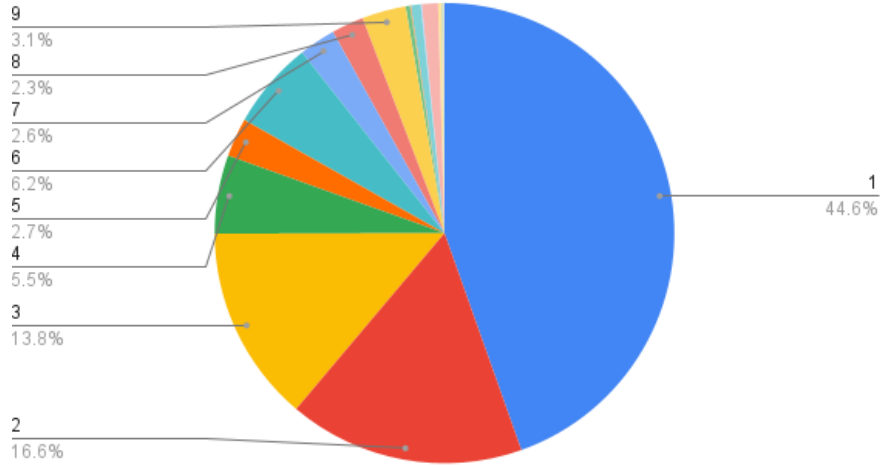
Table 16 shows results for the Bottom up (CSP) algorithm when doing satisfiability check, ie. the algorithm stops when the first satisfiable annotation is found. Average and standard deviation for the number of solutions generated is fixed as the algorithm is fully deterministic (although the process stops when the first viable solution pattern is found). Table 17 shows results for the Bottom-up (CSP) algorithm when generating all satisfiable annotations, ie. the algorithm proceeds to find all solution patterns. With this algorithm, we managed to evaluate all BGPs for satisfiability and, in addition, to generate all possible solution patterns, in the worst case, in less than two seconds. However, the worst results (N\_5T) goes still above the second, which opens the question whether it would be possible to perform further optimisations. We can conclude that the Bottom-up (CSP) algorithm is more efficient and practically sustainable.

### 8.3 Evaluation with real world queries

We evaluate the practical feasibility with basic graph patterns derived from real world queries. We only perform experiments with the most efficient, bottom up algorithm (Listing 6). Here, we summarise the data and results. We refer the reader to Appendix A for extensive details on data, results, and reproducibility.

#### 8.3.1 Data

We extract 1,309 BGPs from 449 queries collected from public GitHub repositories. 228 of those queries have only a single BGP (50.7%). As illustrated in figures 3 and 4, 83.7% of the queries have less than 5 BGPs, 44.6% of the collected BGPs have a single triple, while 75.9% of them



■ **Figure 3** Dimension of BGPs from real-world queries: number of triples.

have up to 5 variables. We can conclude that the majority of real world queries and related BGPs have few triples and variables, and have similar sizes to the curated BGPs designed to evaluate the two algorithms. See further details in Appendix A.

### 8.3.2 Results

Satisfiability (finding the first valid solution pattern) can be verified in less than 100ms for all BGPs with up to 10 triples or 14 variables. The only problematic BGP is one with 19 triples and 19 variables (“query-77.rq” in the supplemental material), which cannot be verified in less than 5 seconds. Similarly, all solution patterns can be generated efficiently in all but two cases (below half second). The only two problematic BGPs are one with 19 triples and 19 variables (“query-77.rq” in the supplemental material), and another with 20 triples and 23 variables (“query-244.rq” in the supplemental material), for which it is not possible to generate all solution patterns in less than 5 seconds. See further details in Appendix A.

## 8.4 Evaluation of benefits with benchmark queries

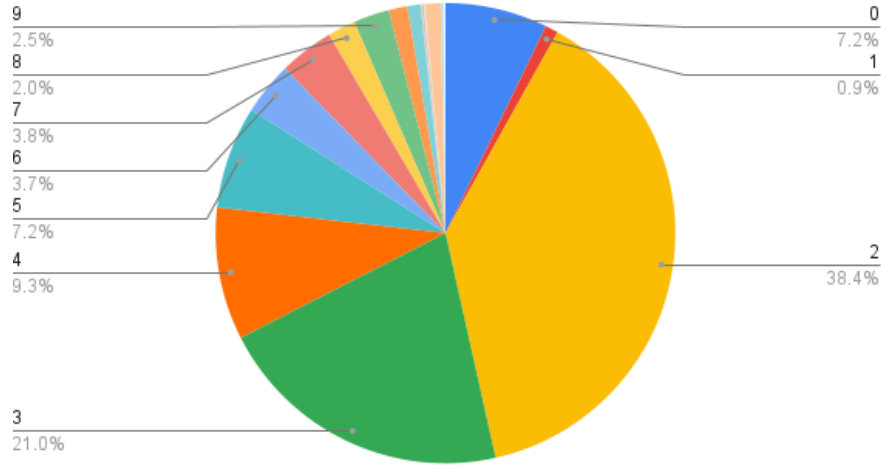
We evaluate the advantages of assessing the satisfiability of a query before its evaluation. We perform experiments with the most efficient, i.e. the bottom up algorithm.

### 8.4.1 Data

The evaluation relied on the of the GTFS-Madrid-Bench benchmark [10], a virtual knowledge graph access benchmark aimed at assessing the performance of Ontology-based Data Access (OBDA) engines. More specifically, we use a specialisation of the benchmark [6] for façade-based data access engines.

The benchmark comprises a set of 18 SPARQL queries that interrogate a collection of data sources. Table 24 (see Appendix B) reports the characteristics of the queries of the benchmark in terms of the number of BGPs, the number of variables per BGP, and the number of triple patterns per BGP.

Furthermore, the benchmark offers a data generator that can increase the size of the original data and distribute the datasets across various formats.



**Figure 4** Dimension of BGPs from real-world queries: number of variables.

In this experiment, we use JSON to represent hierarchical data formats and CSV for tabular data sources. We generate datasets of sizes 1 and 10 and execute the 18 queries on them using the SPARQL Anything engine (version v1.1.0)<sup>24</sup>. We execute each query three times and record the time taken to construct and load the Façade-X RDF graph in memory for each execution. Finally, we calculate the average of these three runs. Moreover, for each query of the benchmark, we run the bottom-up algorithm for assessing the query satisfiability. Table 24 (see Appendix B) reports construction and loading time and time for assessing the satisfiability of the queries.

## 8.4.2 Results

The satisfiability of the GTFS-Madrid-Bench queries can be assessed in less than 15 ms. Excluding the first query reduces the maximum time to 2.4. In contrast, construction and loading times are at least two orders of magnitude longer (the shortest recorded time was 161.5 ms for query 6, size 1, with CSV input). This demonstrates the benefit of assessing satisfiability prior to query execution, since construction and loading time can be avoided for unsatisfiable queries, thus saving computational resources. Naturally, the larger the input, the greater the benefit: while construction and loading times increase with the size of the input, assessment of satisfiability is independent from it.

## 9 Conclusions

In this article, we studied the satisfiability of SPARQL basic graph patterns in Façade-X. We introduced a consolidated version of Façade-X including formalised mappings to RDF. We studied the conditions under which a BGPs are satisfiable as the basis for an algorithm to verify that efficiently. Extensive experiments demonstrate the feasibility of satisfiability check. Two extreme cases of large BGPs motivate future work on further improving the performance, focusing on reducing the space of potential Façade-X annotations before verifying the constraints. In the future, we will extend this study to consider specific Façade-X profiles, studying how those can

<sup>24</sup><https://github.com/SPARQL-Anything/sparql.anything/releases/tag/v1.1.0>

bring additional constraints to verify (e.g. Façade-X graphs from CSVs will never have types). Crucially, identifying solution patterns is a preliminary step for investigating classes of queries (e.g. star queries) whose solutions can be streamed directly from the file sources, without needing to load intermediate results in-memory. Finally, our work paves the way for studying query execution strategies for Façade-X data access with SPARQL and supporting developers in building more efficient data integration systems for knowledge graphs.

## References

- 1 Serge Abiteboul, Sophie Cluet, Vassilis Christophides, Tova Milo, Guido Moerkotte, and Jérôme Siméon. Querying documents in object databases. *International Journal on Digital Libraries*, 1:5–19, 1997. doi:10.1007/S007990050001.
- 2 Renzo Angles, Georg Gottlob, Aleksandar Pavlović, Reinhard Pichler, and Emanuel Sallinger. Sparqllog: A system for efficient evaluation of sparql 1.1 queries via datalog. *Proceedings of the VLDB Endowment*, 16(13):4240–4253, 2023. doi:10.14778/3625054.3625061.
- 3 Marcelo Arenas, Claudio Gutierrez, and Jorge Pérez. On the semantics of sparql. In *Semantic web information management: A model-based perspective*, pages 281–307. Springer, 2009. doi:10.1007/978-3-642-04329-1\_13.
- 4 Julián Arenas-Guerrero, David Chaves-Fraga, Jhon Toledo, María S Pérez, and Oscar Corcho. Morph-kgc: Scalable knowledge graph materialization with mapping partitions. *Semantic Web*, 15(1):1–20, 2024. doi:10.3233/SW-223135.
- 5 Luigi Asprino and Miguel Ceriani. How is your knowledge graph used: Content-centric analysis of SPARQL query logs. In *Proceedings of the 22nd International Semantic Web Conference, Part I*, pages 197–215, 2023. doi:10.1007/978-3-031-47240-4\_11.
- 6 Luigi Asprino, Enrico Daga, Justin Dowdy, Aldo Gangemi, and Paul Mulholland. Materialisation approaches for Façade-based data access with SPARQL. *Semantic Web -- Interoperability, Usability, Applicability*, IOS Press, 2022. URL: <https://www.semantic-web-journal.net/content/materialisation-approaches-fa%C3%A7ade-based-data-access-sparql-0>.
- 7 Luigi Asprino, Enrico Daga, Justin Dowdy, Paul Mulholland, Aldo Gangemi, and Marco Ratta. Streamlining Knowledge Graph Construction with a façade: The SPARQL Anything project. In Publications Office of the European Union and European Commission. Directorate General for Informatics., editor, *ENDORSE, 2nd European Data conference On Reference data and SEMantics ENDORSE 2023: virtual event, 14 16 March 2023 : proceedings*. Publications Office, LU, 2023. doi:10.2830/343811.
- 8 Luigi Asprino, Enrico Daga, Aldo Gangemi, and Paul Mulholland. Knowledge graph construction with a façade: A unified method to access heterogeneous data sources on the web. *ACM Transactions on Internet Technology*, 23(1):1–31, 2023. doi:10.1145/3555312.
- 9 Sarra Bouhenni, Said Yahiaoui, Nadia Nouali-Taboudjemat, and Hamamache Kheddouci. A survey on distributed graph pattern matching in massive graphs. *ACM Computing Surveys (CSUR)*, 54(2):1–35, 2021. doi:10.1145/3439724.
- 10 David Chaves-Fraga, Freddy Priyatna, Andrea Cimmino, Jhon Toledo, Edna Ruckhaus, and Oscar Corcho. GTFS-Madrid-Bench: A benchmark for virtual knowledge graph access in the transport domain. *Journal of Web Semantics*, 65:100596, 2020. doi:10.1016/J.WEBSEM.2020.100596.
- 11 David Chaves-Fraga, Edna Ruckhaus, Freddy Priyatna, Maria-Esther Vidal, and Oscar Corcho. Enhancing virtual ontology based access over tabular data with morph-csv. *Semantic Web*, 12(6):869–902, 2021. doi:10.3233/SW-210432.
- 12 Chris Clifton, Hector Garcia-Molina, and David Bloom. Hyperfile: A data and query model for documents. *The VLDB Journal*, 4:45–86, 1995. URL: <http://www.vldb.org/journal/VLDBJ4/P045.pdf>.
- 13 Richard Cyganiak. Tarql (sparql for tables): Turn csv into rdf using sparql syntax. Technical report, Technical Report, 2015. Available at: <http://tarql.github.io>, 2015.
- 14 Enrico Daga. Sparql-anything/facade-x-queries-on-github: v1, January 2026. doi:10.5281/zenodo.18133036.
- 15 Enrico Daga. Supplementary material - sparql-anything/fxbgp: v4, January 2026. doi:10.5281/zenodo.18196574.
- 16 Enrico Daga, Luigi Asprino, Paul Mulholland, and Aldo Gangemi. Façade-X: an opinionated approach to SPARQL anything. *Studies on the Semantic Web*, 53:58–73, 2021.
- 17 Enrico Daga, Luca Panziera, and Carlos Pedrinaci. A basilar approach for building web apis on top of sparql endpoints. In *CEUR Workshop Proceedings*, volume 1359, pages 22–32, 2015. URL: <https://ceur-ws.org/Vol-1359/paper4.pdf>.
- 18 Alin Deutsch, Nadime Francis, Alastair Green, Keith Hare, Bei Li, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Wim Martens, Jan Michels, et al. Graph pattern matching in gql and sql/pgq. In *Proceedings of the 2022 International Conference on Management of Data*, pages 2246–2258, 2022.
- 19 George HL Fletcher. An algebra for basic graph patterns. In *Proceedings of the Workshop on Logic in Databases*, pages 38–46. Citeseer, 2008.
- 20 Brian Gallagher. Matching structure and semantics: A survey on graph-based pattern matching. In *AAAI Fall Symposium: Capturing and Using Patterns for Evidence Detection*, volume 45, pages 43–53, 2006.

- 21 Herminio García-González, Iovka Boneva, Sławek Staworko, José Emilio Labra-Gayo, and Juan Manuel Cueva Lovelle. Shexml: improving the usability of heterogeneous data mapping languages for first-time users. *PeerJ Computer Science*, 6:e318, 2020. doi:10.7717/PEERJ-CS.318.
- 22 Ramanathan Guha and Dan Brickley. RDF schema 1.1. W3C recommendation, W3C, February 2014. <https://www.w3.org/TR/2014/REC-rdf-schema-20140225/>.
- 23 Steven Harris and Andy Seaborne. SPARQL 1.1 query language. W3C recommendation, W3C, March 2013. <https://www.w3.org/TR/2013/REC-sparql11-query-20130321/>.
- 24 Olaf Hartig and Ralf Heese. The sparql query graph model for query optimization. In *European Semantic Web Conference*, pages 564–578. Springer, 2007. doi:10.1007/978-3-540-72667-8\_40.
- 25 Pieter Heyvaert, Ben De Meester, Anastasia Dimou, and Ruben Verborgh. Declarative rules for linked data generation at your fingertips! In *European Semantic Web Conference*, pages 213–217. Springer, 2018. doi:10.1007/978-3-319-98192-5\_40.
- 26 Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d’Amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al. Knowledge graphs. *ACM Computing Surveys (Csur)*, 54(4):1–37, 2021. doi:10.1145/3447772.
- 27 Enrique Iglesias, Samaneh Jozashoori, David Chaves-Fraga, Diego Collarana, and Maria-Esther Vidal. Sdm-rdfizer: An rml interpreter for the efficient creation of rdf knowledge graphs. In *Proceedings of the 29th ACM international conference on Information & Knowledge Management*, pages 3039–3046, 2020. doi:10.1145/3340531.3412881.
- 28 Enrique Iglesias, Samaneh Jozashoori, and Maria-Esther Vidal. Scaling up knowledge graph creation to large and heterogeneous data sources. *Journal of Web Semantics*, 75:100755, 2023. doi:10.1016/J.WEBSEM.2022.100755.
- 29 Ana Iglesias-Molina, Dylan Van Assche, Julián Arenas-Guerrero, Ben De Meester, Christophe Debruyne, Samaneh Jozashoori, Pano Maria, Franck Michel, David Chaves-Fraga, and Anastasia Dimou. The rml ontology: a community-driven modular redesign after a decade of experience in mapping heterogeneous data to rdf. In *International Semantic Web Conference*, pages 152–175. Springer, 2023. doi:10.1007/978-3-031-47243-5\_9.
- 30 Kostis Kyzirakos, Ioannis Vlachopoulos, Dimitrios Savva, Stefan Manegold, and Manolis Koubarakis. GeoTriples: a Tool for Publishing Geospatial Data as RDF Graphs Using R2RML Mappings. In *TC/SSN@ ISWC*, pages 33–44, 2014. URL: <https://ceur-ws.org/Vol-1401/paper-03.pdf>.
- 31 Laks VS Lakshmanan, Ganesh Ramesh, Hui Wang, and Zheng Zhao. On testing satisfiability of tree pattern queries. In *VLDB*, volume 4, pages 120–131, 2004.
- 32 Markus Lanthaler, Richard Cyganiak, and David Wood. RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C, February 2014. <https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/>.
- 33 Maxime Lefrançois, Antoine Zimmermann, and Noorani Bakerally. A sparql extension for generating rdf from heterogeneous formats. In *Proc of ESWC*, pages 35–50. Springer, 2017. doi:10.1007/978-3-319-58068-5\_3.
- 34 Lorenzo Livi and Antonello Rizzi. The graph matching problem. *Pattern Analysis and Applications*, 16:253–283, 2013. doi:10.1007/S10044-012-0284-8.
- 35 Albert Meroño-Peñuela and Rinke Hoekstra. grlc makes github taste like linked data apis. In *The Semantic Web: ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29–June 2, 2016, Revised Selected Papers 13*, pages 342–353. Springer, 2016. doi:10.1007/978-3-319-47602-5\_48.
- 36 Franck Michel, Catherine Faron-Zucker, and Fabien Gandon. SPARQL micro-services: lightweight integration of web APIs and linked data. In *LDOW@ WWW*, 2018.
- 37 Sitt Min Oo and Olaf Hartig. An algebraic foundation for knowledge graph construction. In Edward Curry, Maribel Acosta, María Poveda-Villalón, Marieke van Erp, Adegboyega K. Ojo, Katja Hose, Cogan Shimizu, and Pasquale Lisena, editors, *The Semantic Web - 22nd European Semantic Web Conference, ESWC 2025, Portoroz, Slovenia, June 1-5, 2025, Proceedings, Part I*, volume 15718 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2025. doi:10.1007/978-3-031-94575-5\_1.
- 38 Yue Pang, Linglin Yang, Lei Zou, and M Tamer Özsu. gfov: A full-stack sparql query optimizer & plan visualizer. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 5081–5085, 2023. doi:10.1145/3583780.3614741.
- 39 Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. In *International semantic web conference*, pages 30–43. Springer, 2006. doi:10.1007/11926078\_3.
- 40 Marco Ratta, Enrico Daga, and Luigi Asprino. Pysparql anything showcase. In *European Semantic Web Conference*, pages 301–305. Springer, 2024. doi:10.1007/978-3-031-78952-6\_46.
- 41 Mariano Rodríguez-Muro and Martin Rezk. Efficient sparql-to-sql with r2rml mappings. *Journal of Web Semantics*, 33:141–169, 2015. doi:10.1016/J.WEBSEM.2015.03.001.
- 42 Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020.
- 43 Jaime Salas and Aidan Hogan. A taxonomy of basic graph pattern motifs for understanding SPARQL query logs. In Benny Kimelfeld, Maria Vanina Martinez, and Renzo Angles, editors, *Proceedings of the 15th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW 2023)*, volume 3409, 2023. URL: <https://ceur-ws.org/Vol-3409/paper15.pdf>.
- 44 Jason Slepicka, Chengye Yin, Pedro A Szekely, and Craig A Knoblock. Kr2rml: An alternative interpretation of r2rml for heterogeneous sources. In *Proc of Cold*, 2015.



- 45 Jean-Marc Steyaert and Philippe Flajolet. Patterns and pattern-matching in trees: An analysis. *Information and Control*, 58(1):19–58, 1983. doi:10.1016/S0019-9958(83)80056-4.
- 46 Markus Stocker, Andy Seaborne, Abraham Bernstein, Christoph Kiefer, and Dave Reynolds. Sparql basic graph pattern optimization using selectivity estimation. In *Proceedings of the 17th international conference on World Wide Web*, pages 595–604, 2008. doi:10.1145/1367497.1367578.
- 47 Dylan Van Assche, Thomas Delva, Gerald Haesendonck, Pieter Heyvaert, Ben De Meester, and Anastasia Dimou. Declarative rdf graph generation from heterogeneous (semi-) structured data: A systematic literature review. *Journal of Web Semantics*, 75:100753, 2023. doi:10.1016/J.WEBSEM.2022.100753.
- 48 Anne-Marie Vercoustre and François Paradis. A descriptive language for information object reuse through virtual documents. In Maria E. Orłowska and Roberto Zicari, editors, *OOSI'97*, pages 299–311, London, 1998. Springer London.
- 49 Maria-Esther Vidal, Edna Ruckhaus, Tomas Lampo, Amadís Martínez, Javier Sierra, and Axel Polleres. Efficiently joining group patterns in sparql queries. In *Extended Semantic Web Conference*, pages 228–242. Springer, 2010. doi:10.1007/978-3-642-13486-9\_16.
- 50 Junchi Yan, Xu-Cheng Yin, Weiyao Lin, Cheng Deng, Hongyuan Zha, and Xiaokang Yang. A short survey of recent advances in graph matching. In *Proceedings of the 2016 ACM on international conference on multimedia retrieval*, pages 167–174, 2016. doi:10.1145/2911996.2912035.
- 51 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, volume 81, pages 82–94, 1981.
- 52 Xiaowang Zhang, Jan Van den Bussche, and François Picalausa. On the satisfiability problem for sparql patterns. *Journal of Artificial Intelligence Research*, 56:403–428, 2016. doi:10.1613/JAIR.5028.
- 53 Antoine Zimmermann. RDF 1.1: On semantics of RDF datasets. W3C note, W3C, February 2014. <https://www.w3.org/TR/2014/NOTE-rdf11-datasets-20140225/>.

## A Experiments with real-world queries

We report on experiments with real world SPARQL queries tailored to facade based data access. The queries were collected from the GitHub API on 18 December 2025 and include all files published in public repositories with extensions “.rq” and “.sparql”, which include the string “x-sparql-anything” in the text. The code and data are published<sup>25</sup> and archived [14]. The analysis was performed on Google drive and can be viewed at <https://docs.google.com/spreadsheets/d/11n7c101WLQBL8aApZs-fXNtHAYTIBIP9nZUDxD6TUa8/edit>.

We perform experiments for both satisfiability and to generate all possible solution patterns on 449 collected queries. We extract 1430 BGPs from the 449 queries. Table 18 shows a summary of the queries grouped by number of BGPs. Of these queries, 58 cannot be parsed because they include syntax errors. 228 of the remaining queries have a single BGP. The majority of queries have less than five basic graph patterns. Table 19 summarizes the BGPs extracted from the queries, to which we apply our satisfiability problem. 103 BGPs with 0 triples refer to BGPs that included SPARQL Anything configuration options, that we exclude. Figure 5 shows the number of BGPs by the number of variables included. Clearly, the more the variables, the more complex it is to verify satisfiability, since variables are annotated with more potential annotations. The majority of BGPs have between 1 and 5 variables. In what follows, we show the average values of 10 executions.

### A.1 Satisfiability check

We perform the same experiment setting as with the designed queries (Section 8), focusing on the most performing bottom-up algorithm only, with the 449 real world queries.

Table 20 shows the average duration of the more efficient bottom-up satisfiability check algorithm presented in Section 7. Data is grouped by BGPs with the same number of triples. BGPs with 0 triples are the ones that only included triples with SPARQL Anything configuration,

<sup>25</sup><https://github.com/SPARQL-Anything/facade-x-queries-on-github/>



■ **Table 18** Number of queries by BGPs.

| Queries | BGPs |
|---------|------|
| 1       | 40   |
| 1       | 31   |
| 1       | 29   |
| 2       | 27   |
| 2       | 26   |
| 1       | 25   |
| 1       | 24   |
| 2       | 22   |
| 1       | 20   |
| 1       | 17   |
| 2       | 13   |
| 1       | 12   |
| 11      | 11   |
| 9       | 10   |
| 4       | 9    |
| 1       | 8    |
| 6       | 7    |
| 4       | 6    |
| 22      | 5    |
| 32      | 4    |
| 37      | 3    |
| 79      | 2    |
| 228     | 1    |

and can be ignored. Table 21 shows the average duration grouped by BGPs with the same number of variables. We can see how for BGPs with up to 16 triples and 18 variables (except one with 15 variables), satisfiability can be verified in less than 100 ms. The only problematic BGP is one with 19 triples and 19 variables (“query-77.rq” in the supplemental material), which cannot be verified in less than 5 seconds. We ignore errors from two other BGPs, one within a DBPedia service clause (“query-66.rq”), and another coming from a negative test within a SPARQL Anything code base (“query-199.rq”).

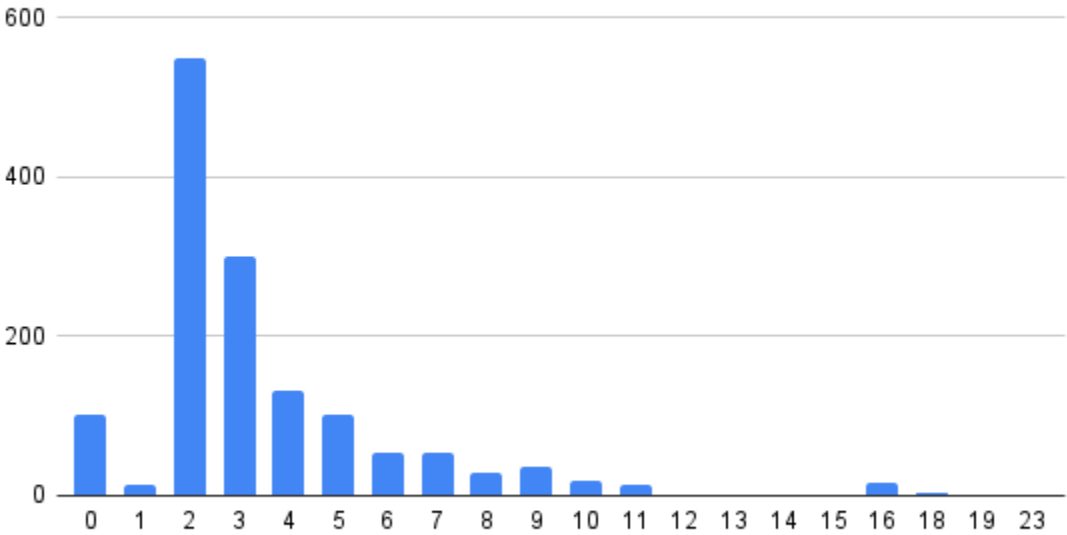
## A.2 Generating all solutions

We perform the same experiment setting as with the designed queries (Section 8), focusing on the most performing bottom-up algorithm only, with the 449 real world queries.

Table 22 shows the average duration of the more efficient bottom-up algorithm presented in Section 7 to generate all possible solution patterns. Data is grouped by BGPs with the same number of triples. In addition, Table 23 shows the average duration grouped by BGPs with the same number of variables. Solutions can be generated efficiently in all but two cases (below half second). The only two problematic BGPs are one with 19 triples and 19 variables (“query-77.rq” in the supplemental material), and another with 20 triples and 23 variables (“query-244.rq” in the supplemental material), for which it is not possible to generate all solution patterns in less than 5 seconds.

■ **Table 19** Basic graph patterns grouped by number of triples (that are not SPARQL Anything configuration options).

| Number of triples | Number of BGPs |
|-------------------|----------------|
| 0                 | 103            |
| 1                 | 591            |
| 2                 | 220            |
| 3                 | 183            |
| 4                 | 73             |
| 5                 | 36             |
| 6                 | 82             |
| 7                 | 34             |
| 8                 | 30             |
| 9                 | 41             |
| 10                | 4              |
| 11                | 1              |
| 12                | 9              |
| 13                | 1              |
| 14                | 15             |
| 16                | 3              |
| 18                | 1              |
| 19                | 1              |
| 20                | 1              |



■ **Figure 5** BGPs by number of variables.

■ **Table 20** Performance of the bottom-up satisfiability check algorithm on real world queries. We group the data by size of BGPs (number of triples). There is only 1 BGP with 11,13,18,19, and 20 triples. One query with 19 triples cannot be evaluated in less the 5 seconds (our limit).

| Triples | AVG    | STD  |
|---------|--------|------|
| 1       | 0.10   | 0.3  |
| 2       | 0.11   | 0.37 |
| 3       | 0.14   | 0.27 |
| 4       | 0.14   | 0.36 |
| 5       | 0.41   | 0.29 |
| 6       | 0.28   | 0.17 |
| 7       | 0.48   | 0.15 |
| 8       | 1.36   | 0.67 |
| 9       | 0.48   | 0.25 |
| 10      | 1.15   | 0.58 |
| 11      | 0.20   | –    |
| 12      | 6.73   | 1.81 |
| 13      | 8.30   | –    |
| 14      | 0.33   | 0.07 |
| 16      | 0.43   | 0.02 |
| 18      | 218.30 | –    |
| 19      | >5s    | –    |
| 20      | 8.20   | –    |

■ **Table 21** Performance of the bottom-up satisfiability check algorithm on real world queries. We group the data by the size of BGPs - number of variables. There are only single BGPs with 12,14,15,19, and 23 variables. One query with 19 variables cannot be evaluated in less the 5 seconds (our limit).

| Variables | AVG    | STD  |
|-----------|--------|------|
| 1         | 0.05   | 0.18 |
| 2         | 0.11   | 0.31 |
| 3         | 0.11   | 0.26 |
| 4         | 0.15   | 0.42 |
| 5         | 0.25   | 0.22 |
| 6         | 0.27   | 0.41 |
| 7         | 0.37   | 0.13 |
| 8         | 0.22   | 0.18 |
| 9         | 1.65   | 0.69 |
| 10        | 0.42   | 0.26 |
| 11        | 4.09   | 1.61 |
| 12        | 0.20   | –    |
| 13        | 4.30   | 0.77 |
| 14        | 0.20   | –    |
| 15        | 218.30 | –    |
| 16        | 0.33   | 0.07 |
| 18        | 0.43   | 0.02 |
| 19        | >5s    | –    |
| 23        | 8.20   | –    |

■ **Table 22** Performance of the bottom-up algorithm on real world queries, when generating all solution patterns. We group the data by size of BGPs (number of triples). There is only 1 BGP with 11,13,18,19, and 20 triples. For two queries with 19 and 20 triples, the complete set of solution patterns cannot be evaluated in less the 5 seconds (our limit).

| BGP Size (number of triples) | AVG      | STD   |
|------------------------------|----------|-------|
| 1                            | 0.05     | 0.37  |
| 2                            | 0.08     | 0.19  |
| 3                            | 0.13     | 0.25  |
| 4                            | 0.23     | 0.16  |
| 5                            | 8.05     | 6.27  |
| 6                            | 1.36     | 0.82  |
| 7                            | 2.23     | 1.01  |
| 8                            | 7.79     | 1.75  |
| 9                            | 5.51     | 1.19  |
| 10                           | 38.58    | 3.21  |
| 11                           | 21.50    | –     |
| 12                           | 58.10    | 20.53 |
| 13                           | 163.70   | –     |
| 14                           | 191.97   | 12.58 |
| 16                           | 1,250.00 | 40.42 |
| 18                           | 1,127.40 | –     |
| 19                           | >5s      | –     |
| 20                           | >5s      | –     |

■ **Table 23** Performance of the bottom-up algorithm on real world queries. We group the data by the size of BGPs – number of variables. There are only single BGPs with 12,14,15,19, and 23 variables. For two queries with 19 and 23 variables the set of solution patterns cannot be generated in less the 5 seconds (our limit).

| BGP Size (number of variables) | AVG      | STD   |
|--------------------------------|----------|-------|
| 1                              | 0.02     | 0.14  |
| 2                              | 0.04     | 0.36  |
| 3                              | 0.08     | 0.29  |
| 4                              | 0.14     | 0.32  |
| 5                              | 0.24     | 0.14  |
| 6                              | 0.46     | 0.16  |
| 7                              | 1.21     | 0.58  |
| 8                              | 1.19     | 0.23  |
| 9                              | 8.47     | 2.01  |
| 10                             | 25.58    | 8.36  |
| 11                             | 21.15    | 3.1   |
| 12                             | 21.50    | –     |
| 13                             | 256.55   | 35.91 |
| 14                             | 67.40    | –     |
| 15                             | 1,127.40 | –     |
| 16                             | 191.97   | 12.58 |
| 18                             | 1,250.00 | 40.42 |
| 19                             | -1.00    | –     |
| 23                             | -1.00    | –     |

## B Experiments with benchmark queries

Table 24 describes the characteristics of the queries of the GTFS-Madrid-Bench benchmark in terms of number of BGPs for each query, number of variables and number of triple patterns per BGP.

■ **Table 24** Characteristics of the queries contained in the GTFS-Madrid-Bench benchmark.

| Query ID | Number of BGPs | Number of variables per BGP | Number of Triple Patterns per BGP |
|----------|----------------|-----------------------------|-----------------------------------|
| 1        | 1              | 5                           | 4                                 |
| 2        | 4              | 2,2,2,3                     | 1,1,1,2                           |
| 3        | 5              | 2,2,2,3,2                   | 1,1,1,2,1                         |
| 4        | 7              | 2,2,2,2,2,4,2               | 1,1,1,1,1,3,1                     |
| 5        | 1              | 4                           | 3                                 |
| 6        | 1              | 3                           | 2                                 |
| 7        | 10             | 2,2,2,3,2,2,3,2,2,3         | 1,1,1,2,1,1,2,2,1,2               |
| 8        | 11             | 2,2,2,3,2,2,4,2,2,2,3       | 1,1,1,2,1,1,3,1,1,1,2             |
| 9        | 3              | 5,2,4                       | 4,1,3                             |
| 10       | 2              | 2,3                         | 1,2                               |
| 11       | 4              | 3,4,1,2                     | 2,3,2,1                           |
| 12       | 4              | 3,3,3,3                     | 2,2,2,3                           |
| 13       | 3              | 3,2,4                       | 2,1,3                             |
| 14       | 4              | 5,3,2,2                     | 4,2,1,1                           |
| 15       | 1              | 4                           | 2                                 |
| 16       | 2              | 4,3                         | 3,3                               |
| 17       | 4              | 2,2,4,4                     | 1,1,3,3                           |
| 18       | 5              | 2,4,2,2,2                   | 2,3,1,1,1                         |

Table 25 shows how long it takes to construct and load the RDF graph required to answer each query into memory. Additionally, the time required to compute the satisfiability of all BGPs in each query is reported.

■ **Table 25** Construction Loading time and time for assessing the satisfiability of the queries in the GTFS-Madrid-Bench benchmark.

| Query | Construction + Loading time (ms) |         |        |         | Time for computing the satisfiability of all BGPs (ms) |
|-------|----------------------------------|---------|--------|---------|--------------------------------------------------------|
|       | CSV                              |         | JSON   |         |                                                        |
|       | Size 1                           | Size 10 | Size 1 | Size 10 |                                                        |
| 1     | 1049.5                           | 7476.5  | 1221.5 | 6252    | 14.40                                                  |
| 2     | 310.5                            | 607.5   | 284    | 603.5   | 0.80                                                   |
| 3     | 402.5                            | 595     | 301    | 673.5   | 0.70                                                   |
| 4     | 283                              | 251     | 218    | 241.5   | 1.70                                                   |
| 5     | 166.5                            | 230     | 215.5  | 222     | 0.20                                                   |
| 6     | 161.5                            | 177     | 166    | 218     | 0.10                                                   |
| 7     | 390                              | 1031.5  | 390    | 1151.5  | 2.40                                                   |
| 8     | 449                              | 1598.5  | 410    | 1627    | 1.10                                                   |
| 9     | 1390                             | 6078.5  | 1260   | 5609.5  | 0.70                                                   |
| 10    | 292                              | 656.5   | 292    | 609.5   | 0.90                                                   |
| 11    | 240.5                            | 331.5   | 218.5  | 306     | 1.80                                                   |
| 12    | 379.5                            | 851     | 305    | 841     | 1.70                                                   |
| 13    | 275.5                            | 566.5   | 246.5  | 515     | 1.60                                                   |
| 14    | 327                              | 1107    | 348.5  | 1173    | 1.20                                                   |
| 15    | 270                              | 1340    | 289    | 1291    | 0.40                                                   |
| 16    | 202.5                            | 299.5   | 213    | 294.5   | 0.40                                                   |
| 17    | 233                              | 438     | 264.5  | 422.5   | 0.80                                                   |
| 18    | 214                              | 309     | 237    | 298     | 0.80                                                   |

# OM4OV: Leveraging Ontology Matching for Ontology Versioning

Zhangcheng Qiang\*  

Australian National University, Canberra, Australia

Kerry Taylor  

Australian National University, Canberra, Australia

Weiqing Wang  

Monash University, Melbourne, Australia

---

## Abstract

Due to the dynamics of the Semantic Web, version control is necessary to manage changes in widely used ontologies. Despite the long-standing recognition of ontology versioning (OV) as a crucial component of efficient ontology management, many approaches treat OV as similar to ontology matching (OM) and directly reuse OM systems for OV tasks. In this study, we systematically analyse similarities and differences between OM and OV and formalise an OM4OV framework to offer more advanced OV support. The framework is implemented and evalu-

ated in the state-of-the-art OM system Agent-OM. The experimental results indicate that OM systems can be effectively reused for OV tasks, but without the necessary extensions, can produce skewed measurements, poor performance in detecting update entities, and limited explanations of false mappings. To tackle these issues, we propose an optimisation method called the cross-reference (CR) mechanism, which builds on existing OM alignments to reduce the number of matching candidates and to improve overall OV performance.

**2012 ACM Subject Classification** Information systems → Information integration

**Keywords and phrases** ontology matching, ontology versioning

**Digital Object Identifier** 10.4230/TGDK.4.2.6

**Category** Research

**Related Version** *Full Version*: <https://arxiv.org/abs/2409.20302>

**Supplementary Material** For more information on Supplementary Materials and Additional Resources, including all third-party resources used, we refer to the list at the end of this article.

*Software (OM4OV and Agent-OV)*: <https://github.com/qzc438/ontology-versioning>

*Software (Brick Schema Version Track)*: <https://github.com/qzc438/brick-version-track>

**Acknowledgements** The authors thank reviewers for their comments that improved the manuscript. The authors thank Alice Richardson of the Statistical Support Network (Australian National University) for helpful advice on the statistical analysis in this paper. The authors thank Gabe Fierro at Colorado School of Mines for providing useful insights on Brick Schema version changes. The authors also thank the Commonwealth Scientific and Industrial Research Organisation (CSIRO) for supporting this project.

**Received** 2025-07-02 **Accepted** 2026-06-22 **Published** 2026-09-03

**Editors** Stefania Dumbrava, George Fletcher, Matteo Lissandrini, and Riccardo Tommasini

**Special Issue** Data Management for (Knowledge) Graphs

**Generative AI Declaration** During the preparation of this work, no Generative AI was used to produce the manuscript. However, Generative AI is intrinsic to the research reported here and is properly identified throughout the work.

---

\* Corresponding author



© Zhangcheng Qiang, Kerry Taylor, and Weiqing Wang;  
licensed under Creative Commons License CC-BY 4.0

Transactions on Graph Data and Knowledge, Vol. 4, Issue 2, Article No. 6, pp. 6:1–6:19



Transactions on Graph Data and Knowledge  
TDGDK Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



## 1 Introduction

Ontologies serve as the backbone of the Semantic Web, providing formal concept descriptions of shared concepts across various applications [14]. An ontology is not static, and the need for version control arises. While web data is dynamic, any ontology in use must undergo periodic revisions to keep pace with the growth in domain knowledge, modifications to the application adaptation, or corrections to the shared conceptualisation [20]. For example, it is unrealistic to expect ontologies created in the 1990s to contain concepts such as “touchscreen”, “fingerprint sensor”, or “WiFi antenna” [15]. These modifications may cause undesirable deficiencies in artifacts that conform to or that reuse the ontology being changed, leading to severe non-compliance and incompatibility issues in downstream tasks.

Ontology versioning (OV), also known as ontology evolution, aims to distinguish and recognise changes between different ontology versions. This enables data that applies the ontology, ontologies that reuse the ontology, and software that uses the ontology to correctly respond to the ontology version changes [20]. Although OV plays an important role in ontology management, this area remains underexplored. Many studies [30, 29, 31, 8, 13, 41, 22, 23, 24, 34] treat OV as similar to ontology matching (OM) and assume OM systems can be reused for OV tasks. OM is a well-studied problem, so leveraging this deep research legacy for OV should be beneficial. We call this approach “OM4OV”.

We analyse the complementary relationship between OM and OV tasks and provide a task formulation for the OM4OV framework. We implement the framework in our Agent-OM [37] system to validate its effectiveness in unifying the OM and OV tasks. We investigate the links between these two tasks in practice and explore optimisations for OM4OV. To the best of our knowledge, this work is the first to systematically analyse the OM4OV framework and to provide theoretical foundations and empirical evidence for its effectiveness and pitfalls. Specifically, our key contributions include:

- We analyse the similarities and differences between OM and OV. In the OM4OV framework, we describe four disjoint subset alignments produced in the OV process: *remain*, *update*, *add*, and *delete*, each corresponding to a subset of match or non-match in OM. We provide formal definitions and formulations.
- We extend Agent-OM with the OM4OV framework and call the extended system Agent-OV. We evaluate the performance of OV on the testbed derived from the Ontology Alignment Evaluation Initiative (OAEI) [32] datasets. We find that OM systems can be reused for OV tasks, but extensions are necessary to improve both overall OV performance and its meaningful quality measurement.
- We propose an optimised OM4OV framework. The new framework introduces a cross-reference mechanism, which builds on prior OM alignments to overcome the pitfalls of the naive OM4OV framework by reducing the number of matching candidates and filtering out ambiguous false mappings, thereby significantly improving overall OV performance.

In this paper, we assume the OWL 2 Web Ontology Language [5] is used to represent an ontology. We use both *concept* and *entity* to refer to OWL classes or properties. We use *subsumption* to mean `rdfs:subClassOf` or `rdfs:subPropertyOf` and *equivalence* to mean `owl:equivalentClass` or `owl:equivalentProperty` as formal class and property inter-relations.

The remainder of the paper is organised as follows. Section 2 reviews the related work. Section 3 provides the task formulation for the OM4OV framework, while Section 4 evaluates the OM4OV framework and discusses several common pitfalls for improvement. Section 5 proposes a cross-reference mechanism as a framework optimisation method. Section 6 discusses the use of OM4OV in real-world applications. Section 7 concludes the paper.

## 2 Related Work

Version control is recognised as a vital element in ontology management. Different versions need to support interoperability so that version changes do not impede the effective and sustainable use of the ontology.

One manual approach is to extend the ontology itself with internal version information. The Simple HTML Ontology Extensions [17] uses the tag `BACKWARD-COMPATIBLE-WITH` to record version information. The authors of [21] argue that a carefully-managed version numbering system embedded in the URI of the ontology (and therefore the fully expanded name of entities defined in the ontology) can minimise the impact of adopting updated versions because unchanged entities will be unaffected in practice. These approaches have been largely adopted by the later OWL languages, where a set of annotation properties related to version information is defined. These include `owl:versionInfo` and `owl:priorVersion` to describe the version number of the ontology, `owl:backwardCompatibleWith` and `owl:incompatibleWith` to specify an entity's compatible or incompatible corresponding entity in the previous version, and `owl:DeprecatedClass` and `owl:DeprecatedProperty` to declare deprecated entities. Later, the ontology language  $\tau$ OWL [42] was introduced to extend the OWL triple schema to a 4-tuple schema to represent the versioning of concepts within an ontology. This idea is now incorporated in the new proposals for RDF 1.2 Schema [38], which allow time-varying information to be deduced from a temporal dimension within the 4-tuple. An alternative option is to create a separate version log to track changes in versions. Unlike earlier approaches that use an unstructured plain text file, the authors of [35] propose a novel approach that uses a version ontology with a change definition language to create a version log. In [7], the authors construct an historical knowledge graph. Storing the version log in a knowledge graph not only avoids repetition, but also enables advanced search functions. The authors of [39] argue that version logs may contain redundancy and inconsistent information. They propose a graph-of-relevance approach to interlink different version logs and remove less relevant versions.

Maintaining version information in the ontology can be labour-intensive and error-prone. Either extending the current standard 3-tuple schema or using change logs requires consistent updating over time. In most cases, this process is hand-crafted by the ontology engineer or requires human intervention (e.g. pre-defining a schema or creating a template for the change log). A manual process is more likely to make mistakes and fail to propagate changes to dependent artifacts. Moreover, an ontology may lack or have incomplete version information. In such cases, current approaches have limited capability to detect incorrect or missing version information. These approaches rely on the version information contained in or attached to the ontology by manual methods. If such information is missing or incorrect, automated versioning of ontology concepts becomes necessary.

Reusing OM systems for OV tasks (OM4OV) is a lightweight and fully automatic approach to detecting changes in ontologies. OM4OV is compatible with ontologies that use different recording methods, including URIs or additional versioning triples, as well as ontologies that lack internal version information. The PromptDiff suite [30, 29, 31] is the canonical work that integrates different heuristic matchers and applies a matching-like comparison for OV. The concept of “matching” is later used in the H-Change methodology [8] to assess the assimilation of new concepts into the old ontology. Several studies [13, 41] note that updates to an ontology will generally create a requirement to update its existing alignments to external ontologies. Methods [22, 23, 24] are proposed to identify changes that affect alignments, and in some cases the framework is proposed to automate alignment updates [34], but these works do not address the discovery of alignments from an ontology to its own updated version.

While existing OM4OV approaches directly use OM systems for OV and focus on developing new algorithms and architectures for ontology version control, they pay less attention to understanding the underlying differences between OM and OV. OM is not exactly the same as OV. OM determines *matched* entities between two different ontologies, while OV can distinguish *add*, *delete*, *remain*, and *update* changes over different versions of one ontology. The OM input consists of two distinct ontologies, whereas the OV input is expected to consist of two versions of a single ontology. The output of OM is a set of entity mappings, whereas the output of OV comprises four sets, each defined by the nature of the change made. These differences may affect the adaptability of reusing OM systems for OV tasks.

### 3 Task Formulation

We now draw on concepts in OM to formalise OV tasks. Given an ontology  $O$ , we say that some entity  $e \in O$  when  $e$  is of interest to ontology matching or ontology versioning, typically representing a concept of the domain of interest being modelled by  $O$ . We consider an entity to be the full URI of the reference in an ontology, or its equivalent abbreviation via a prefix, for example `cmt:ProgramCommitteeChair`. We consider an *entity name* to be the substring of an entity that provides a meaningful description of the entity (e.g. *ProgramCommitteeChair*), excluding any entity prefix or expanded URI (e.g. `cmt:ProgramCommitteeChair` is not an entity name).

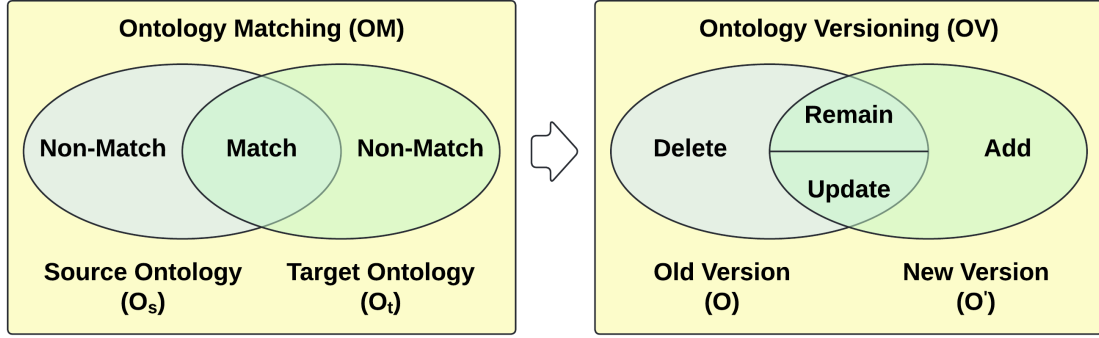
#### 3.1 OM Preliminaries

OM identifies correspondences between entities of two ontologies [11]. Formally, given a source ontology ( $O_s$ ) and a target ontology ( $O_t$ ), the OM task is to find an alignment  $A$  (i.e. a set of mappings) with respect to a given similarity threshold  $s \in [0, 1]$ , defined as [10]:  $A = \{(e_1, e_2, r, c) \mid e_1 \in O_s, e_2 \in O_t, s \leq c \leq 1\}$ , where  $e_1$  and  $e_2$  are ontology entities in  $O_s$  and  $O_t$  respectively,  $r$  is the relation between  $e_1$  and  $e_2$  that may be equivalence ( $\equiv$ ) or subsumption ( $\subseteq$ ), and  $c$  is the confidence. For example, `(cmt:ProgramCommitteeChair, confof:Chair_PC,  $\equiv$ , 0.90)` means `cmt:ProgramCommitteeChair` and `confof:Chair_PC` are equivalent with a confidence of 0.90.

An entity match is asserted by a matching system according to its own internal algorithm, its own interpretation of “match”, and its own assessment of confidence. In general for matching, a source entity may occur in multiple mappings that vary with respect to the target entity, the relation, and the confidence. In this study, we consider only alignments made by matching systems that are comprised of one-to-one equivalence mappings, admitting only matches with the highest  $c$  as determined by the matching system and requiring the matching system to resolve tied confidence scores. Typically, equivalence relations are determined by lexical or embedding similarity in matching systems, although logical entailment is equally admissible in our study. OM performance is measured by comparing the matching system alignment ( $A$ ) with a reference alignment ( $R$ ) using precision, recall, and F1 score (see Section 4.3 for details).

#### 3.2 OM4OV Formalisation

OV identifies correspondences between two versions of a single ontology [20]. Given an old version of an ontology ( $O$ ) and a new version of the same ontology ( $O'$ ), the OV task can be considered as finding an alignment ( $A$ ) from the old ontology to the new one. As illustrated in Figure 1, while an alignment for OM only mentions matched entities, OV needs to recognise both matched and non-matched entities. Furthermore, OV matched entities are comprised of two subsets: *remain* and *update*, while non-matched entities are comprised of *add* and *delete* entities. While some of these sets are comprised of entities, and others of mappings, when convenient henceforward, we will loosely refer to all of these as either alignments or sets of entities interchangeably.



■ **Figure 1** Overview of the OM4OV framework. The sets are comprised of entities, with the matched intersection containing entities that are considered equivalent by an ontology alignment.

An OV task can be formalised as finding an alignment and a non-alignment,  $A_{match}$  and  $A_{non-match}$  respectively, between different ontology versions  $O$  and  $O'$  with respect to a given similarity threshold  $s \in [0, 1]$ . OV focuses only on the equivalence relation; subsumption mappings are beyond the scope of versioning.

► **Definition 1.** Given ontologies  $O$  and  $O'$  and similarity threshold  $s \in [0, 1]$ , the OV task is to find  $A_{match}$  and  $A_{non-match}$  as follows:

$$A_{match} = \{(e_1, e_2, c) \mid e_1 \in O, e_2 \in O', s \leq c \leq 1 \text{ and there is no } (e_1, e_i, c') \text{ with } e_i \neq e_2 \text{ or with } c \neq c' \text{ and there is no } (e_j, e_2, c') \text{ with } e_j \neq e_1 \text{ or with } c \neq c'\} \quad (1)$$

$$A_{non-match} = \{e \mid e \in O \cup O' \text{ and there is no } (e_1, e_2, c) \in A_{match} \text{ with } e \in \{e_1, e_2\}\}$$

We say that  $e_1$  and  $e_2$  are equivalent whenever there is a  $c$  satisfying  $(e_1, e_2, c) \in A_{match}$ .

For convenience, we will sometimes refer to the elements of  $A_{match}$  as *entities* rather than *mappings*, by which we mean entities that are mentioned in mappings in  $A_{match}$ .

The *update* and *remain* entities are entities successfully matched over different ontology versions. The *update* entities are, in general, the closest near-equivalent matches of entities from different ontology versions. We recommend that matching systems assign lower confidence to entity matches when either (a) structural changes have been made near the entity, such as changes to class membership or property restrictions, or (b) the entity's name has changed.

We recommend that all other successful matches not satisfying either (a) or (b) are assigned very high confidence by matching systems, most especially when a so-called *trivial* match is found. These matches are then interpreted as *remain* entities for OV. In particular, we say that an entity is interpreted as not changed in OV for any successful match where the matching system reports a confidence of 1. Further, we recognise that some OV systems may prefer, additionally, to interpret an entity as not changed when the confidence is only slightly less than 1.

► **Definition 2.**  $A_{remain}(A \odot)$  and  $A_{update}(A \otimes)$  are defined as:<sup>1</sup>

$$\begin{aligned} A \odot &= \{(e_1, e_2, c) \mid (e_1, e_2, c) \in A_{match} \text{ and } c = 1\} \\ A \otimes &= \{(e_1, e_2, c) \mid (e_1, e_2, c) \in A_{match} \text{ and } c < 1\} \end{aligned} \quad (2)$$

<sup>1</sup> In some cases, OV systems may consider matches that are near-enough with a confidence slightly less than 1 to be perfect matches. Without loss of generality, we assume in such cases that the OV system reassigns  $c$  to be 1.

We can see that  $A_{match} = A \odot \cup A \otimes$ . By definition,  $A \odot$  and  $A \otimes$  are disjoint.

The *add* and *delete* entities are actually non-matched entities between different ontology versions. The *add* entities are the entities in  $O'$  that have no matches in  $O$ . Similarly, we can interpret the *delete* entities as entities in  $O$  that have no matches in  $O'$ .

► **Definition 3.**  $A_{add}(A\oplus)$  and  $A_{delete}(A\ominus)$  are defined as:

$$\begin{aligned} A\oplus &= \{e \mid e \in O' \text{ and } e \in A_{non-match}\} \\ A\ominus &= \{e \mid e \in O \text{ and } e \in A_{non-match}\} \end{aligned} \tag{3}$$

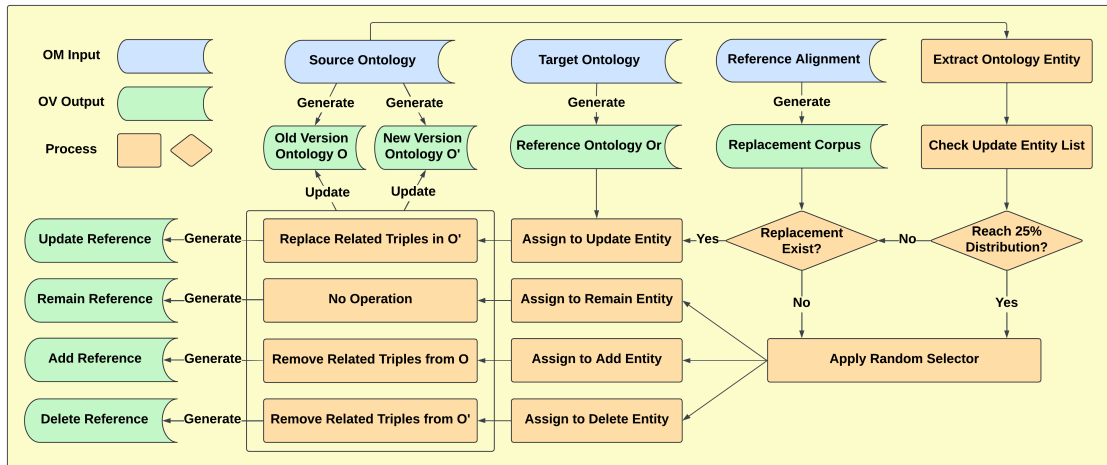
We can see that  $A_{non-match} = A \oplus \cup A \ominus$ . By definition,  $A \oplus$  and  $A \ominus$  are disjoint.

## 4 OM40V Evaluation

We now consider the evaluation of OM4OV. In the literature, OV are evaluated using OM measures, which we refine here to give better insights into OV performance.

## 4.1 Dataset Construction

There is a dearth of benchmark datasets for evaluating OV. The Ontology Alignment Evaluation Initiative (OAEI) contains several datasets related to OM tasks, but none are specifically designed for OV tasks. We propose an approach to constructing synthetic OV datasets from real-world OM datasets. Figure 2 illustrates the generation of OM4OV datasets.



■ **Figure 2** Generation of OM4OV datasets from initial OM inputs of a source ontology, a target ontology, and a reference alignment, and outputs of versioning ontologies and reference alignments with four categories. Optionally, a reference ontology and a replacement corpus may be produced.

- (1) The original OAEI datasets for OM provide two ontologies, the *source*  $O_s$  and the *target*  $O_t$ . We choose either one as the ontology for versioning and duplicate it into  $O$  and  $O'$ . Another ontology will be used later to provide references for synthetic *update* entities. We duplicate this ontology into  $O_r$ .
- (2) There are four possible entity changes in OV tasks: *remain*, *update*, *add*, and *delete*. We retrieve all ontology entities from  $O$  or  $O'$  (since they are identical at this stage). Each ontology entity will be randomly assigned to one of these categories, and the four sets are

pairwise disjoint by construction. In real-world OV, entity updates are relatively infrequent. We allow a user-defined parameter to fix the proportion of update entities in the synthetic dataset, defaulting to 25%. The update entities are randomly selected from the replacement corpus, so the number of update entities is limited by the size of the replacement corpus, which may be less than the user-defined parameter value. For this reason, selection of update entities is prioritised over other categories, and excess entities are redistributed evenly over the other three categories.

- (3) For each ontology entity  $e$  assigned to a category, the corresponding dataset operations are described as follows. While there is no operation for *remain* entities, entities assigned to *add* and *delete* are deleted from all triples that mention them in  $O$  or  $O'$  respectively. Each *update* mapping requires that all occurrences of its domain entity in triples in  $O'$  are replaced by the corresponding co-domain entity of the mapping. We generate four corresponding versioning references based on the entity assignments. These files provide the ground truth for evaluation.

To ensure the replaced entity is reasonable and reflects real-world facts, we use equivalent classes or properties specified in the reference alignment. For example, we could replace `cmt:ProgramCommitteeChair` with its equivalent class `confof:Chair_PC`. This operation is cascaded, meaning that we update not only the entity name but also its related triples, such as changing the semantic information from `(cmt:ProgramCommitteeChair, rdfs:subClassOf, cmt:ProgramCommitteeMember)` to `(confof:Chair_PC, rdfs:subClassOf, confof:Person)`. While this procedure can be unlimited, we restrict it to extract only the triples directly related to the entity (i.e. a 1-hop subgraph) in the ontology  $O_r$ . For entities whose names are unique identifiers or codes (and not textually meaningful names), we retrieve their meaningful descriptions from annotation properties. For example, we extract an additional annotation triple `(mouse:MA_0000270, rdfs:label, "eyelid tarsus")` for the entity `mouse:MA_0000270`. After extraction, we also need to update the prefix from “*confof*” to “*cmt*” where the new entity URI will be `cmt:Chair_PC` and the replacement triple will be `(cmt:Chair_PC, rdfs:subClassOf, cmt:Person)`. Finally, we add these new triples to the existing ontology  $O'$ . Note that where entity names do not change but structural changes have been made near the entity, this will be represented as an entity update. Table 1 summarises the steps to construct an *update* entity  $e$  in the benchmark dataset.

■ **Table 1** Steps to construct an *update* entity  $e$  in the benchmark dataset.

| Step | Dataset Operation                                                                                                        |
|------|--------------------------------------------------------------------------------------------------------------------------|
| #1   | Remove all the triples related to $e$ in $O'$ .                                                                          |
| #2   | Find the equivalent class/property $e_r$ in <i>replacement corpus</i> .                                                  |
| #3   | Extract all the triples related to $e_r$ in $O_r$ .<br>(Optional) Extract annotation triples related to $e_r$ in $O_r$ . |
| #4   | Update the prefix from $e_r$ to $e$ and create a new entity $e'$ .                                                       |
| #5   | Add all the triples related to $e'$ to $O'$ .                                                                            |

Unlike the original OAEI datasets used for OM, we introduce randomness to ensure that the synthetic OAEI datasets for OV are different each time they are constructed. This suits the dynamic nature of OV, where the changes vary between different versions. For this reason, we consider the new OAEI datasets for OV more like a testbed, as they can simulate a variety of situations for OV tasks. When reproducibility is required, we recommend using a fixed seed to control the randomness.

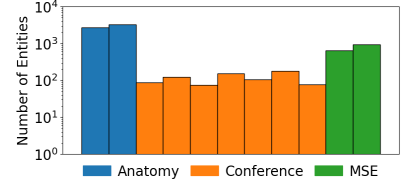


Table 2 details the OAEI tracks selected for the OV testbed. The current version of the OV testbed contains a total of 11 distinct ontologies from three different OAEI tracks. The anatomy track contains two large ontologies, while the MSE track has two medium ontologies, and the conference track has 7 small ontologies. We exclude EMMO [9] from the MSE track because it assigns opaque symbolic codes to entity names, using a naming convention that differs from MaterialInformation [3] and MatOnto [19], which use meaningful entity names. Figure 3 shows the number of entities in each ontology in each track.

■ **Table 2** Selected OAEI tracks for the OV testbed.

| Track      | Domain                            | # of Ontologies |
|------------|-----------------------------------|-----------------|
| Anatomy    | Human and Mouse Anatomy           | 2               |
| Conference | Research Conference               | 7               |
| MSE        | Materials Science and Engineering | 2               |

■ **Figure 3** Number of entities.



## 4.2 Evaluation System

We choose the matching system Agent-OM and its extended version Agent-OV to demonstrate the OM4OV framework. Agent-OM is one of the flagship OM systems powered by large language models (LLMs) and multi-agent architectures. Its foundation framework is designed for traditional OM tasks. We extend Agent-OM with the OM4OV framework so that it can be used for OV tasks. Inherited from Agent-OM, Agent-OV supports a wide range of LLMs, including commercial API-accessed LLMs OpenAI GPT [33] and Anthropic Claude [2], as well as open-source LLMs Meta Llama [28], Alibaba Qwen [1], Google Gemma [12], and ChatGLM [40]. Inherited from Agent-OM, *similarity\_threshold* and *top@k* are the adjustable key hyperparameters for Agent-OV.

As an aside, we observe that confidence scores for successful matches as determined by OM systems may be unhelpful for direct use in OV for distinguishing *remain* entities from *update* entities. This is because, in practice, the confidence boundary to best distinguish *remain* and *update* entities varies in different domains and datasets. In Agent-OV, we handle this problem in the following way. A matched entity is determined to be a *remain* entity when its syntactic name and lexical descriptions are unchanged. Otherwise it is determined to be an *update* entity. We observe that this approach means that Agent-OV usually considers entities to be *remain* entities when they have structural changes but conserve syntactic names and lexical descriptions. This approach will tend to consider ambiguous entities as *remain* in preference to *update*, so tending to trade off performance in *remain* versus performance in *update*. While other OV systems may handle this differently, we propose that all OM4OV systems could benefit from this approach to partitioning successful matches into *remain* entities and *update* entities.

## 4.3 Evaluation Criteria

OM typically measures performance using precision, recall, and the F1 score. Given a gold standard reference (R) and a system-discovered alignment (A), precision measures matching correctness and recall measures matching completeness, while F1 score offers a harmonic mean to balance correctness and completeness. OV can reuse these measures, but they need to be extended into four sub-measures for add, delete, remain, and update performance. While an OV method will trade off performance over each of these sets, an OV application problem may well prioritise accuracy on some over others.

$$Precision = \frac{|A \cap R|}{|A|} \quad Recall = \frac{|A \cap R|}{|R|} \quad F_1 \text{ Score} = \frac{2}{Precision^{-1} + Recall^{-1}} \quad (4)$$



#### 4.4 Results

We choose to use the Meta open-source model llama-3-8b [27] for our experiments in this paper, eschewing the higher-end models that offer small improvements that may not justify the financial investment. We use embeddings from the same LLM model to further avoid expensive API calls and thereby provide an entirely free-to-use version of Agent-OV. The hyperparameter settings are *similarity\_threshold* = 0.90 and *top@k* = 3 across all the system alignments generated. We set *top@k* = 3 based on empirical experience that 3-5 works well for Agent-OM. We experiment with the alternative values of the similarity threshold later in this paper. The fixed seed for the OV testbed is set to 42. Due to the non-determinism of LLMs, repeated experiments can produce insignificantly different results.

Figure 4 shows the performance of Agent-OM and Agent-OV on the OAEI Conference Track. The results indicate that it is possible to unify the OM and OV tasks using the OM4OV framework. Both Agent-OM and Agent-OV can produce alignments with precision, recall, and F1 scores. However, we can see that Agent-OM consistently performs surprisingly well across all alignments. This is because the traditional OM system captures only matched entities between two ontologies, which in the OV task correspond to the *remain* and *update* entities. Because *remain* entities typically comprise the major proportion of entities over two different versions, the correctness of *remain* entities dominates the overall evaluation and other categories are de-emphasised. We believe this is a common situation that leads to misleading performance measurement when using an OM system in OV tasks.

Agent-OV overcomes the skewed measure in Agent-OM. By decomposing the measure into four sub-measures, we can observe more informative matching performance across different OV tasks. In general, we observe that the matching performance is highest in *remain*, followed by *add* and *delete*, and relatively low in *update*. This trend is consistent across different tracks and ontologies. The measures for *remain* are typically very close to 100%. The measures for *add* and *delete* are generally good. Our system uses LLMs as the backend, and LLMs generally have strong background knowledge to detect non-matched entities. The measures for *update* show scope for improvement. This is because finding non-trivial alignments and appropriate similarity thresholds is not easy. We examine the false mappings produced by Agent-OV. Unlike OM tasks, we find that some of these identified false mappings are not actually false. Several examples are listed below with explanations.

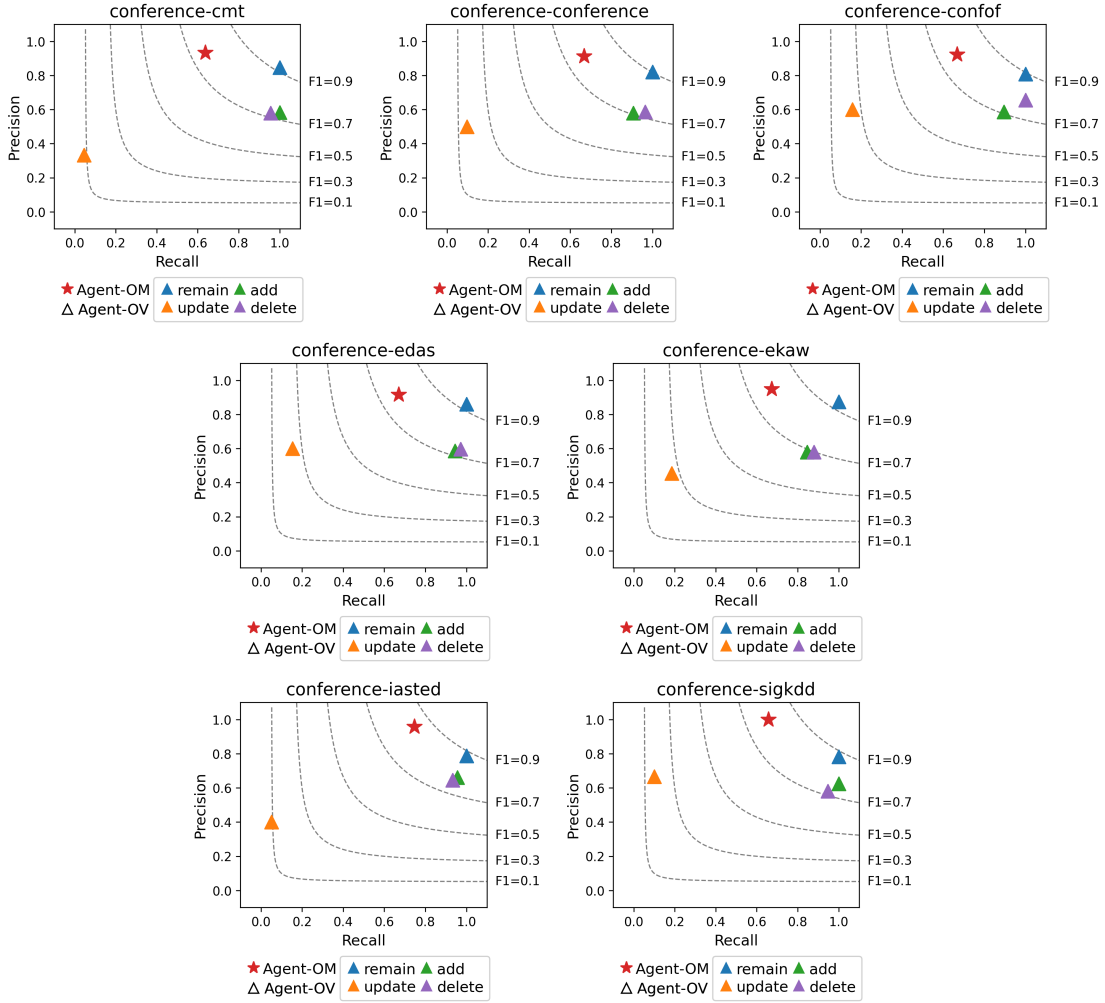
- (1) False mappings in OV can result from flawed ontology design choices, as shown here.

```
(edas:ConferenceEvent, edas:Event, c)
(edas:ConferenceEvent, edas:AcademicEvent, c)
```

In the example, the reference shows `edas:ConferenceEvent` is updated to `edas:Event`, but the OV system predicts `edas:ConferenceEvent` is updated to `edas:AcademicEvent`. An event can be a non-academic event (e.g. a social event) or an academic event (e.g. a seminar). In the context of a research conference, a conference event is more likely to mean the latter. However, mapping to event still makes sense. Within one ontology, the meaning of two entities is too close to be distinguished and therefore leads to them being synonyms for each other. Ideally, we should avoid this type of ontology design. This example also demonstrates a unique benefit of using OM for OV, which could potentially assist in ontology design.

- (2) False mappings in OV can create ambiguous “equivalent” relationships, as shown here.

```
(confof:dealsWith, confof:hasSubjectArea, c)
(confof:dealsWith, confof:reviews, c)
```



■ **Figure 4** Evaluation of OAEI Conference Track on the OV testbed. Agent-OM, shown as a red star, determines only the single category of entity matches. The extended Agent-OV, shown as triangles, determines four different categories of entity matches (*remain* or *update*) and non-matches (*add* or *delete*).

In the example, the reference shows `confof:dealsWith` is updated to `confof:hasSubjectArea`, but the OV system predicts `confof:dealsWith` is updated to `confof:reviews`. This is not necessarily wrong, but it interprets “deal with” differently. It is vital to notice that the term “equivalent” in OV is weaker than that in OM. OV allows for roughly “equivalent” mappings. The entities mapped in OV can slightly alter their meanings in response to real-world changes in the domain or evolution of natural language.

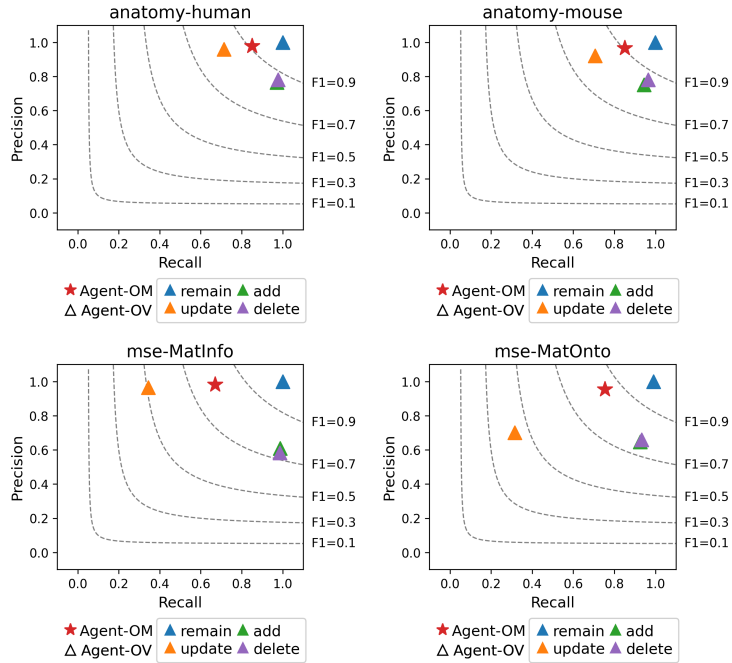
- (3) False mappings in OV can arise from different similarity thresholds, as shown here.

```
(sigkdd:Start_of_conference, None, 0.90)
(sigkdd:Start_of_conference, sigkdd:startDate, 0.80)
```

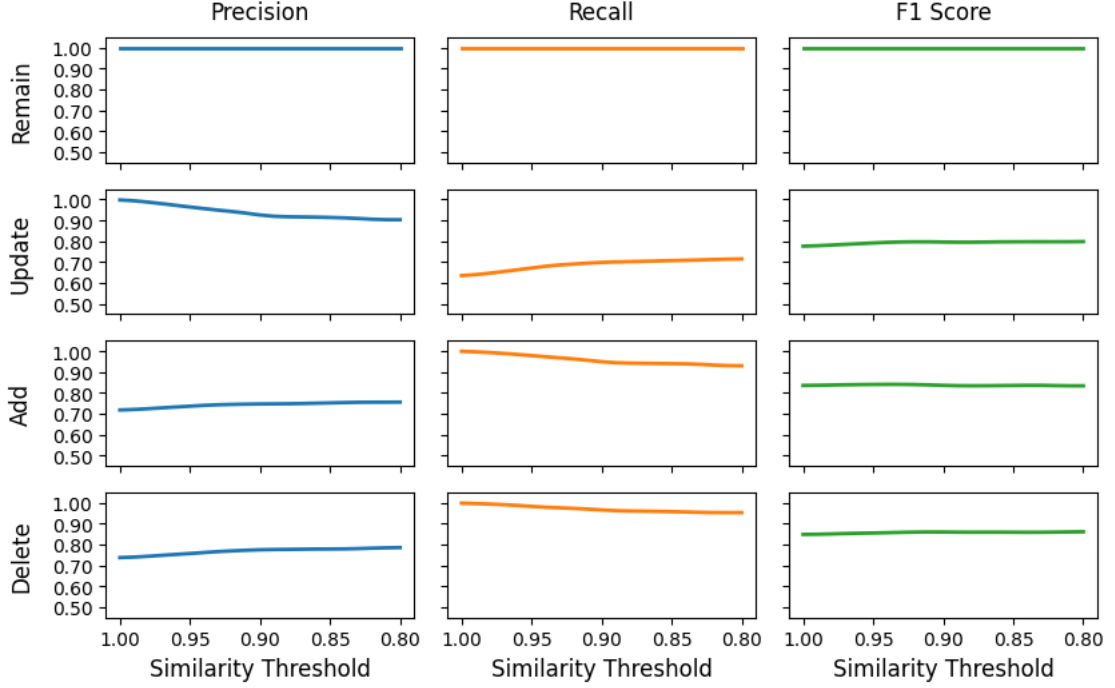
In the example, if *similarity\_threshold* = 0.90, `sigkdd:Start_of_conference` in *O* will be assigned to *delete* entities as it does not have matching entities; if *similarity\_threshold* = 0.80, `sigkdd:Start_of_conference` in *O* will be assigned to *update* entities as it has a matching entity `sigkdd:startDate` in *O'*. Both results are valid because defining the boundary between

matching and non-matching is context- and application-dependent. For example, the similarity threshold could be relatively higher in the biomedical domain to ensure that each term is unique, whereas the similarity threshold in the conference domain could be relatively lower to improve the interoperability of terminologies used in research conferences.

We extend our analysis to other OAEI tracks and varying similarity thresholds. Figure 5 shows that the trends are consistent with other medium and large OAEI tracks evaluated on the OV testbed. Additionally, we observe a longer computation time in the Anatomy Track. Although Agent-OV has an optimisation module for the matching candidate selection process (inherited from Agent-OM), it is still insufficient for some OV tasks. There is a need to optimise the framework for OV in large-scale ontologies. Figure 6 studies the effect of varying similarity thresholds for the Mouse ontology in Agent-OV. We change the similarity threshold from 1.00 to 0.80 to evaluate its effect on matching performance. While OV entities are classified into four different categories, each category requires a corresponding sub-measurement. Similar to OM tasks, changes in hyperparameter settings in OV tasks lead to a trade-off between precision and recall. Moreover, the hyperparameter settings also influence the sub-measures. We find that the similarity threshold has no effect on detecting *remain* entities, but significantly affects the detection of *update* entities and slightly affects the detection of *add* and *delete* entities. Detecting *update* entities and detecting *add* and *delete* entities are negatively correlated, as is expected because a near-miss *update* alignment will fall back to a pair of *non-match* entities, one from each ontology. Therefore, lower similarity thresholds can result in more *update* entities being detected, while higher similarity thresholds may find more *add* and *delete* entities.



■ **Figure 5** Evaluation of OAEI Anatomy and MSE Tracks on the OV testbed has shown similar trends.



■ **Figure 6** Evaluation varying similarity thresholds in Agent-OV. We apply a 1-D Gaussian filter for smoothing. The standard deviation for a Gaussian kernel is 1 (i.e.  $\sigma = 1$ ). We can see that the similarity threshold has no effect on detecting *remain* entities in the top row, but affects the detection of *update*, *add*, and *delete* entities.

## 4.5 Discussion

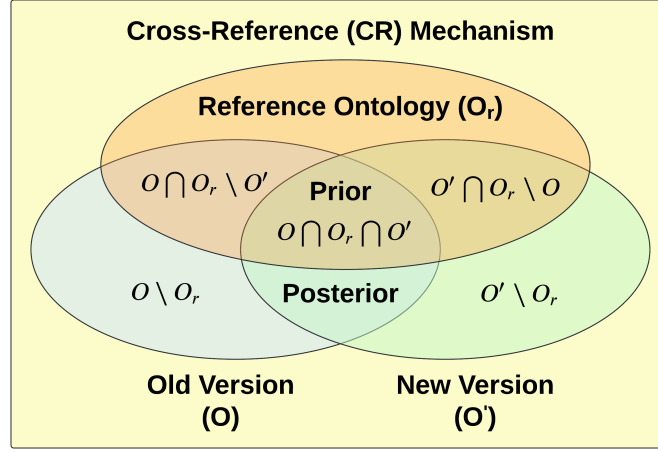
The experimental results have shown that OM systems can be reused for OV tasks, but they have limitations and require necessary extensions. The following key points need to be taken into account when reusing OM systems for OV tasks.

- (1) **Skewed Measurement:** OM systems exhibit skewed measurement of OV tasks because unchanged *remain* entities dominate in practice. OM measures are reusable, but these measures need to be extended into four sub-measures for *add*, *delete*, *remain*, and *update* performance. Within each sub-measure, the inherent precision-recall trade-off still holds. Across different sub-measures, they are not independent. This is because each change in a part of an alignment in one category will cause other categories to change accordingly. For example, if a new mapping is found in  $A \odot$  or  $A \otimes$ , then the number of mappings in  $A \oplus$  and  $A \ominus$  will be reduced by one each, and vice versa.
- (2) **Update Pitfalls:** OM systems are sensitive to changes in similarity thresholds and backend LLMs. For OV tasks, only *update* entities are sensitive to these changes, while *remain*, *add*, and *delete* are relatively stable across these hyperparameter settings. Poor *update* performance is common, while tuning the similarity threshold trades precision for recall without resolving the underlying difficulty.
- (3) **Disputed False Mappings:** OM systems used for OV tasks can produce mappings that are false with respect to the ground truth reference, but that are true with respect to expert review. This can be caused by flawed ontology design choices, ambiguous “equivalent” relationships, or inappropriate settings of the similarity threshold. This is difficult for OM systems to recognise and generally requires human effort.

## 5 OM4OV Optimisation

Having established the current pitfalls of OM4OV, we explore optimisations for the framework. Often, ontology creators provide cross-references to alternative ontologies to enhance interoperability and facilitate integration. For example, a cross-reference between the Mouse ontology and the Human ontology is provided in the OAEI Anatomy Track. Reusing the cross-references developed for OM tasks, we propose a novel mechanism to reduce the number of matching candidates and also to improve overall OV performance.

Figure 7 illustrates the cross-reference (CR) mechanism used in the OM4OV framework. We can see that, without using the cross-reference  $O_r$ , the matching candidates cover the range of  $O \cup O'$ . This number can be significantly reduced by removing prior matches (i.e.  $O \cap O_r \cap O'$ ) and non-matches (i.e.  $O \cap O_r \setminus O'$  and  $O' \cap O_r \setminus O$ ). The prior matching will be incorporated into the final alignment, while the known non-matching regions will be removed entirely in the subsequent OV process. The OV process then only determines the posterior alignment.



■ **Figure 7** Illustration of the cross-reference (CR) mechanism.

In practice, the prior alignment usually contains a large number of *remain* entities and a small number of *update* entities. Matching performance is also enhanced by utilising these established mappings. Since prior matches are inferred from OM references validated by domain experts, they provide a solid ground truth for alignment within a specific domain. Removing known non-matches from the OV process also simplifies posterior alignment detection. The optimised OM4OV framework using the CR mechanism is defined as follows.

► **Definition 4.** Given a reference ontology ( $O_r$ ), an old version of an ontology ( $O$ ) and a new version of the same ontology ( $O'$ ), two cross-references between  $O$  and  $O_r$  ( $R_{or}$ ) and between  $O'$  and  $O_r$  ( $R_{o'r}$ ) are defined respectively as:

$$\begin{aligned} R_{or} &= \{(e_1, e_3, c) \mid e_1 \in O, e_3 \in O_r, s \leq c \leq 1 \text{ and there is no } (e_1, e_3, c') \text{ with } c' > c\} \\ R_{o'r} &= \{(e_2, e_3, c) \mid e_2 \in O', e_3 \in O_r, s \leq c \leq 1 \text{ and there is no } (e_2, e_3, c') \text{ with } c' > c\} \end{aligned} \quad (5)$$

We can use  $R_{or}$  and  $R_{o'r}$  to infer some known mappings between  $O$  and  $O'$  before performing OV. We call these mappings a *prior* alignment ( $A_\pi$ ). After subsequently performing OV, we have our *posterior* alignment ( $A_{\pi^*}$ ). In this setting,  $A_{match}$  in OV can be decomposed into two parts:  $A_{match} = A_\pi \cup A_{\pi^*}$ .  $A_\pi$  can be directly inferred from the two cross-references  $R_{or}$  and  $R_{o'r}$ . By assuming the equivalence relation as extracted from the two cross-references is transitive, we have

that  $e_1 \in O$ ,  $e_2 \in O'$ , and  $e_3 \in O_r$ , if  $e_1 \equiv e_3$  and  $e_2 \equiv e_3$  then  $e_1 \equiv e_2$ .  $A_{\pi^*}$  aims to detect missing mappings from the cross-reference. None of these mappings would come from any pairwise intersection of  $O$ ,  $O_r$ , and  $O'$  because  $O \cap O_r \setminus O'$  and  $O' \cap O_r \setminus O$  are pre-defined as non-matched entities, and the matched entities in  $O \cap O_r \cap O'$  have already been captured in the  $A(\pi)$ . As a result,  $A_{\pi^*}$  can be defined within a smaller scope.

► **Definition 5.** *Prior alignment ( $A_{\pi}$ ) and posterior alignment ( $A_{\pi^*}$ ) are defined as:*

$$\begin{aligned} A_{\pi} &= R_{or} \cap R_{o'r} = \{(e_1, e_2, c) \mid e_1, e_2 \in O \cap O_r \cap O', s \leq c \leq 1\} \\ A_{\pi^*} &= \{(e_1, e_2, c) \mid e_1 \in O \setminus O_r, e_2 \in O' \setminus O_r, s \leq c \leq 1\} \end{aligned} \quad (6)$$

An ontology can have multiple cross-references available. In such cases, the *prior* reference becomes the union of all known cross-references ( $R_{or1} \dots R_{orn}$ ), and the ontology used in the *posterior* alignment ( $O_{ra}$ ) becomes the union of all reference ontologies ( $O_{r1} \dots O_{rn}$ ).

► **Definition 6.**  *$A_{\pi}$  and  $A_{\pi^*}$  in multiple cross-references can be formulated as:*

$$\begin{aligned} A_{\pi} &= (R_{or1} \cap R_{o'r1}) \cup (R_{or2} \cap R_{o'r2}) \cup \dots \cup (R_{orn} \cap R_{o'rn}) \\ A_{\pi^*} &= \{(e_1, e_2, c) \mid e_1 \in O \setminus O_{ra}, e_2 \in O' \setminus O_{ra}, c \geq s\} \text{ where } O_{ra} = O_{r1} \cup O_{r2} \cup \dots O_{rn} \end{aligned} \quad (7)$$

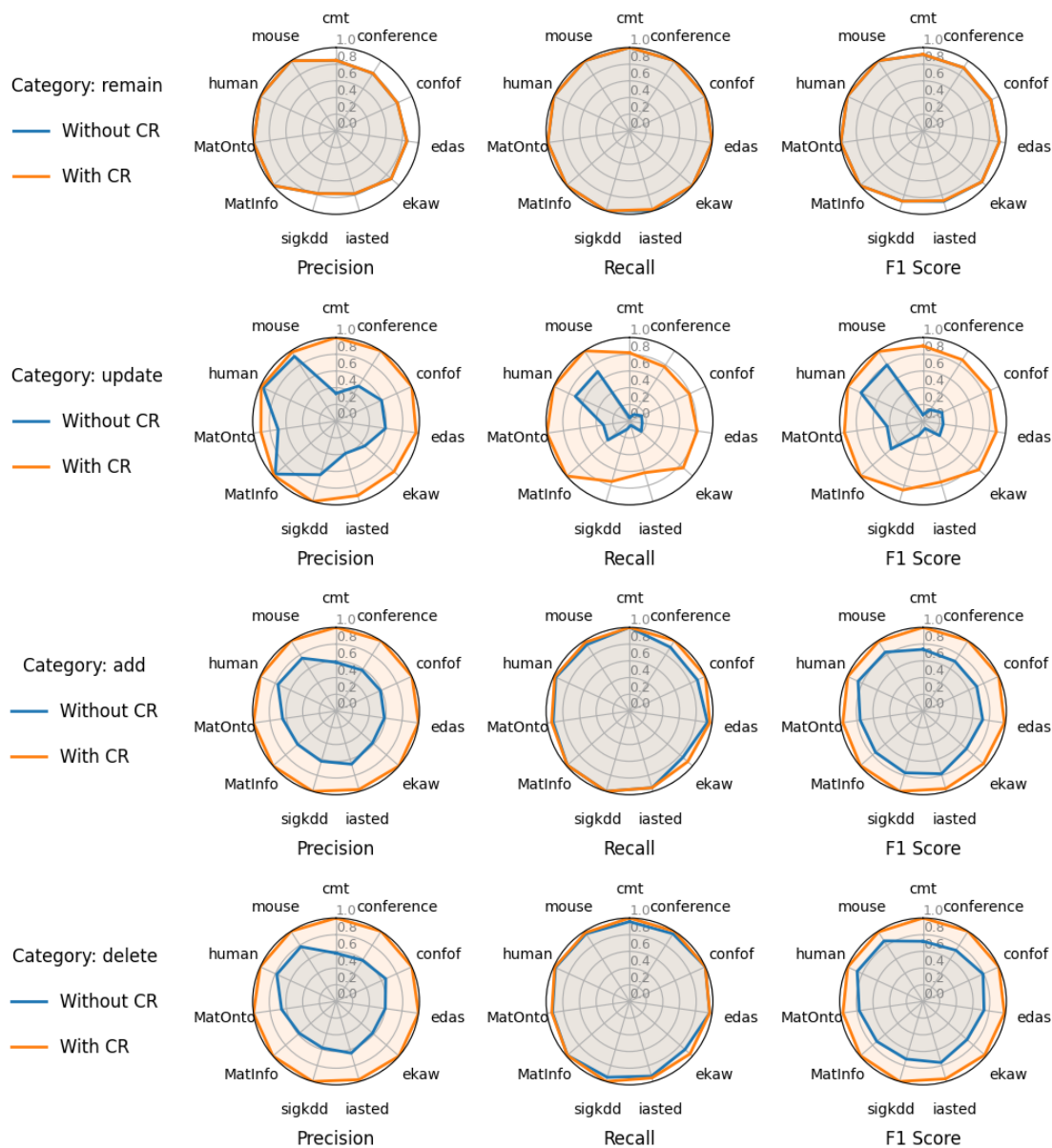
Figure 8 compares the OV performance with and without the CR mechanism on the same OV testbed that we analysed in Section 4. We can see that the CR mechanism significantly enhances Agent-OV's ability to detect *update* entities and slightly improves its performance in detecting *add* and *delete* entities, with no effect on *remain* entities. More specifically, performance improvement is mainly attributed to recall for *update* entities, whereas it is more evident in precision for *add* and *delete* entities. Figure 9 compares the effects of different similarity thresholds for the Mouse ontology versioning. The results of the experiment show that the CR mechanism is less sensitive to the similarity threshold than the original OM4OV framework in Figure 6, suggesting that it can be helpful in scenarios where the optimal similarity threshold is unclear or difficult to determine.

## 6 OM4OV in Real-World Applications

Brick Schema [4] is a popular ontology used in the building industry. Introduced in 2016, Brick Schema has undergone several major version changes, including URI changes, schema language changes, and ontology harmonisation (combining Brick Schema with RealEstateCore [16] and Project Haystack [25]). We use the change logs from the official GitHub repository to provide near-ground-truth alignments for evaluation. We are not confident about the reliability of the manually-generated logs as we observe instances where the log entry for a change conflicts with the difference observed in the ontology itself. We demonstrate versioning performance over three distinct major version updates: from v1.1.0 to v1.4.0, v1.2.0 to v1.4.0, and v1.3.0 to v1.4.0. We have collapsed multiple human-identified minor version changes into the corresponding major version changes for our experiments to consider more complex version changes than are evident in successive minor versions. We filter out classes and properties that are used in Brick Schema conventions but that user experts determined to be unimportant for version tracking and hence have not been captured in change logs. For example, “Tag”, “NodeShape”, and “PropertyShape” are excluded from the matching candidates.

Figure 10 shows that our OM4OV implemented in the OM system is more precise than our previous OM-only system in tracking changes across different versions of Brick Schema. Similar trends are observed in skewed measurements, update pitfalls, and disputed false mappings. We are unable to empirically evaluate our cross-reference mechanism with Brick Schema. Unfortunately,

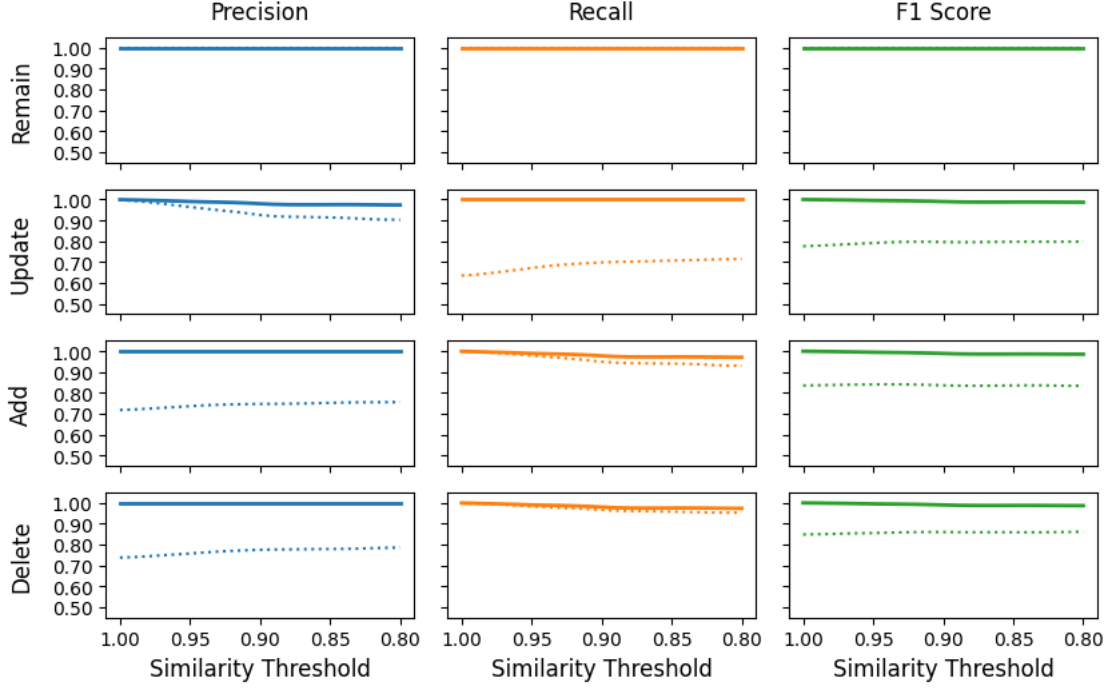




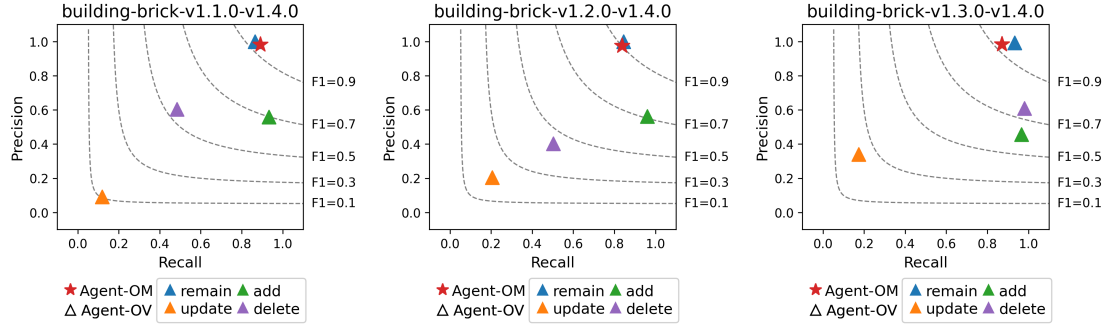
■ **Figure 8** Evaluation of the cross-reference (CR) mechanism on the OV testbed. Each radar chart shows measurements (precision, recall, and F1 score) for a change category over 11 datasets analysed around the perimeter, both with (*orange*) and without (*blue*) the CR mechanism applied.

while Brick Schema offers high-level alignments with RealEstateCore and Project Haystack, these alignments are highly incomplete, addressing only few classes and properties. Further, they do not indicate versions of the ontologies to which they apply. We have been unable to locate an alternative ontology that offers the data needed for evaluating our cross-reference mechanism. We hope our work contributes to the quality improvement of OV so that building high-quality versioned reference alignments becomes standard industry practice. At that time, automated system updates with respect to ontology version changes would become feasible, and we could evaluate our cross-reference mechanism on real-world data.





■ **Figure 9** Evaluation of the cross-reference (CR) mechanism on different similarity thresholds. The solid lines contribute to OV performance with the CR mechanism, while the dotted lines contribute to OV performance without the CR mechanism. We apply a 1-D Gaussian filter for smoothing. The standard deviation for a Gaussian kernel is 1 (i.e.  $\sigma = 1$ ).



■ **Figure 10** Evaluation of the OM4OV framework tracking Brick Schema version changes.

## 7 Conclusion

In this paper, we systematically analyse similarities and differences between OM and OV tasks and provide formal definitions and task formulations for OM4OV. The experimental results from Agent-OM and Agent-OV demonstrate that the OM4OV framework is workable, though it presents several limitations. To tackle these issues, we propose a novel optimisation method that uses a cross-reference (CR) mechanism to leverage reference alignments from OM for OV. This method (1) overcomes several pitfalls in using OM for OV tasks, (2) significantly reduces the number of matching candidates, and (3) improves overall OV performance. We showcase the use of our OM4OV framework in tracking version changes in a widely used ontology for the building industry.

We observe three opportunities for future work. (1) While OM has a universal evaluation measure, our proposed measures for OV have four sub-measures. We plan to investigate a merged formula for OV sub-measures in the future, for example, using a harmonic mean to combine sub-measures. (2) We limit the semantic analysis to noting changes in 1-hop subgraph relations that can be either ontology keywords (such as domain, range, and subclass) or domain concepts as object properties. A more complete analysis of structural evolution could employ description-logic-based reasoning for subsumption checking, consistency validation, and detection of semantically equivalent expressions with only syntactic differences. This kind of analysis has not been previously explored in OM and is not addressed in this study, but could be worthwhile further work. (3) We plan to apply the OM4OV framework in emerging domain ontologies that demonstrate regular changes over time. We are working with the Brick Schema development team to encourage adoption of our OM4OV framework.

## Statement on Supplementary Materials and Additional Resources

The following supplementary materials belong to this paper:

- (S1) *Software (OM4OV and Agent-OV)*: <https://github.com/qzc438/ontology-versioning>  
This artifact contains source code for OM4OV and Agent-OV, as well as the versioning datasets mentioned in Section 4.1.
- (S2) *Software (Brick Schema Version Track)*: <https://github.com/qzc438/brick-version-track>  
This artifact contains the source code for tracking Brick Schema [4] version changes mentioned in Section 6.

The following resources were used in the mentioned parts of the paper:

- (R1) *Software (Agent-OM) [36]*: <https://github.com/qzc438/ontology-11m>  
This artifact contains the source code for Agent-OM [37] mentioned in Sections 4.2 and 4.4.
- (R2) *Dataset (OAEI Datasets) [26]*: <https://dwsllab.github.io/melt/track-repository>  
This artifact shows the Matching Evaluation Toolkit (MELT) [18] repository to download the original OAEI datasets for (S1) (retrieved April 1, 2025). According to the OAEI data policy, “OAEI results and datasets, are publicly available, but subject to a use policy similar to the one defined by NIST for TREC. These rules apply to anyone using these data.” Please find more details from the official website: <https://oaei.ontologymatching.org/doc/oaei-deontology.2.html> (retrieved April 1, 2025).
- (R3) *Dataset (Brick Schema Change Logs) [6]*: <https://github.com/BrickSchema/Brick/releases>  
This artifact shows the link to the Brick Schema [4] official repository to download the change logs for (S2) (retrieved April 1, 2025).

---

## References

- 1 Alibaba Qwen Team. Qwen Models, n.d. URL: <https://qwenlm.github.io>.
- 2 Anthropic. Claude Models, n.d. URL: <https://docs.anthropic.com/en/docs/about-claude/models>.
- 3 Toshihiro Ashino. Materials ontology: An infrastructure for exchanging materials information and knowledge. *Data Science Journal*, 9:54–61, 2010. doi:10.2481/dsj.008-041.
- 4 Bharathan Balaji, Arka Bhattacharya, Gabriel Fierro, Jingkun Gao, Joshua Gluck, Dezhi Hong, Aslak Johansen, Jason Koh, Joern Ploennigs, Yuvraj Agarwal, Mario Berges, David Culler, Rajesh Gupta, Mikkel Baun Kjærgaard, Mani Srivastava, and Kamin Whitehouse. Brick: Towards a unified metadata schema for buildings. In *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*, pages 41–50, Palo Alto, CA, USA, 2016. ACM. doi:10.1145/2993422.2993577.
- 5 Conrad Bock, Achille Fokoue, Peter Haase, Rinke Hoekstra, Ian Horrocks, Alan Ruttenberg, Uli Sattler, and Michael Smith. OWL 2 web ontology language, 2012. URL: <https://www.w3.org/TR/owl2-syntax/>.

- 6 Brick Schema Development Team. Brick Schema change logs. Last accessed: April 1, 2025. URL: <https://github.com/BrickSchema/Brick/releases>.
- 7 Silvio Domingos Cardoso, Marcos Da Silveira, and Cédric Pruski. Construction and exploitation of an historical knowledge graph to deal with the evolution of ontologies. *Knowledge-Based Systems*, 194:105508, 2020. doi:10.1016/j.knosys.2020.105508.
- 8 Silvana Castano, Alfio Ferrara, and Stefano Montanelli. A matchmaking-based ontology evolution methodology. In *Proceedings of the Open Interop Workshop on Enterprise Modelling and Ontologies for Interoperability, Co-located with CAiSE'06 Conference*, volume 200, Luxembourg, 2006. CEUR-WS.org. URL: <https://ceur-ws.org/Vol-200/02.pdf>.
- 9 European Materials Modelling Council. EMMO, n.d. URL: <https://github.com/emmo-repo/EMMO>.
- 10 Jérôme Euzenat. Semantic precision and recall for ontology alignment evaluation. In *Proceedings of the 20th international joint conference on Artificial intelligence*, pages 348–353, Hyderabad, India, 2007. Morgan Kaufmann. URL: <http://ijcai.org/Proceedings/07/Papers/054.pdf>.
- 11 Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching (2nd ed.)*. Springer, Berlin, Germany, 2013. doi:10.1007/978-3-642-38721-0.
- 12 Google DeepMind. Google Gemma Models, n.d. URL: <https://ai.google.dev/gemma>.
- 13 Anika Gross, Michael Hartung, Andreas Thor, and Erhard Rahm. How do computed ontology mappings evolve? - a case study for life science ontologies. In *Proceedings of the 2nd Joint Workshop on Knowledge Evolution and Ontology Dynamics in conjunction with the 11th International Semantic Web Conference (ISWC 2012)*, volume 890, Boston, MA, USA, 2012. CEUR-WS.org. URL: <https://ceur-ws.org/Vol-890/paper1.pdf>.
- 14 Thomas R. Gruber. A translational approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993. doi:10.1006/knac.1993.1008.
- 15 Armin Haller and Axel Polleres. Are we better off with just one ontology on the web? *Semantic Web*, 11(1):87–99, 2020. doi:10.3233/SW-190379.
- 16 Karl Hammar, Erik Oskar Wallin, Per Karlberg, and David Hälleberg. The RealEstateCore ontology. In *The Semantic Web - ISWC 2019*, pages 130–145, Auckland, New Zealand, 2019. Springer. doi:10.1007/978-3-030-30796-7\_9.
- 17 Jeff Heflin and James Hendler. Dynamic ontologies on the web. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 443–449, Austin, TX, USA, 2000. AAAI Press. URL: <https://aaai.org/papers/00443-AAAI00-068-dynamic-ontologies-on-the-web/>.
- 18 Sven Hertling, Jan Portisch, and Heiko Paulheim. MELT – matching evaluation toolkit. In *Semantic Systems. The Power of AI and Knowledge Graphs*, volume 11702, pages 231–245, Karlsruhe, Germany, 2019. Springer. doi:10.1007/978-3-030-33220-4\_17.
- 19 iNovex IRAD. Ontologies for the MatOnto project, n.d. URL: <https://github.com/inovexcorp/MatOnto-Ontologies>.
- 20 Michel Klein and Dieter Fensel. Ontology versioning on the semantic web. In *Proceedings of the First International Conference on Semantic Web Working*, pages 75–91, Stanford University, CA, USA, 2001. CEUR-WS.org. URL: <https://dl.acm.org/doi/10.5555/2956602.2956610>.
- 21 Michel Klein, Dieter Fensel, Atanas Kiryakov, and Damyan Ognyanov. Ontology versioning and change detection on the web. In *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, pages 197–212, Sigüenza, Spain, 2002. Springer. doi:10.1007/3-540-45810-7\_20.
- 22 Adrianna Kozierekiewicz and Marcin Pietranik. Triggering ontology alignment revalidation based on the degree of change significance on the ontology concept level. In *Business Information Systems*, pages 137–148, Seville, Spain, 2019. Springer. doi:10.1007/978-3-030-20485-3\_11.
- 23 Adrianna Kozierekiewicz and Marcin Pietranik. Updating ontology alignment on the concept level based on ontology evolution. In *Advances in Databases and Information Systems*, pages 201–214, Bled, Slovenia, 2019. Springer. doi:10.1007/978-3-030-28730-6\_13.
- 24 Adrianna Kozierekiewicz and Marcin Pietranik. Updating ontology alignment on the relation level based on ontology evolution. In *Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering*, pages 241–248, Prague, Czech Republic, 2020. SciTePress. doi:10.5220/0009142002410248.
- 25 John J. McGowan. Project haystack data standards. In *Energy and Analytics*, pages 237–243. River Publishers, Aalborg, Denmark, 2020. doi:10.1201/9781003151944-16.
- 26 MELT Team. OAEI datasets. Last accessed: April 1, 2025. URL: <https://dwsllab.github.io/melt-track-repository>.
- 27 Meta. llama-3-8b, 2024. URL: <https://huggingface.co/meta-llama/Meta-Llama-3-8B>.
- 28 Meta. Meta Llama Models, n.d. URL: <https://www.llama.com>.
- 29 Natalya F. Noy, Sandhya Kunnatur, Michel Klein, and Mark A. Musen. Tracking changes during ontology evolution. In *The Semantic Web - ISWC 2004*, pages 259–273, Hiroshima, Japan, 2004. Springer. doi:10.1007/978-3-540-30475-3\_19.
- 30 Natalya F. Noy and Mark A. Musen. PROMPTDIFF: A fixed-point algorithm for comparing ontology versions. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 744–750, Edmonton, Canada, 2002. AAAI Press. URL: <https://aaai.org/papers/00744-aaai02-112-promptdiff-a-fixed-point-algorithm-for-comparing-ontology-versions/>.
- 31 Natalya F. Noy and Mark A. Musen. Ontology versioning in an ontology management framework. *IEEE Intelligent Systems*, 19(4):6–13, 2004. doi:10.1109/MIS.2004.33.

- 32 OAEI Community. Ontology Alignment Evaluation Initiative (OAEI), n.d. URL: <https://oaei.ontologymatching.org>.
- 33 OpenAI. OpenAI Models, n.d. URL: <https://platform.openai.com/docs/models>.
- 34 Marcin Pietranik and Adrianna Kozierekiewicz. Methods of managing the evolution of ontologies and their alignments. *Applied Intelligence*, 53(17):20382–20401, 2023. doi:10.1007/S10489-023-04545-0.
- 35 Peter Plessers and Olga De Troyer. Ontology change detection using a version log. In *The Semantic Web – ISWC 2005*, pages 578–592, Galway, Ireland, 2005. Springer. doi:10.1007/11574620\_42.
- 36 Zhangcheng Qiang, Weiqing Wang, and Kerry Taylor. Agent-OM. Last accessed: April 1, 2025. URL: <https://github.com/qzc438/ontology-llm>.
- 37 Zhangcheng Qiang, Weiqing Wang, and Kerry Taylor. Agent-OM: Leveraging LLM agents for ontology matching. *Proceedings of the VLDB Endowment*, 18(3):516–529, 2024. doi:10.14778/3712221.3712222.
- 38 RDF-star Working Group. RDF 1.2 schema, 2024. URL: <https://www.w3.org/TR/rdf12-schema/>.
- 39 Najla Sassi, Wassim Jaziri, and Saad Alharbi. Supporting ontology adaptation and versioning based on a graph of relevance. *Journal of Experimental & Theoretical Artificial Intelligence*, 28(6):1035–1059, 2016. doi:10.1080/0952813X.2015.1056239.
- 40 Team GLM. ChatGLM: A family of large language models from GLM-130B to GLM-4 all tools, 2024. doi:10.48550/arXiv.2406.12793.
- 41 Ahmed Zahaf and Mimoun Malki. Alignment evolution under ontology change. *International Journal of Information Technology and Web Engineering*, 11(2):14–38, 2016. doi:10.4018/IJITWE.2016040102.
- 42 Abir Zekri, Zouhaier Brahmia, Fabio Grandi, and Rafik Bouaziz.  $\tau$ OWL: A systematic approach to temporal versioning of semantic web ontologies. *Journal on Data Semantics*, 5(3):141–163, 2016. doi:10.1007/s13740-016-0066-3.



# Logica-TGD: Transforming Graph Databases Logically

Evgeny Skvortsov 

Google LLC, Kirkland, WA, USA

Yilin Xia 

School of Information Sciences, University of Illinois Urbana-Champaign, IL, USA

Bertram Ludäscher 

School of Information Sciences, University of Illinois Urbana-Champaign, IL, USA

Shawn Bowers 

Department of Computer Science, Gonzaga University, Spokane, WA, USA

---

## Abstract

Graph transformations are a powerful computational model for manipulating complex networks, but handling temporal aspects and scalability remain significant challenges. We present a novel approach to implementing graph transformations using Logica, an open-source logic programming language and system that operates over standard database systems including PostgreSQL, DuckDB, and BigQuery. Logica leverages the inherent parallelism of these engines and offers a practical and scalable way to carry out a variety of graph transformations, including complex scenarios such as time-varying graphs. We illustrate Logica's graph

querying and transformation capabilities with several examples, including a declarative program for pathfinding in a dynamic graph and a taxonomic analysis over a full current Wikidata dump. Experimental results compare Logica running on DuckDB engine to the Datalog engines Soufflé and Nemo, and to DuckPGQ, an implementation of SQL/PGQ. We argue that a logic-based declarative syntax, a built-in visualization library, and (plug-and-play) support for large-scale database engines make Logica a convenient and practical tool for a wide range of graph transformation tasks.

**2012 ACM Subject Classification** Information systems → Query languages for non-relational engines

**Keywords and phrases** Logic rules, Graph queries, Graph transformations

**Digital Object Identifier** 10.4230/TGDK.4.2.7

**Category** Resource

**Related Version** *Previous Version:* <https://arxiv.org/abs/2503.00568>

**Supplementary Material** *Software (Source Code):* <https://github.com/EvgSkv/logica>

**Received** 2025-07-19 **Accepted** 2026-06-22 **Published** 2026-09-03

**Editors** Stefania Dumbrava, George Fletcher, Matteo Lissandrini, and Riccardo Tommasini

**Special Issue** Data Management for (Knowledge) Graphs

**Generative AI Declaration** Generative AI was used in a limited, assistive capacity in preparing this article, and all output was reviewed and verified by the authors. Generative AI was not used to generate references or citations.

## 1 Introduction

Graph transformations are a powerful and versatile method for modeling and manipulating complex systems across diverse fields, ranging from software engineering [9, 27] and social network analysis [14] to biology and chemistry [35, 31, 36]. These transformations typically operate by applying rewrite rules to a graph, altering its structure and properties. While such transformations are known for their expressiveness and flexibility, their implementation can often be complex, especially when dealing with time-varying (i.e., evolving) graphs and when scalable solutions are needed. Additionally, existing graph database systems often provide limited support for such evolution mechanisms, creating a gap between theoretical models and practical implementations.



© Evgeny Skvortsov, Yilin Xia, Bertram Ludäscher, and Shawn Bowers;  
licensed under Creative Commons License CC-BY 4.0

Transactions on Graph Data and Knowledge, Vol. 4, Issue 2, Article No. 7, pp. 7:1–7:27



Transactions on Graph Data and Knowledge

TGD Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Logic programming, on the other hand, provides a declarative approach to problem solving by expressing relationships using rules rather than explicitly stating control-flow. Logic-based systems (e.g., Prolog and Answer Set Programming [17]) are well-established in various areas, including (symbolic) AI and knowledge representation and reasoning.

More recently, Logica (*Logic* + *Aggregation*) [41, 40], an open-source logic programming language and system, has emerged, which employs performant, parallel data processing environments such as DuckDB and BigQuery. Logica combines the declarative power of logic programming with the scalability and efficiency of modern databases, offering a promising new path for tackling graph transformation challenges.

This paper explores the application of Logica to the domain of graph transformations, bridging the gap between these two paradigms. Our approach leverages Logica’s ability to process large datasets in a parallel and efficient manner, providing a novel means of implementing and executing graph transformations at scale using logical rules. We argue that with a graph transformation mindset, logic programming can provide a natural, powerful, and intuitive approach for defining transformations, opening new opportunities within both communities. Crucially, Logica also enables a practical approach to addressing issues that are difficult in classical graph transformation approaches, such as temporal transformations and seamless scalability.

To illustrate the practical nature of our approach, we introduce several example Logica programs implementing different types of graph transformations. Specifically, we show how Logica can be used to define and perform basic graph transformations including message passing, removing transitively implied graph edges, graph extraction, and computing complex annotation properties (by finding solution labels for two-player combinatorial game graphs). We also introduce a novel approach to time-aware pathfinding (e.g., [25, 8]), directly addressing the need for principled and tractable mechanisms to handle evolving graph data, a notoriously difficult problem in classical graph transformation approaches. These examples serve both to show the expressiveness of our approach, and to illustrate its potential benefits.

In the experimental evaluation (Section 4), we include a same-generation benchmark where aggregation within recursion enables a purely declarative solution: the generation of a child is defined as the next generation after its parent’s, and same-generation pairs follow by equality. This illustrates how Logica’s support for aggregation in recursive rules extends the class of problems for which a declarative definition also serves as an efficient algorithm.

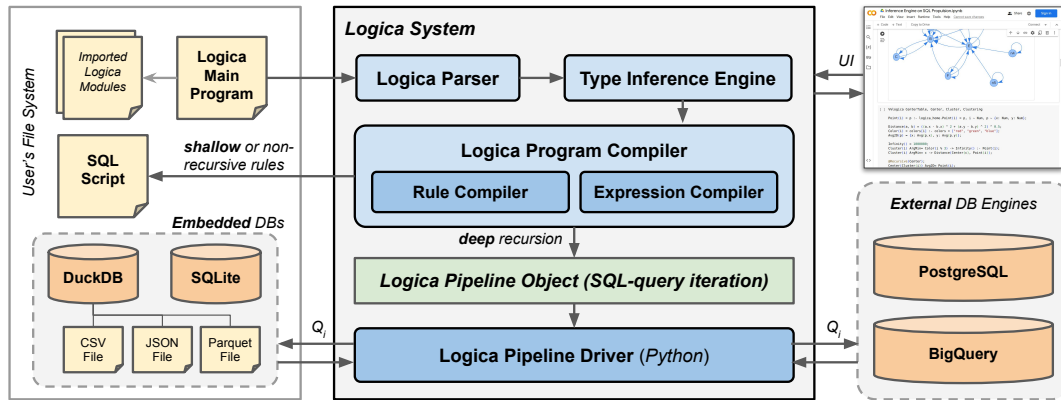
Examples demonstrated in this paper are available online [39], as part of the Logica repository [38]. Examples are represented in online Google Jupyter Notebooks (via Colab), where they can be executed and modified.

The rest of this paper is organized as follows. Section 2 provides an overview of Logica including its overall architecture, syntax and extensions (compared to traditional Datalog), and its program evaluation approach. Section 3 demonstrates how to express a variety of graph transformations in Logica. In Section 4 we report results of comparison of Logica execution speed with Soufflé, Nemo and DuckPGQ engines. And in Section 5 we describe location of Logica in the body of related work. Section 6 concludes with a discussion of limitations and potential avenues for future work.

## **2      Logica Overview**

Logica is a freely available, open-source logic programming language that bridges declarative rule specification with the computational power of modern parallel databases. Rather than implementing a specialized engine, Logica employs a compilation-based architecture that translates logical programs into optimized SQL, enabling execution on industrial database systems DuckDB, SQLite, PostgreSQL, and BigQuery.





■ **Figure 1 System Architecture:** Logica supports a module system for importing libraries (top left); users write programs either locally (top left), e.g., as part of a larger Python script, or from within Jupyter notebooks (top right); programs are parsed, type checked (top middle), and, if well-formed, compiled to a self-contained SQL script (no/limited recursion) or to an iterative plan of pipelined SQL-queries that is passed to a pipeline driver (bottom middle). Logica then executes and monitors queries via underlying SQL engines, either locally (bottom left) or externally (bottom right).

## 2.1 Logica System Architecture

The Logica compilation process, illustrated in Figure 1, consists of three main stages. The *parser* reads the source program and build an Abstract Syntax Tree (AST). The *type inference engine* analyzes this tree, assigning types for error detection and dialect-specific SQL generation. Finally, the *compiler* translates the type-annotated program into executable SQL.

For non-recursive or shallow-recursive programs, the compiler produces a single, self-contained SQL script. For complex recursive computations, which are common in graph transformation, Logica generates SQL that unfolds recursion into a fixed number of iterations using intermediate tables or nested queries. Each iteration applies the recursive rule to results from the previous iteration, with results combined via UNION ALL. This iterative unfolding mechanism is fundamental to Logica’s ability to handle the complex transformations central to graph processing. For programs requiring dynamic fixpoint computation, Logica can generate a Python-based pipeline that iteratively executes these SQL queries until convergence or a termination condition is met.

Logica is capable of using several SQL engines, running its pipeline on SQLite, DuckDB, PostgreSQL, or BigQuery. SQLite is a light yet efficient engine that comes along with Python; with it you can run Logica without installing anything else, however you need to enable it explicitly in your program with an `@Engine("sqlite")` directive. The default engine for Logica is DuckDB, a highly efficient engine built specifically for massive data analysis. DuckDB achieves such efficiency because massive data processing and analysis is its primary purpose, while SQLite and PostgreSQL have much of their focus on efficiently supporting individual-level transactions; thus DuckDB is more aligned with Logica’s purpose. Finally, when data that needs processing is of terabytes in volume, you can process it with Logica using BigQuery, a massively parallel data warehouse.

Logica is distributed via PyPI and can be installed with `pip install logica`. It can be used from the command line or from Jupyter notebooks. Documentation and source code are available at [logica.dev](http://logica.dev) and [github.com/EvgSkv/logica](https://github.com/EvgSkv/logica). The compiler architecture is modular with respect to the SQL backend; for instance, experimental support for ClickHouse and Trino has been explored. Supporting a new engine requires accommodating its SQL dialect and available features.

## 2.2 Logica Syntax Extensions

Logica extends traditional Datalog with several syntactic features designed to help with readability and robustness, particularly for applications involving structured relations and graph transformations.

**Named and Positional Arguments.** Logica supports both traditional positional arguments and named arguments. Positional arguments define predicates through variable order, suitable for simple relationships. The following example defines two facts and a single rule in Logica using positional arguments. In Logica, predicates begin with uppercase characters, variables begin with lowercase characters, and statements (facts and rules) are terminated by semicolons.

```
Parent("Adam", "Cain");
Parent("Cain", "Enoch");
Grandparent(x, z) :- Parent(x, y), Parent(y, z);
```

Disjunction in Logica, like in Prolog, can be expressed with multiple rules for one predicate, or with the binary operator `|`. Conjunction is of higher priority than disjunction, so parentheses are not required around individual disjuncts. For example, the two facts above can equivalently be written as a single rule:

```
Parent(x, y) :- x == "Adam", y == "Cain" |
               x == "Cain", y == "Enoch";
```

Named arguments, as in SQL, can help with readability, especially for relations with many attributes. The following example shows an `Employee` relation with four named attributes representing the employee's id, name, department, and salary.

```
Employee(employee_id: 101, name: "Alice",
          department: "Sales", salary: 80000);
Employee(employee_id: 102, name: "Bob",
          department: "Engineering", salary: 95000);
```

Named arguments provide position independence and robustness to schema evolution: queries remain valid when attributes are added, removed, or reordered, as they only reference the attributes they need. The following example queries for names of employees in the engineering department, referencing only the name and department attributes.

```
Engineer(name:) :-
  Employee(name:, department: "Engineering");
```

As shown, in Logica the named attribute can be used in place of the corresponding variable (where the variable name is assumed to be the same as the named attribute).

Logica allows using named attributes directly as variables when the variable name matches the attribute name, as shown with (employee) “`name:`” above.

Positional arguments are internally treated as named arguments `col0`, `col1`, etc. A predicate always corresponds to a single table regardless of how many arguments are mentioned in a particular rule. When arguments are omitted, Logica does not constrain or project on those columns. For example, if we have `Parent(x, y)` and `Grandparent(x, y)` predicates where `x` is a parent (grandparent) of `y`, and we want to find people who are parents or grandparents, we can write:

```
Invite(x) :- Parent(x) | Grandparent(x);
```

Here `Parent(x)` refers to the same table as `Parent(x, y)`, with the second argument simply left unconstrained.

**Composite Data Types.** Logica supports structured data through lists and records, enabling natural representation of complex relationships common in graph data. Lists in Logica are ordered collections and can contain any type of element. Lists are defined using square bracket notation and Logica supports list iteration and membership testing. The following gives a simple example demonstrating lists and membership checks for an input predicate `Country` containing a list of cities for each country.

```
# Membership test using a list literal
European(city) :- city in ["Paris", "London", "Berlin"];

# Iterate over list elements to create a new relationship
CountryCity(country:, city:) :-
    Country(name: country, cities:), city in cities;
```

Records provide structured access to related data through named fields, similar to objects or structs in other languages. The following example defines two books each containing a nested record and a corresponding query using path expressions to obtain the nested record information.

```
# Define two tuples containing nested records
Book(title: "To Kill a Mockingbird",
      info: {author: "Harper Lee", publication_year: 1960});
Book(title: "1984",
      info: {author: "George Orwell", publication_year: 1949});

# Access record fields using dot notation
RecentBook(title:) :-
    Book(title:, info:),
    info.publication_year > 1950;
```

Records and lists can be arbitrarily nested in Logica, which provides greater flexibility in modeling data as well as for interoperating with external systems and data sources (e.g., that leverage JSON encodings). Lists and records are also useful for representing property graphs in which nodes and edges can be annotated with complex attribute values.

**Aggregation.** Like SQL, Logica supports aggregation operations. Logica provides two aggregation modes: *predicate-level* and *inline*. The following is an example of predicate-level aggregation. For each author that has written a book, the query computes the number of books written as well as the most recent year the author published a book.

```
# Number of books and most recent publication year by author
AuthorStats(author:,
            book_count? += 1,
            latest_year? Max= publication_year) distinct :-
    Books(author:, publication_year:);
```

The `distinct` keyword is used to trigger aggregation in Logica, indicating the desire to create a set of objects. The columns to be aggregated are denoted with `?` after which an aggregation operator is given followed by `=`; other columns are not aggregated. The set is created for keys

## 7:6 Logica-TGD: Transforming Graph Databases Logically

composed of values of non-aggregated columns. Values of aggregated columns are grouped for each key and the aggregation operator is applied to the group. In the example above, the result is one unique row per author. Since multiple books may have the same author, Logica naturally groups these together and applies the aggregation functions (`+=` for counting, `Max=` for latest year) within each group.

Note that whenever a SQL statement has a `GROUP BY` clause, the resulting table has no duplicates, i.e., all rows in the table are *distinct*. In Logica, the `distinct` keyword in the head of the rule serves this same purpose, unifying the concept of sets with aggregation.

Thus while Logica inherently uses multi-set semantics, set semantics organically emerges from aggregation. Denoting a predicate that has no columns aggregated, `distinct` makes it a set, in accordance with aggregation as defined above. Note that similarly in SQL, while there is a `DISTINCT` keyword, the same effect can be achieved via grouping by all columns.

When multiple rules define the same predicate, their contributions are combined via `UNION ALL` and the aggregation operator is then applied over the combined result, which is expressed in SQL as `UNION ALL` wrapped in `GROUP BY`. Logica requires all rules for a predicate to use a consistent aggregation signature, and inconsistent signatures result in a compilation error.

In-line aggregation allows aggregation in the body of the predicate. It uses syntax similar to set-builder notation. The following example uses inline aggregation to compute the total number of special-offer points awarded for each shopping cart where a shopping cart consists of a list of item names and each special offer relates an item name to a record containing its corresponding number of bonus points.

```
CartBonusPoints(cart_id:, total_points:) :-
  ShoppingCart(cart_id:, items:),
  total_points = Sum{
    bonus_points :-
      item_name in items,
      SpecialOffers(item_name:, bonus_points:)
  };
```

In-line aggregation can also construct lists in the body of rules (a form of list comprehension) using the `List` aggregating operator. In the following example, `prices_of_premium_items` is computed assuming that each `item` is a record that has fields `is_premium` and `price`.

```
PremiumCarts(cart_id:, prices_of_premium_items:) :-
  ShoppingCart(cart_id:, items:),
  prices_of_premium_items = List{
    item.price :-
      item in items,
      item.is_premium == true
  };
```

Note that `List` aggregation is non-deterministic: the order of elements in the resulting list depends on race conditions of parallel data processing, just like `array_agg` in PostgreSQL and `list` in DuckDB.

In-line aggregation expressions distinguish two kinds of variables by their scope. Variables that appear both inside and outside the aggregation are *correlated*: their values are fixed by the outer context, just as in SQL correlated subqueries. Variables that appear only inside the aggregation are *existential*: they range over all matching values and are aggregated over. The same scoping rule applies to negation: in  $\sim P(x, y)$ , a variable bound in the outer rule body is correlated, while a variable appearing only inside the negation is existentially quantified. Aggregation expressions can be nested. In this case, a variable mentioned in an enclosing aggregation scope is correlated in the inner one.

**Functional Notation.** Logica bridges declarative logic and functional programming by allowing predicates to be defined and used as functions. Internally, functional values are stored in a special attribute `logica_value`. The syntax  $F(x) = y$  provides a syntactic shorthand for predicate definitions. The following provides an example of the (implicit) functional style provided in Logica and the corresponding (explicit) definition using the `logica_value` attribute.

```
# Defining a square operation in Functional style:
Square(x) = x * x;

# Defining square using an explicit relational definition:
Square(x, logica_value: x * x);
```

When predicates are used functionally, Logica performs semantic “desugaring.” For example, the following example uses the square operation in its functional form followed by the corresponding “desugared” syntax.

```
# Using the square operation (implicitly)
Q(x, Square(x)) :- T(x);

# An equivalent definition with square used in the body of the rule:
Q(x, y) :- T(x), Square(x, logica_value: y);
```

One use of functional predicates is for lookup tables, as shown in the following example.

```
# Set the captial of France and Japan
CountryCapital("France") = "Paris";
CountryCapital("Japan") = "Tokyo";

# Obtain the capital of France
FrenchCapital(CountryCapital("France"));
```

Functional predicates can also be defined using aggregation operators, where the functional value itself is computed by aggregating over related data. The following example computes the total population of each country using the functional value of `CityPopulation` (which gives the population value of a given city) where the `Country` relation consists of countries and their corresponding list of cities.

```
# For each country, compute the total population of its cities:
Population(country) += CityPopulation(city) :-
    Country(name: country, cities:), city in cities;
```

The above example defines `Population` as a functional predicate that returns the total population for any given country by summing the populations of its cities. Note that the `distinct` keyword is omitted here: when the aggregated column is the return value of a functional predicate, the aggregation is clearly visible at the definition site, making `distinct` unnecessary. For predicate-level aggregation with `?`, the aggregated columns may be harder to spot among many columns, so `distinct` serves as an explicit marker that set semantics is intended.

**Negation as Aggregation.** Logica treats logical negation (denoted as  $\sim$ ) not as a primitive operation but as emerging naturally from aggregation. As a simple example, the following rule finds birds that can sing but cannot fly.

## 7:8 Logica-TGD: Transforming Graph Databases Logically

```
SingingNonFlyer(name:) :-  
  Bird(name:), CanSing(name:), ~CanFly(name:);
```

The negation operator is syntactic sugar for aggregation-based non-existence checking. Thus, the above rule is equivalent to the following aggregate query in Logica.

```
SingingNonFlyer(name:) :-  
  Bird(name:), CanSing(name:), Sum{1 :- CanFly(name:)} is null;
```

The expression `Sum{1 :- CanFly(name:)}` counts the number of tuples where the bird can fly. Empty sets yield `null` aggregation results, checked with standard `is null` operators. This approach unifies negation with aggregation principles, simplifying the language's conceptual foundation.

### 2.3 Value Creation

Logica supports string and number manipulation, including arithmetic operations and string concatenation. These operations can be used to create new values, playing the same role as Skolemization. For instance, the following is an example of basic graph reification, where edges become nodes in the given graph.

```
EdgeNode(x, y) = "Edge:" ++ ToString(x) ++ "-to-" ++ ToString(y);  
Edge(x, EdgeNode(x, y)) :- DirectEdge(x, y);  
Edge(EdgeNode(x, y), y) :- DirectEdge(x, y);
```

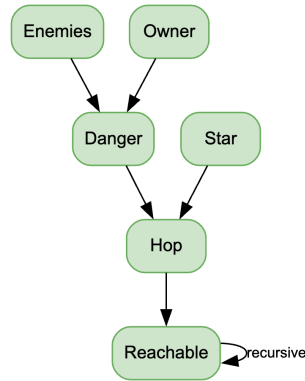
Value creation [2] can easily lead to programs without a finite fixpoint. Defining semantics for such programs is more challenging than for finite-domain Datalog, where well-founded semantics offers an iterative solution and stable model semantics reduces to search within a finite space. With value creation, the search space itself may be infinite, and the question of whether evaluation terminates can be undecidable [19]. In practice, Logica uses a simple iterative evaluation strategy, repeating evaluations up to a fixed threshold or until a stopping condition is met; see Section 2.4 for further details. As one example, the simple reification program above does not use recursion, and thus terminates after a single rule application. The following Logica program, however, does not have a finite fixpoint, and its naive bottom-up evaluation will not terminate. By default, Logica executes programs under a limit of 32 iterations, which can be modified by the user via a language directive.

```
NextHop(x) = "next-" ++ ToString(x);  
Path(x, NextHop(x)) :- Path(x, something);
```

### 2.4 Logica Evaluation Strategy

Understanding how Logica executes graph transformation programs provides essential insight for writing effective code. This section describes Logica's systematic approach to evaluating programs with complex dependencies and recursion.

**Dependency Analysis.** Program evaluation in Logica begins by analyzing predicate dependencies to determine the order of rule evaluation.



■ **Figure 2** Dependencies  $P \rightarrow Q$  between predicates  $P$  and  $Q$ , stating that  $P$  is used in the body of a rule to define  $Q$ , for the example program of Section 2.4. The predicate **Reachable** is recursive and iterated either a fixed number of times or until a user-specified stopping condition is met (where the two different approaches are specified by the user as part of the program configuration).

Consider the following example. The predicate **Reachable** is assumed to hold pairs of **Stars** such that a spaceship can **Hop** from star to star if there does not exist a **Danger** in performing the hop. A **Danger** is present if the **Owners** of the stars are **Enemies** of each other. In this example, **Star**, **Owner**, and **Enemies** are given as inputs to the program (i.e., form the EDB).

```

Danger(a, b) :- Enemies(Owner(a), Owner(b));

Hop(a, b) :- Star(a), Star(b), ~Danger(a, b);

Reachable(a, b) distinct :- Hop(a, b);
Reachable(a, b) distinct :- Reachable(a, x), Reachable(x, b);

```

The **distinct** keyword in the **Reachable** rules ensures set semantics: without it, the multiplicity of each pair would equal the number of paths connecting them, which is infinite in the presence of cycles. The corresponding predicate dependencies of the program are shown on Figure 2

**Strongly Connected Components.** Prior to program execution, Logica uses predicate dependencies to construct a dependency graph from which strongly connected components (SCCs) are identified. Each SCC represents a group of predicates that have dependencies on each other, either directly or indirectly through a graph cycle. If a predicate is not involved in recursion then it forms its own connected component. In the above example, each predicate is a separate connected component and the predicate **Reachable** is the only recursive predicate.

Components in Logica are evaluated in dependency order: a component cannot be processed until all of the components it depends on have been evaluated (following a typical “bottom-up” evaluation strategy). The dependency graph tracks both positive and negative dependencies between predicates. When negation is applied only to predicates from earlier components, the program is stratified and the component-wise evaluation naturally yields the standard stratified semantics (see Section 2.5.2). In the example, the derived predicates **Danger**, **Hop**, and **Reachable** (i.e., forming the IDB) start as empty relations. After the first iteration, **Hop** holds all pair of



## 7:10 Logica-TGD: Transforming Graph Databases Logically

(non-Danger) stars. The second iteration results in **Reachable** holding for all such pairs of stars. In subsequent iterations, **Reachable** becomes “self-sustaining”. We note that components without recursion are evaluated once, as everything that they depend on is already computed.

**Handling Recursion.** When evaluating recursive components, Logica uses an iterative approach:

- **Start** with base cases established by non-recursive rules.
- **Iterate:**
  - Apply all rules – both base and recursive – using the previous iteration’s state computing a new set of facts that hold.
  - Replace the previous state of each relation entirely with those new facts.
  - **Terminate** after a fixed number of iterations (32 by default on DuckDB) or when an explicit stopping condition is met.

This process builds up complete solutions step by step. For reachability, it discovers connections of length 1, then length 2, then length 3, and so on, until all possible connections are found. Also note that fact remains in the state on the next step only if it was re-derived with the facts of the state. Thus if a fact is no longer supported by the state it disappears and thus non-monotonicity is achieved. Hence when we compute distances a derived fact about a distance between two nodes disappears if a shorter distance is found.

**Termination and Control.** Logica continues evaluating predicates until a stopping condition is met. Logica provides two ways to control iteration via arguments of **@Recursive** annotation of the corresponding predicate.

**Fixed number of iterations** can be specified if you know from the size of the graph the number of iterations required. Generally we expect algorithm to converge to a fixed point and thus an upper bound of possibly needed number of iterations can be used.

```
@Recursive(Reachable, 20);
Reachable(a, b) distinct :- Hop(a, b);
Reachable(a, b) distinct :- Reachable(a, x), Reachable(x, b);
```

**Default behavior.** By default when neither the number of iterations nor explicit stopping conditions are specified the iteration proceeds for 32 steps.

Although a small number of iterations, the default has proven useful as an initial starting point for iteration. For example, for many smaller-sized graphs, 32 steps is often sufficient to converge to a fixpoint. Additionally, double-recursion algorithms usually converge with exponential speed [44]. The implementation of reachability with a linear recursion in 32 steps would find paths of length up to 32. However, as shown in the above example, where reachability is implemented through double recursion, the length of the explored path grows exponentially with the number of iterations, and thus, 32 steps would suffice even for very large graphs.

**Explicit stopping conditions** can be specified in Logica. In this case, iteration will continue until the stopping condition is met.

```
@Recursive(Reachable, -1, stop: ReachabilityComplete);
Reachable(a, b) distinct :- Hop(a, b);
Reachable(a, b) distinct :- Reachable(a, x), Reachable(x, b);
ReachabilityComplete() distinct :-
    (Reachable(a, x), Hop(x, b)) => Reachable(a, b);
```

The implication  $P(x, y) \Rightarrow R(x, z)$  is shorthand for  $\neg(P(x, y), \neg R(x, z))$ , meaning “there’s no case where  $P$  holds but  $R$  does not.” Thus `ReachabilityComplete()` holds when every extendable path has already been extended. The explicit stopping condition can be combined with an upper bound on the number of iterations, or the number of iterations can be given as `-1`, meaning iteration proceeds indefinitely until the stopping condition is met.

A natural question is whether Logica could provide automatic fixpoint detection, iterating until the state no longer changes without requiring an explicit stopping predicate. While this would sometimes be convenient, in many cases a user-specified stopping condition remains preferable for efficiency, for instance, comparing fact counts is cheaper than comparing entire relation states. Moreover, automatic fixpoint detection would require comparing relation states between iterations, a substantial addition to the current execution model where everything that is computed corresponds directly to a user-defined predicate. Automatic fixpoint detection is a subject of future work.

**Power and Limitations.** Logica’s expressive power and limitations both stem from its execution strategy. Each iteration fully leverages the SQL ecosystem, e.g., if the backend is a distributed engine like BigQuery, a single iteration executes in parallel across the cluster. As SQL engines continue to improve, Logica benefits automatically. However, Logica does not optimize the flow between iterations; it does not implement techniques such as semi-naïve evaluation that avoid redundant recomputation across iterations. Consequently, Logica performs well when problems can be solved in a small number of iterations. Fortunately, many practical graph problems fall into this category: transitive closure can be computed in logarithmic iterations using squared recursion, and single-source shortest paths on social networks typically converge within six iterations due to the small-world property. Problems requiring tens of thousands of iterations, however, may not be well-suited for Logica. Additionally, Logica does not implement combinatorial search strategies like those found in Answer Set Programming systems such as Clingo, making it a less suitable syntax for solving NP-complete problems that require exploring exponentially many candidate solutions.

## 2.5 Logica Semantics

### 2.5.1 Logica and *While* Semantics

In the previous section, we described Logica’s evaluation strategy. From a theoretical perspective, this strategy aligns with the **while** semantic framework introduced by Abiteboul, Hull, and Vianu [1] for database query languages, including **while**( $\varphi$ ), where iteration continues as long as a logical condition  $\varphi$  is satisfied.

The Logica evaluation loop works similarly but uses an **until** loop, i.e., using a condition of termination rather than a condition of continuation. To execute a **while**( $\varphi$ ) evaluation of a recursive predicate  $P$  in Logica, a `@Recursive( $P$ ,  $-1$ , stop:  $S$ )` directive is given as a program annotation, where a predicate  $S$  defines the user-specified termination condition  $\neg\varphi$ . An example using this annotation (with `ReachabilityComplete` as the stopping predicate) is given in Section 2.4.

While Logica execution follows **while** semantics, data types of Logica are richer than classical Datalog. Notably, Logica supports composite values of unbounded size, including lists and records with arbitrarily nested structures, combined with arbitrary arithmetic operations and comparison operators. These additional features enable Turing-complete computation, which establishes computational equivalence with **while**<sub>N</sub> [1].

### 2.5.2 Special Cases with Simpler Semantics

Within the while semantics framework, different classes of Logica programs admit less expressive semantic characterizations.

**Classical Datalog Semantics.** For programs without aggregation or negation, Logica’s evaluation reduces to standard least-fixpoint semantics. The iterative process within each strongly connected component corresponds exactly to the well-known immediate consequence operator, ensuring compatibility with traditional Datalog results.

**Stratified Datalog Semantics.** A Datalog program is stratified if negation in the rules defining a predicate is applied only to predicates that are outside the same strongly connected component [47]. The component-wise evaluation naturally respects this stratification: each SCC is computed to completion using only positive recursion before proceeding to dependent SCCs that may use negation on the completed results.

**Monotonic programs** refine predicate values in a consistent direction as they iterate. These programs can be characterized by the existence of a partial order on the predicate values (i.e., on the relations themselves) such that each rule application moves values monotonically along this order [26]. In classic Datalog without aggregation, this order is simply set inclusion – rules can only add facts, never remove them, so each iteration produces a relation that contains all previous facts plus potentially new ones. With monotonic aggregation operators (such as  $\text{Min=}$ ,  $\text{Max=}$ , or  $\text{+=}$  with non-negative values), the order is induced by the properties of the specific operator being used. Examples include transitive closure computations and shortest path algorithms.

**Non-monotonic programs** cannot be characterized by such a consistent directional refinement. Values may fluctuate during iteration without a clear monotonic progression visible from the program structure. A classic example is the Win-Move game solver described in Section 3.3, where move evaluations can change as the algorithm refines its analysis of the game tree, and intermediate values may oscillate before converging to the final solution.

Logica is designed to be a practical programming language consistent with the **while** semantics framework. This foundation provides both the expressiveness needed for complex iterative algorithms and the semantic clarity of well-studied formal models.

This generality requires that the programmer reason about execution and ensure that iteration converges in a reasonable number of steps. The programmer must specify either a fixed iteration count or a stopping condition for termination. In practice, a fixed iteration bound often suffices, as the number of steps needed is typically clear from the problem structure.

## 3 Transforming Graphs with Logica

In Logica, as in Prolog and Answer-Set Programming, graphs can be naturally represented: Nodes are denoted using unique identifiers and edges are represented as facts about the nodes. Graphs can be represented as predicates with two or more *positional* arguments. For example, a simple directed graph can be represented as a binary relation  $E(\text{source}, \text{target})$ , where *source* and *target* are nodes connected by a directed edge. Additional graph data, e.g., node or edge attributes, can be expressed using separate predicates or extra positional (in Logica also *named*) arguments. For instance, a graph whose edges are assigned colors can be represented using a ternary predicate and facts of the form  $E(\text{source}, \text{target}, \text{color})$ .

Graph transformations can be implemented by defining new predicates that depend on the original graph: e.g., given a graph with edges  $E$ , a new graph  $E2$  extending  $E$  by adding new edges between nodes that are separated by two “hops” can be derived as follows.

```
E2(x, z) :- E(x, y), E(y, z);
E2(x, y) :- E(x, y);
```

This simple example demonstrates an important aspect of defining graph transformations with logic rules. In native graph transformation languages users define how the graph changes, and edges not involved in the change remain. However, when defining graph transformations logically, we must include rules to *preserve* edges not involved in the transformation (line 2).

### 3.1 Message Passing

Our first example involves the simple passing of a message along the directed edges of a graph, as described in [28].

```
M0(0);                                # start node
# Rule 1: Message initialization.
M(x) :- M=nil, M0(x);
# Rule 2: Message passing.
M(y) :- M(x), E(x, y);
# Rule 3: Message retention.
M(x) :- M(x), ~E(x, y);
```

Fact  $M0(0)$  states that initially the message is at node 0. Rule 1 initializes the message relation  $M$ . It states that node  $x$  has a message  $M$  if it was specified as the initial node in  $M0(x)$ . Condition  $M=nil$  makes the rule fire (trigger) only at the start of the recursive iteration. The message propagates from node  $x$  to its neighbors  $y$  through the Rule 2. Finally, Rule 3 requires the message of a node  $x$  to be retained if  $x$  has no outgoing edges. While the rule  $M(x) :- M(x), \sim E(x, y)$  might seem tautological as it derives a fact from itself, it plays an important role in the context of the iterative semantics (see Section 2.4): each iteration replaces the previous state entirely, so without Rule 3 messages at nodes with no outgoing edges would disappear on the next iteration.

### 3.2 Computing Distances in a Graph

Logica has native support for aggregation. Predicate level aggregation is computed over the specified values of the fields of the defined predicate. The following Logica program computes the minimum distance  $D(x)$  of a node  $x$  from a given **Start** node.

```
# Rule 1: Distance from the Start node is 0.
D(Start()) Min= 0;
# Rule 2: Triangle inequality.
D(y) Min= D(x) + 1 :- E(x,y);
```

**Start()** is used as a *functional predicate*: as discussed in Section 2 Logica relations can be turned to functions by having an additional “special attribute” (named **logica\_value**) to store and access a relation’s functional value, i.e., its value when used syntactically as a function. Here, **Start()** simply returns (a predefined) starting value. Rule 1 assigns the distance 0 to the starting node, where  $D(x)$  is also a functional predicate that returns the (minimum) distance of a node  $x$ . Rule 2 computes the distance to a node  $y$  as *at most* one more than the distance to its predecessor  $x$ . The **Min=** aggregation operator specifies that the minimum distance will be taken among all possible paths.

### 3.3 Solving Win-Move Games

Win-Move is a two-player combinatorial game played on a finite directed graph (representing a board) where nodes represent positions and edges represent moves [42, 15]. After choosing a starting position (node), the two players take turns moving a game pebble along the edges of the graph. A player who cannot move loses the game. We propose to compute the solution to Win-Move games in a new way, inspired by graph transformations. A concise formulation of the game is given by the logic rule:

```
Win(x) :- Move(x,y), ~Win(y).
```

A solution to the game can be computed using the 3-valued well-founded semantics [45]. The moves of the game are given by the predicate `Move`. A game starting at node `x` is objectively won if there exists a (*winning*) move to some `y` that is lost. Then, no matter how an opponent (Player II) moves from `y` (assuming there is an outgoing move to begin with), Player I can always force a win after the opponent has moved. The well-founded model assigns `Win(x)` the value *true*, *false*, and *undefined* if and only if `x` is objectively *won*, *lost*, and *drawn*, respectively. In contrast, the 2-valued *stable model* semantics does not compute the desired game solution: There can be 0, 1, or more stable models, none of which may agree with the correct, well-founded solution. Similarly, in Logica, the above rule cannot be used as-is.

Looking at the problem through a graph-transformation lens, however, we can define a new predicate `W(x,y)`, indicating that the move from `x` to `y` is a winning move. Computation of `W`, from a graph transformation perspective, amounts to selecting a collection of winning moves. In Logica this can be specified with a single rule:

```
W(x,y) :- Move(x,y), (Move(y,z1) => W(z1,z2));
```

The rule says: A move from `x` to `y` is *winning* iff for *every* opponent move from `y` to some `z1` there *exists* a move from `z1` to `z2` for Player I that is again winning! The logical implication `Move(y,z1) => W(z1,z2)` is a shorthand for the nested negation `~(Move(y,z1), ~W(z1,z2))`.

The position values can now be derived from the winning moves `W`. The source and target nodes `x` and `y` of a winning move are won and lost positions, respectively. Positions that are neither won nor lost are drawn:

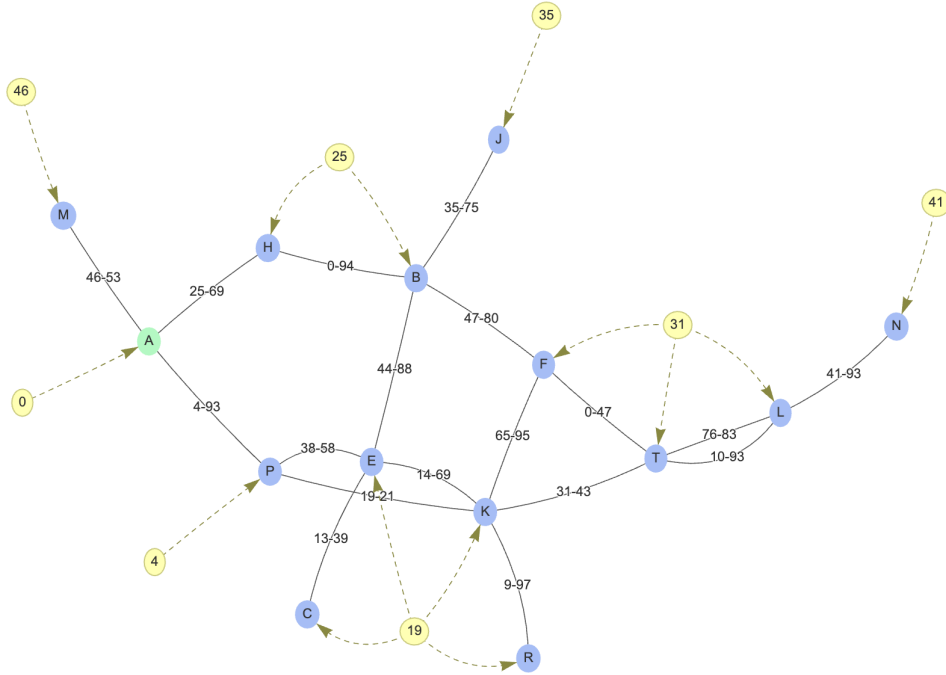
```
Won(x) :- W(x, y);
Lost(y) :- W(x, y);
Drawn(x) :- Position(x), ~Won(x), ~Lost(x);
```

Here, the unary predicate `Position` is defined as the union of the domain and range of the `Move` relation:

```
Position(x) distinct :- x in [a,b], Move(a,b);
```

### 3.4 Finding Paths in Evolving Graphs

Graphs that change over time [25, 8] can be naturally modeled and computed over in Logica, where time can be represented using extra positional arguments, named arguments, or as functional values. As a simple example, assume a graph that changes over time is represented by the relation `E(x,y,t0,t1)` stating that there exists an edge from node `x` to `y` added to the graph between time moments `t0` and `t1`. For simplicity we assume that time starts at moment 0, edges are transitioned instantly, and the start node is given as `Start()`. The following Logica program computes the earliest possible arrival time for each node.



**Figure 3** Visualizing the solution of pathfinding in a dynamic graph. The time of existence is shown as labels on edges; the earliest possible time of arrival is shown via yellow nodes. Section 3.6 shows how such visualizations can be created in Logica.

```
# Rule 1: Starting condition.
Arrival(Start()) Min= 0;
# Rule 2: Traversal of an edge when edge exists.
Arrival(y) Min= Greatest(Arrival(x),t0) :-
    E(x,y,t0,t1), Arrival(x) <= t1;
```

Rule 1 sets the arrival time of the start node to 0 (the assumed initial time). Rule 2 considers edge transitions: if we arrive at the source  $x$  of an edge before the edge expires (given by  $t1$ ) then the arrival time of the edge's target  $y$  is set to the larger (latest) of  $x$ 's arrival time and the time  $t0$  that the edge was added to the graph. Here **Greatest** is a built-in function that returns the larger of its arguments (analogous to SQL's **GREATEST**), as opposed to aggregation operators like **Max=** which aggregate over multiple rows. Once the arrival times are computed, additional rules can be easily added to select specific (time aware) paths of the graph. Figure 3 gives an example of path finding over an evolving graph. An initial graph (with nodes A, B, etc., and colored blue) is given with edges labeled by their start and end times. The start node A is shown in green. The computed arrival times are displayed as additional nodes (in yellow). The graph in Figure 3 was created within Logica as further described below in Section 3.6.

### 3.5 Transitive Reduction of DAGs

The *transitive reduction* [3] of a directed graph is a graph with the fewest possible edges that has the same reachability as the original graph. For directed acyclic graphs (DAGs), the transitive reduction coincides with finding the (unique) *minimum equivalent subgraph*, i.e., a proper subgraph

## 7:16 Logica-TGD: Transforming Graph Databases Logically

with the fewest possible edges needed to maintain the same reachability relation as the original graph. While finding a minimum equivalent subgraph for (arbitrary) graphs with cycles is NP-complete, for DAGs, the problem can be solved in PTIME. The following Logica program computes the transitive reduction  $TR(x,y)$  of a DAG given by the edge relation  $E(x,y)$ . The program first computes the transitive closure and then uses the transitive closure to remove redundant edges.

```
# Rule 1: Transitive closure base case.
TC(x,y) distinct :- E(x,y);
# Rule 2: Transitive closure inductive step.
TC(x,y) distinct :- TC(x,z), TC(z,y);
# Rule 3: Transitive reduction.
TR(x,y) :- E(x,y), ~(E(x,z), TC(z,y));
```

Rules 1 and 2 compute the transitive closure. Rule 3 identifies edges  $E(x,y)$  in the original graph that cannot be bypassed by going to some other node and taking a transitive path from there. These edges are the essential edges needed to maintain reachability and make up the edges of the resulting transitive reduction graph  $TR$ .

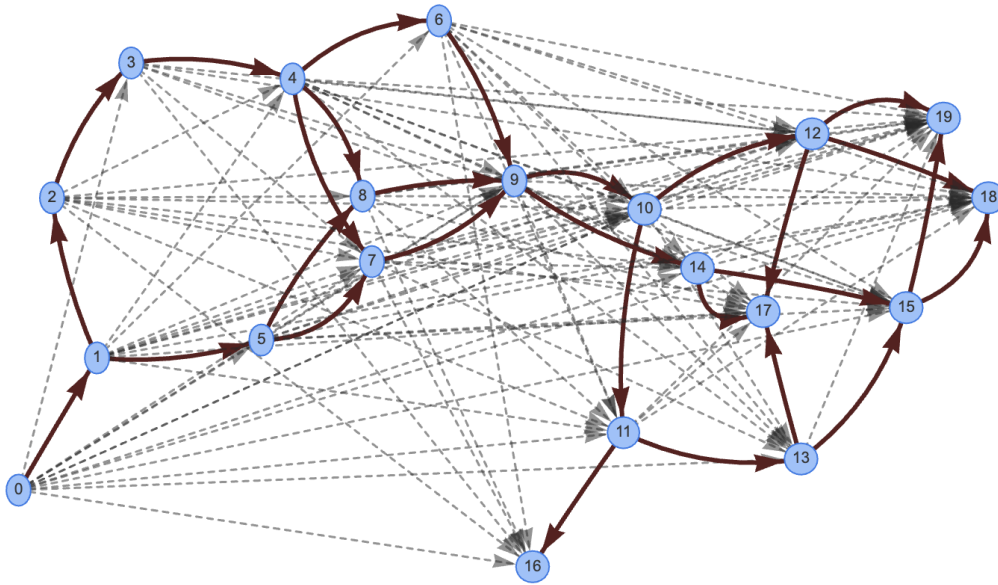
### 3.6 Rendering Graphs with Logica

One of the benefits of Logica is the convenience of rendering graphs directly from predicate definitions. With just a few lines of Logica code and a minimal Python wrapper, it is possible to generate visually appealing and informative graph representations that highlight case-specific attributes. For example, consider the transitive reduction computed in the previous section. Instead of exporting the  $TR$  and  $E$  predicates and configuring their rendering in a separate graphing library, Logica makes it possible to define a predicate that *directly* specifies the visual attributes of the graph. The following relations define graph visualization properties in Logica highlighting the edges in both the original graph and its transitive reduction:

```
R(x, y,
  arrows: "to",
  color? Max= "rgba(40, 40, 40, 0.5)",
  dashes? Min= true,
  width? Max= 2,
  physics? Max= false,
  smooth? Max= false) distinct :- E(x, y);
R(x, y,
  arrows: "to",
  color? Max= "rgba(90, 30, 30, 1.0)",
  dashes? Min= false,
  width? Max= 4,
  physics? Max= true,
  smooth? Max= true) distinct :- TR(x, y);
```

The  $R(x,y,\dots)$  relation defines the edges of the graph and their visual properties. The first rule draws the original edges from  $E(x, y)$  in a light gray, dashed style with a thin line. The second rule draws the transitive reduction edges from  $TR(x, y)$  in bold red, solid lines. The **distinct** keyword triggers aggregation, so that each edge occurs only once and values of attributes such as **color**, **dashes**, etc., are chosen based on whether this edge belongs to the transitive reduction. The visual attributes, e.g., **arrows**, **color**, **dashes**, **width**, **physics**, and **smooth**, are directly embedded within the Logica rules. The question mark in “**color? Max=**” indicates that the **color** can have different values from different rules. The **Max=** indicates that if there are multiple rules that apply to the same edge, the maximum value will be used. The actual rendering is then handled with a Python function call:





■ **Figure 4** Visualizing a transitive reduction with Logica: Original edges are displayed with dashed lines, while edges present in the transitive reduction are highlighted with solid lines. This visualization, generated directly from Logica predicates, shows how the transitive reduction simplifies the graph while preserving reachability.

```
# from logica.common import graph
graph.SimpleGraph(
    R, extra_edges_columns=[
        "arrows", "physics", "dashes", "smooth"],
    edge_color_column="color",
    edge_width_column="width")
```

This code leverages Logica’s `graph` module. The `SimpleGraph` function takes the R predicate, specifies which columns in R represent edge attributes (using `extra_edges_columns`, `edge_color_column`, and `edge_width_column`), and provides overall graph options. The resulting visualization is shown in Figure 4 (where the transitive reduction graph is overlaid over the original graph). The tight integration between logical definitions and graph rendering makes Logica a powerful tool for analyzing and understanding complex relationships in graph data. The ability to define graph properties directly in Logica, rather than relying on external tools, can significantly streamline the data exploration and visualization process.

### 3.7 Graph Condensation

Graph condensation [43] is a technique for simplifying a graph by collapsing strongly connected components (SCCs) into single nodes. This can be useful for visualizing the overall structure of a graph and for performing analyses that are less sensitive to the details within SCCs. In the condensed graph, each node represents an SCC, and an edge exists between two nodes if there is an edge in the original graph between nodes in the corresponding SCCs.

The following Logica program computes the condensation of a graph given by edges defined by the  $E(x, y)$  relation, and where all nodes are defined by a  $\text{Node}(x)$  predicate. We assume that transitive closure is already computed as in Subsection 3.5.

```

# Minimal node ID of the component
# .. is used as the component ID.
CC(x) Min= x :- Node(x);
CC(x) Min= y :- TC(x,y), TC(y,x);
# Compute condensation graph edges.
ECC(CC(x),CC(y)) distinct :-
    E(x,y), CC(x) != CC(y);

```

Once the condensation is computed, it can be easily rendered using Logica’s graph visualization capabilities. To enhance readability, we use the following naming conventions: nodes in the original graph are named directly with their IDs (e.g., “1”, “2”, “3”), while the condensed components are prefixed with “c-” (e.g., “c-1”, “c-2”). The predicate definitions for rendering both the original and condensed graphs simultaneously are as follows:

```

NodeName(x) = ToString(ToInt64(x));
CompName(x) = "c-" ++ ToString(ToInt64(x));

# Edges of the original graph.
Render(NodeName(a), NodeName(b),
    physics: true, arrows: "to",
    dashes: false, smooth: true,
    color: "#33e") :- E(a, b);
# Edges of the condensation.
Render(CompName(x), CompName(y),
    physics: true, arrows: "to",
    dashes: false, smooth: true,
    color: "#33e") :- ECC(x, y);
# Mapping from the graph to
# the condensation.
Render(NodeName(ToInt64(a)), CompName(CC(a)),
    physics: false, arrows: "to",
    dashes: true, smooth: false,
    color: "#888");

```

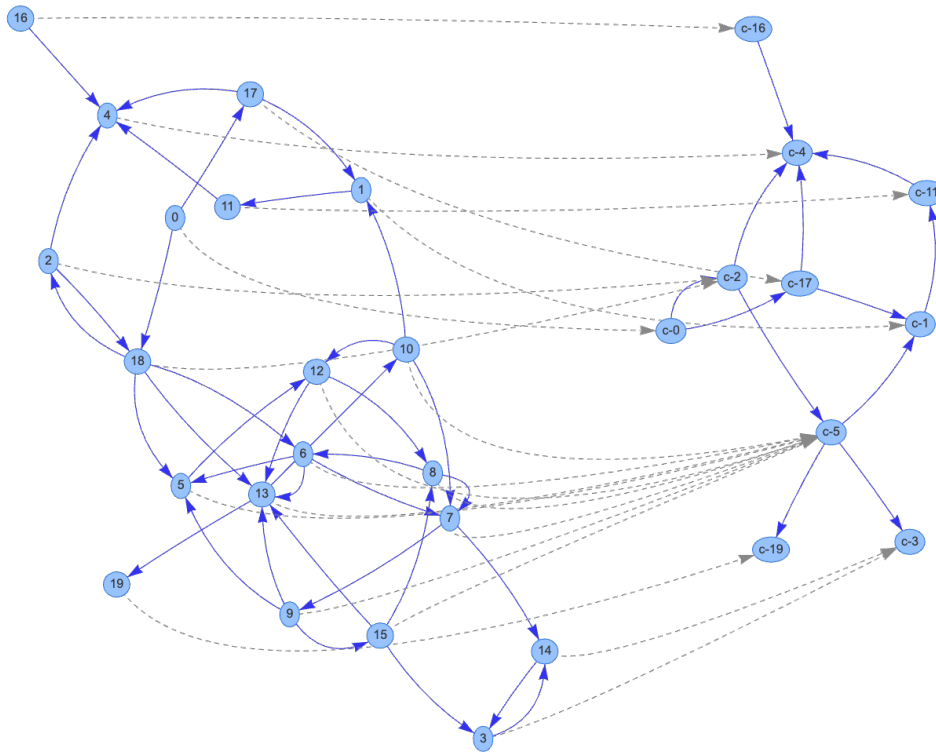
The `NodeName` and `CompName` functions convert node and component IDs to strings with appropriate prefixes. The `Render` predicate then defines the appearance of the graph:

- Edges in the original graph (defined by `E`) are rendered as solid blue lines.
- Edges between components (defined by `ECC`) are also rendered as solid blue lines.
- Dashed gray lines connect each original node to its corresponding component. The physics is disabled between nodes and their condensation components to help with readability.

Figure 5 shows the result of using the above Logica code and the appropriate Python wrapper as described in Subsection 3.6 to view the original and corresponding condensed graph. By combining the power of Logica for defining graph structures and relationships with its seamless graph rendering capabilities, it is possible to gain valuable insights into complex systems through clear and concise visualizations.

### 3.8 Inferring a Taxonomic Tree

Another application of Logica lies in its ability to infer relationships from large datasets and represent them as graphs. Consider the problem of constructing a simplified taxonomic tree for a set of species, showing their evolutionary relationships leading back to common ancestors. Logica can perform this task on massive amounts of data. Here, we aim to build a taxonomic tree for *Homo sapiens* (humans), *Crocodylidae* (crocodiles), *Tyrannosaurus* (T-Rex), and *Columbidae* (pigeons). The input is a knowledge graph of triples  $T(a, b, c)$  and a functional predicate  $L(a) = \ell$  that maps objects  $a$  to human readable labels  $\ell$ . We define `SuperTaxon(item, parent)` as the result



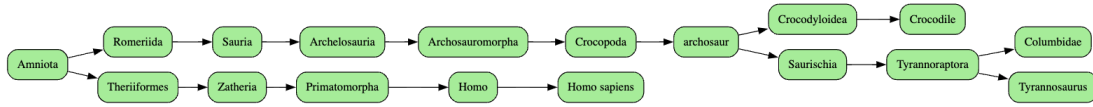
**Figure 5** Graph Condensation visualized with Logica. This figure displays both the original graph and its condensed representation. Solid blue lines represent edges within the original graph and between condensed components. Dashed gray lines connect each node to its corresponding condensed component, illustrating how strongly connected components are collapsed into single nodes. This condensation simplifies the graph while preserving high-level connectivity information.

of the query  $T(item, "P171", parent)$ , which states that *parent* is a direct supertaxon of *item*, and  $TaxonLabel(x) = L(x)$  to give labels restricted to taxons. The following Logica program builds the taxonomic tree.

```
# Use unbounded depth (-1) to find ancestors.
@Recursive(E, -1, stop: FoundCommonAncestor);
E(x, item,
  TaxonLabel(x), TaxonLabel(item)) distinct :-
  SuperTaxon(item, x),
  ItemOfInterest(item) | E(item);
NumRoots() += 1 :- E(x, y), ~E(z, x);
# Stop when common ancestor is found.
FoundCommonAncestor() :- NumRoots() = 1;
```

Figure 6 shows an example output from running the program, where the GraphViz<sup>1</sup> library is used to render the resulting tree. The input for the example consists of the set of statements obtained by a dump of Wikidata, which contains 806M facts defined over 89M objects. When stored using DuckDB, the entire input is 13GB in size. The full recursive search was run on a Google Cloud c2d-standard-32 instance in under 7 seconds via DuckDB with no custom index setup. The majority of the execution time was spent selecting the taxonomy edges from all possible

<sup>1</sup> <https://graphviz.org/>



■ **Figure 6** Inferred Taxonomic Tree for humans, crocodiles, T-Rex, and pigeons, generated by a Logica program from a large Wikidata dump. Edges represent evolutionary relationships, leading from an ancestral species to descendant species. This demonstrates that Logica can operate on very large, real-world knowledge graphs.

relations in Wikidata. Note that the number of taxons returned by the original program is large. The result shown in Figure 6 is only a sample of the obtained taxonomic tree (where the sampling is also performed by Logica; not shown above).

By expressing the problem in Logica, we leverage its ability to efficiently perform (recursive) query processing (e.g., through its compilation to DuckDB). In this case, Logica is used to quickly navigate the complex relationships within a large taxonomic database and generate output that can then be displayed visually through its Python integration (here, using the GraphViz library).

## 4 Performance

■ **Table 1** Benchmark Comparison: Logica vs. Soufflé vs. Nemo vs. DuckPGQ (SQL/PGQ-Cypher).

| Problem                                 | Dataset | Nodes  | Output | Runtime (s)  |              |             |             |
|-----------------------------------------|---------|--------|--------|--------------|--------------|-------------|-------------|
|                                         |         |        |        | Logica       | Soufflé      | Nemo        | DuckPGQ     |
| Transitive Closure                      | G1k     | 1,000  | 1M     | 1.88         | 0.58         | 3.35        | <b>0.35</b> |
|                                         | G2k     | 2,000  | 4M     | 3.62         | <b>1.73</b>  | 14.87       | 2.19        |
|                                         | G3k     | 3,000  | 9M     | 7.23         | <b>3.61</b>  | 36.40       | 7.66        |
|                                         | G4k     | 4,000  | 16M    | 12.56        | <b>6.98</b>  | 70.53       | 18.23       |
|                                         | G5k     | 5,000  | 25M    | 19.56        | <b>10.96</b> | 116.35      | 33.11       |
| Pairwise Distances<br>(Aggregation)     | G1k     | 1,000  | 1M     | <b>2.10</b>  | N/A          | N/A         | 14.46       |
|                                         | G2k     | 2,000  | 4M     | <b>4.34</b>  | N/A          | N/A         | 116.94      |
|                                         | G3k     | 3,000  | 9M     | <b>8.61</b>  | N/A          | N/A         | > 200.00    |
|                                         | G4k     | 4,000  | 16M    | <b>15.26</b> | N/A          | N/A         | > 200.00    |
|                                         | G5k     | 5,000  | 25M    | <b>23.64</b> | N/A          | N/A         | > 200.00    |
| Same Generation<br>(Standard Recursion) | Tree7   | 180    | 18k    | 0.79         | 0.24         | <b>0.02</b> | N/A         |
|                                         | Tree8   | 471    | 107k   | 0.92         | 0.29         | <b>0.11</b> | N/A         |
|                                         | Tree9   | 1,205  | 672k   | 1.61         | <b>0.62</b>  | 0.99        | N/A         |
|                                         | Tree10  | 3,080  | 4.3M   | <b>2.18</b>  | 3.94         | 5.93        | N/A         |
|                                         | Tree11  | 7,781  | 26M    | <b>7.71</b>  | 26.25        | 38.10       | N/A         |
|                                         | Tree12  | 19,504 | 162M   | <b>37.75</b> | 153.46       | > 200.00    | N/A         |
| Same Generation<br>(via Aggregation)    | Tree7   | 180    | 18k    | 1.03         | N/A          | N/A         | <b>0.05</b> |
|                                         | Tree8   | 471    | 107k   | 1.13         | N/A          | N/A         | <b>0.17</b> |
|                                         | Tree9   | 1,205  | 672k   | 1.05         | N/A          | N/A         | <b>0.50</b> |
|                                         | Tree10  | 3,080  | 4.3M   | <b>1.20</b>  | N/A          | N/A         | 5.68        |
|                                         | Tree11  | 7,781  | 26M    | <b>1.43</b>  | N/A          | N/A         | 56.67       |
|                                         | Tree12  | 19,504 | 162M   | <b>2.55</b>  | N/A          | N/A         | > 200.00    |

To help assess the performance of Logica’s approach of compiling recursive queries to iterated SQL, we conducted a series of benchmarks comparing Logica against Soufflé [24], a high-performance Datalog engine written in C++, Nemo [21], a rule engine implemented in Rust, and DuckPGQ<sup>2</sup>, an experimental SQL Property Graph Query (PGQ) implementation and extension for DuckDB.

Logica delegates query execution to an SQL engine, so its performance stems from the capabilities of that underlying engine. The benchmarks in this paper use DuckDB, a highly efficient SQL database that makes full use of modern multi-core hardware for data-parallel queries. Logica’s own contribution is its declarative logical syntax, clarity of which we aimed to illustrate extensively by the examples in Sections 2 and 3.

## 4.1 Experimental Setup

All experiments were performed using a Google Cloud `c2d-standard-32` instance equipped with an AMD EPYC 7B13 (Milan) processor featuring 32 virtual CPUs (16 physical cores with 2 threads per core). The machine was configured with a single NUMA region and operates in 64-bit mode under the `x86_64` architecture. Each core runs at a base clock speed of 2.45 GHz, with a maximum turbo frequency of 3.5 GHz. The instance is provisioned with 128 GB of RAM.

The experimental evaluations utilize two types of synthetic dataset to test different aspects of recursive query evaluation:

1. **Random Graphs ( $G_{Xk}$ ):** These datasets consist of randomly generated graphs ranging from one thousand to five thousand vertices having a fixed average degree of 10, where  $X$  denotes the number vertices in thousands. With an average degree of 10, these graphs form a single large strongly connected component, so the transitive closure output is near-complete at approximately  $N^2$  pairs. Nevertheless, computing it remains a nontrivial task: the solver must discover global reachability through the local edge structure of the graph. The pairwise distances problem over the same graphs is additionally interesting, as the computed distances are varied.
2. **Random Trees ( $Tree_N$ ):** These datasets consist of randomly generated trees of depth  $N$  (ranging from a depth  $N$  between 7 and 12). For every non-leaf layer, each node possesses between one and five children (randomly selected).

While the input graphs are modest in size (up to 5 000 nodes), the computed outputs are large, ranging from millions to over 160 million rows (see Table 1). Thus for these problems, the size of the output is a better indicator of the computational effort than the size of the input.

We note that the reported execution times for Logica include the full end-to-end pipeline: (1) parsing the Logica code; (2) compiling the parsed code to SQL; (3) sending the SQL query to the database engine; and (4) executing the resulting SQL query using the database engine. This pipeline introduces a constant startup overhead that is visible in the smaller-sized datasets. Optimizing this compilation pipeline to minimize startup latency is a subject of future work.

We generated random input data with Logica, so it was populated in the DuckDB database, which was read when benchmarking by Logica and DuckPGQ. Data was also offloaded into CSV to be consumed by Nemo and Soufflé. DuckDB storage is highly optimized so Logica would be spending less time reading data, however by the nature of these benchmarks the output is orders of magnitude larger than input and thus the total reading time is making an insignificant contribution to overall execution time.

All benchmark queries and scripts are available as Jupyter notebooks in the Logica repository.<sup>3</sup>

<sup>2</sup> <https://duckpgq.org/>

<sup>3</sup> <https://github.com/EvgSkv/logica/tree/main/examples/graph/tgdk>

## 4.2 Performance Analysis

We consider four types of computations for evaluating performance: (1) transitive closure; (2) pairwise distances (using aggregation); (3) finding nodes that are part of the same generation (using standard recursion); and (4) same generation via aggregation. The results of the experimental evaluation are summarized in Table 1 and are further discussed below for each type of computation.

### 4.2.1 Transitive Closure

Implementation of Transitive Closure in Logica is discussed in Section 3.5. In the benchmarks we use a linear recursion variant for efficiency:

```
@Recursive(TC, stop: Done);
TC(a, b) distinct :- G(a, b);
TC(a, c) distinct :- TC(a, b), G(b, c);
PrevTC(a, b) :- TC(a, b);
Done() :- TC(a, b) => PrevTC(a, b);
```

The second rule uses linear recursion ( $TC(a, b), G(b, c)$ ) rather than quadratic ( $TC(a, b), TC(b, c)$ ), which is cheaper in practice. The `stop` directive halts iteration once the fixpoint is reached, i.e., when no new pairs are produced.

For pure reachability tasks on the  $G_{Xk}$  datasets, specialized engines show an advantage. DuckPGQ and Soufflé outperform Logica on smaller graphs. However, Logica’s performance remains within the same order of magnitude. As the size of the dataset grows, the compilation overhead becomes negligible in Logica relative to the total runtime, and as shown, Logica times begin to outperform DuckPGQ and move closer towards those of Soufflé.

### 4.2.2 All-Pairs Shortest Paths

All-pairs shortest paths problem is finding the minimum-length path between each pair of nodes, that is finding matrix of distances between all pairs of vertices. Given predicate `E` containing all edges, in Logica this problem is solved as follows.

```
D(x, y) Min= 1 :- E(x, y);
D(x, z) Min= D(x, y) + D(y, z);
```

Recursion stop condition for `D` and `SG` is done analogously to the Transitive Closure program above.

This problem highlights the differences in feature sets and scalability among the three systems. In particular, Soufflé is excluded from this benchmark because it restricts the use of aggregation within recursive cycles (stratification). In the case of DuckPGQ, we observed that the query execution time increased rapidly with dataset size, resulting in timeouts for datasets larger than  $G_{2k}$ . Logica, however, successfully compiled the logic into recursive SQL using standard arithmetic increments. By leveraging the database’s native ability to handle aggregation within recursion, Logica completed the  $G_{5k}$  task in under 24 seconds, whereas the PGQ extension for DuckDB failed to complete within the 200-second time limit.

### 4.2.3 Same Generation

The Same Generation problem is to find all pairs of nodes at the same depth in a tree. This problem can be solved using two fundamentally different approaches, illustrating the tradeoff between compilation overhead and runtime scaling.

**Standard Recursion.** The first approach to computing generations uses pure recursive rules without aggregation. The following is a Logica implementation of Same Generation for the predicate `Parent`, where the implementation in Soufflé is analogous.

```
SG(x, y) :- Parent(p, x), Parent(p, y);
SG(x, y) :- SG(a, b), Parent(a, x), Parent(b, y);
```

As shown, Soufflé dominates on small trees where Logica pays a consistent compilation overhead. However, as the trees grow exponentially with depth, Logica outperforms Soufflé in terms of overall execution time. Soufflé requires dramatically more time on deeper trees while Logica's runtime cost grows much more gradually. Unlike Logica and Soufflé, DuckPGQ is not designed to express this type of query, as shown by N/A's across each of the corresponding benchmarks in Table 1.

**Using Aggregation.** The second approach for computing the Same Generation problem uses aggregation to first find the depth of each node in the tree. The implementation of this approach in Logica is shown below for input predicates `Parent` and `Node`.

```
Gen(x) Min= x :- Node(x);
NextGen(Gen(x)) Min= Gen(y) :- Parent(x, y);
Gen(y) Min= NextGen(Gen(x)) :- Parent(x, y);
SG(x, y) :- Gen(x) = Gen(y);
```

This implementation defines functional predicates `Gen` and `NextGen`. The predicate `Gen` maps each node to the representative of its generation, i.e., the smallest node in the generation. The predicate `NextGen` connects the generation of parent nodes to the generation of their children. These two predicates are defined through mutual recursion using `Min` aggregations such that the generation that a child belongs to is the next generation after the generation of its parent.

To answer the query using aggregation in DuckPGQ, we assume the root of the tree is known, and compute same generations using distances from nodes to the root. However, as in the pairwise-distances problem, Soufflé cannot be used to compute generations since it restricts the use of aggregation through recursive cycles. As shown in Table 1, DuckPGQ performs well on shallow trees, but as tree depth increases, it scales substantially slower than Logica, eventually timing out. Logica's performance remains stable across all tree sizes. Once Logica's execution time is no longer dominated by compilation overhead in the examples, the aggregation-based approach implemented in Logica scales efficiently even on the deeper trees in the benchmark.

### 4.3 Discussion

The evaluation results highlight the benefits of compiling logic programs to SQL and delegating execution to a modern analytical database engine. DuckDB, which serves as Logica's backend in these benchmarks, is designed for efficient parallel execution on a single machine. As the output grows, DuckDB effectively utilizes all available cores – for instance, in the Same Generation problem, Logica computes 26 million output rows (Tree11) in 7.71 seconds, while computing 162 million rows (Tree12) takes 37.75 seconds. The output grew 6× while execution time grew under 5×, demonstrating sub-linear scaling relative to output size. By delegating execution to DuckDB, Logica achieves performance competitive with specialized engines like Soufflé and DuckPGQ.



## 5 State of the Art

Our work is related to three approaches for graph transformation: rule-based, logic-based, and graph query languages. This section reviews the major systems and further examines how these approaches handle challenges related to expressiveness, scalability, and temporal reasoning, compared to Logica.

### 5.1 Rule-Based Graph Transformation Systems

Classical graph transformation systems have developed consistently since the 1970s, when rule-based graph transformation emerged as a research field [10].

The Attributed Graph Grammar system (AGG) [12] represents one of the earliest implementations of the algebraic single-pushout approach [11]. AGG supports the visual editing of graphs and rules, enabling users to define transformations in a visual programming environment. The GGraph-based Object-Oriented VERification (GROOVE) system [18] emphasizes efficient exploration of state spaces comprising generated graphs by applying various search strategies (e.g., depth-first), with built-in model checking and isomorphism recognition. GrGen.NET [22] utilizes generative programming to compile graph rewrite rules into highly optimized .NET code for faster execution speed, proven by the Varró benchmark [46].

However, because these rule-based graph transformation systems prioritize formal verification and model transformation capabilities over computational efficiency, they face fundamental scalability barriers when applied to large-scale graphs. For instance, GROOVE suffers from dramatic growth in state space size as problem complexity increases. Furthermore, these tools function as standalone applications, making it difficult to integrate them with modern data processing pipelines.

### 5.2 Logic-Based Graph Transformation

Logic programming systems also adopt a rule-based approach for graph transformation. By leveraging database optimization techniques, Datalog-based systems show particular promise for scalable graph analytics.

In Section 4 we compared Logica to Soufflé [24] and Nemo [21]. Soufflé compiles Datalog programs to optimized parallel C++ (our benchmarks use its compiled mode), achieving high performance on tasks like transitive closure and reachability, although it does not support aggregation within recursive cycles. Nemo is a recent rule engine implemented in Rust, designed for large-scale reasoning with knowledge graphs and ontologies, capable of ingesting data from various sources including CSV, RDF, and SPARQL endpoints. At present Nemo’s evaluation is single-threaded, which places it at a disadvantage on our multi-core benchmark hardware; nevertheless its Rust implementation is very promising, and once multi-threaded evaluation is added Nemo may well outperform Logica on the problems we study. RecStep [13] achieved performance improvements on graph transformation tasks by building Datalog engines on top of parallel relational database systems. DcDatalog [48] leverages shared-memory multicore architectures to optimize graph query execution time.

Several other Datalog-based systems are relevant to graph processing. RDFox [32] is a high-performance in-memory RDF store with built-in Datalog reasoning, supporting parallel incremental materialization over large knowledge graphs. Socialite [37] extends Datalog with recursive aggregates for efficient graph analytics. Vadalog [4] targets knowledge graph reasoning with Warded Datalog<sup>±</sup>. Flix [29] extends Datalog with user-defined lattices and first-class functions for expressing fixed-point computations.

The declarative nature of these approaches shares conceptual foundations with schema mapping systems, which has been widely used in data integration. Modern schema mapping systems like LogMap, an actively maintained scalable ontology matching system with built-in reasoning engine [23], and CODI [20], which introduces probabilistic-logical alignment into data integration. Most of these systems scale from small to medium scale of graphs, and face challenges with large-scale graph processing.

Logica [41] distinguishes itself by compiling logical queries into existing database systems such as DuckDB, and Google BigQuery. This approach leverages established database optimization techniques for graph transformation while benefiting from continuous database improvements and enabling large-scale graph processing.

### 5.3 Graph Query Languages

Besides logic programming languages for graph processing, there are other graph query languages, which usually have close relationships to graph database systems, for instance, Cypher [16] to Neo4j [30] and DuckPGQ (which performance is also shown in Section 4). These query languages are usually designed for different query needs.

Cypher [16] provides a visual graph matching mechanism for Neo4j, enabling users to express graph patterns using an intuitive syntax while supporting grouping and aggregation. Gremlin [34], part of Apache TinkerPop project, is a functional programming language natively support both imperative and declarative querying for graph traversal. SPARQL [33] serves for querying RDF knowledge graphs. Its ability to federate queries across multiple data sources makes it particularly suitable for semantic web applications.

Recent work by Bonifati et al. [6, 7, 5] addresses declarative graph-to-graph transformations within the Cypher ecosystem, extending GQL/SQL-PGQ foundations with rule-based transformation specifications. While their approach enables property graph transformations directly within graph database backends, Logica takes a complementary path by compiling to SQL, leveraging established database engines for scalability while maintaining full Datalog expressiveness for recursive graph computations.

## 6 Conclusion

We have presented a novel approach to implementing graph transformations using Logica, a free and open-source logic programming language designed for parallel databases. While Logica is typically used for data processing and analysis, we have shown that it can also effectively be used for graph transformations. By leveraging Logica’s ability to process large datasets in a parallel and efficient manner, graph transformations can be performed efficiently. By showcasing several examples, including the Win-Move problem, dynamic path finding, transitive reduction, graph condensation, and graph querying, we aimed to demonstrate that graph transformation problems can be expressed in an intuitive and efficient manner. The examples show how describing graph transformations using logic rules can lead to concise declarative specifications that also improve readability. For example, our approach models time-varying graphs in a principled manner, an issue that can be challenging in classical graph transformation languages.

In future work, we plan to explore more complex graph transformation patterns, including rewritings that may require solving NP-hard problems. We also plan to benchmark our approach on large complex reasoning problems, which would be valuable for deeper understanding of Logica’s performance on real-world datasets.

We hope that Logica, with its ability to leverage underlying SQL engines, can make graph transformations a more widely accessible and practical approach for managing complex and dynamic data.

## References

- 1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- 2 Serge Abiteboul and Victor Vianu. Datalog extensions for database queries and updates. *J. Comput. Syst. Sci.*, 43(1):62–124, August 1991. doi:10.1016/0022-0000(91)90032-Z.
- 3 A. V. Aho, M. R. Garey, and J. D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972. doi:10.1137/0201008.
- 4 Luigi Bellomarini, Emanuel Sallinger, and Georg Gottlob. The Vadalog System: Datalog-based Reasoning for Knowledge Graphs. *VLDB*, 11(9):975–987, 2018. doi:10.14778/3213880.3213888.
- 5 Angela Bonifati. Versatile property graph transformations. *Proc. VLDB Endow.*, 18(12):5516–5526, August 2025. doi:10.14778/3750601.3760517.
- 6 Angela Bonifati, Filip Murlak, and Yann Ramusat. Transforming property graphs. *Proc. VLDB Endow.*, 17(11):2906–2918, July 2024. doi:10.14778/3681954.3681972.
- 7 Angela Bonifati, Yann Ramusat, Filip Murlak, Amela Fejza, and Rachid Echahed. Dtgraph: Declarative transformations of property graphs. *Proc. VLDB Endow.*, 17(12):4265–4268, August 2024. doi:10.14778/3685800.3685851.
- 8 Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. *Algorithmica*, 83(9):2754–2802, September 2021. doi:10.1007/s00453-021-00831-w.
- 9 Francisco de la Parra and Thomas Dean. Survey of graph rewriting applied to model transformations. In *International Conference on Model-Driven Engineering and Software Development*, pages 431–441, 2014. doi:10.5220/0004731504310441.
- 10 Hartmut Ehrig, Claudia Ermel, Ulrike Golas, and Frank Hermann. Graph and model transformation. *Monographs in Theoretical Computer Science*. Springer, page 17, 2015.
- 11 Hartmut Ehrig, Reiko Heckel, Martin Korff, Michael Löwe, Leila Ribeiro, Annika Wagner, and Andrea Corradini. Algebraic approaches to graph transformation—part ii: Single pushout approach and comparison with double pushout approach. In *Handbook Of Graph Grammars And Computing By Graph Transformation: Volume 1: Foundations*, pages 247–312. World Scientific, 1997.
- 12 Claudia Ermel, Michael Rudolf, and Gabriele Taentzer. The agg approach: Language and environment. In *Handbook of Graph Grammars and Computing by Graph Transformation: Volume 2: Applications, Languages and Tools*, pages 551–603. World Scientific, 1999.
- 13 Zhiwei Fan, Jianqiao Zhu, Zuyu Zhang, Aws Albarghouthi, Paraschos Koutris, and Jignesh M Patel. Scaling-up in-memory datalog processing: Observations and techniques. *Proceedings of the VLDB Endowment*, 12(6), 2019. doi:10.14778/3311880.3311886.
- 14 Maribel Fernández, Hélène Kirchner, Bruno Pinard, and Jason Vallet. Labelled graph rewriting meets social networks. In *International Workshop on Rewriting Logic and Its Applications*, volume 9942 of *LNCS*, pages 1–25. Springer, 2016. doi:10.1007/978-3-319-44802-2\_1.
- 15 Jörg Flum, Max Kubierschky, and Bertram Ludäscher. Total and partial well-founded datalog coincide. In *Intl. Conf. on Database Theory (ICDT)*, LNCS 1186, pages 113–124. Springer, 1997. doi:10.1007/3-540-62222-5\_40.
- 16 Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindäcker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 international conference on management of data*, pages 1433–1445, 2018. doi:10.1145/3183713.3190657.
- 17 Michael Gelfond. Answer sets. In Frank van Harmelen, Vladimir Lifschitz, and Bruce W. Porter, editors, *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 285–316. Elsevier, 2008. doi:10.1016/S1574-6526(07)03007-6.
- 18 Amir Hossein Ghamarian, Maarten de Mol, Arend Rensink, Eduardo Zambon, and Maria Zimakova. Modelling and analysis using groove. *International journal on software tools for technology transfer*, 14(1):15–40, 2012. doi:10.1007/S10009-011-0186-X.
- 19 Tomasz Gogacz and Jerzy Marcinkowski. All instances termination of chase is undecidable. In *Proc. ICALP*, volume 8573 of *LNCS*, pages 293–304. Springer, 2014. doi:10.1007/978-3-662-43951-7\_25.
- 20 Jakob Huber, Timo Szttyler, Jan Noessner, and Christian Meilicke. Codi: Combinatorial optimization for data integration—results for oaei 2011. *Ontology Matching*, 134, 2011.
- 21 Alex Ivliev, Stefan Ellmauthaler, Lukas Gerlach, Maximilian Marx, Matthias Meißner, Simon Meusel, and Markus Krötzsch. Nemo: First glimpse of a new rule engine. In *Proceedings of the 39th International Conference on Logic Programming (ICLP 2023)*, 2023.
- 22 Edgar Jakumeit, Sebastian Buchwald, and Moritz Kroll. Grgen. net: The expressive, convenient and fast graph rewrite system. *International Journal on Software Tools for Technology Transfer*, 12(3):263–271, 2010. doi:10.1007/S10009-010-0148-8.
- 23 Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. Logmap: Logic-based and scalable ontology matching. In *International Semantic Web Conference*, pages 273–288. Springer, 2011. doi:10.1007/978-3-642-25073-6\_18.
- 24 Herbert Jordan, Bernhard Scholz, and Pavle Subotić. Soufflé: On Synthesis of Program Analyzers. In *Computer Aided Verification*, LNCS, pages 422–430. Springer, 2016. doi:10.1007/978-3-319-41540-6\_23.
- 25 David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *J. Comput. Syst. Sci.*, 64(4):820–842, June 2002. doi:10.1006/jcss.2002.1829.

- 26 Mahmoud Abo Khamis, Hung Q. Ngo, Reinhard Pichler, Dan Suciu, and Yisu Remy Wang. Datalog in wonderland. *SIGMOD Rec.*, 51(2):6–17, July 2022. doi:10.1145/3552490.3552492.
- 27 Tim Kräuter, Adrian Rutle, Harald König, and Yngve Lamo. Formalization and analysis of bpmn using graph transformation systems. In *International Conference on Graph Transformation*, pages 204–222, 2023. doi:10.1007/978-3-031-36709-0\_11.
- 28 Barbara König. Graph transformation meets logic. GReTA - Graph Transformation Theory and Applications Symposium, 2020. URL: <https://www.irif.fr/~greta/event/nov20th2020-knig/>.
- 29 Magnus Madsen, Ming-Ho Yee, and Ondřej Lhoták. From Datalog to Flix: A declarative language for fixed points on lattices. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2016)*, 2016.
- 30 Justin J Miller. Graph database applications and concepts with neo4j. In *Proceedings of the southern association for information systems conference, Atlanta, GA, USA*, volume 2324 (36), pages 141–147, 2013.
- 31 Manfred Nagl. Graph rewriting systems and their application in biology. In *Mathematical Models in Medicine*, pages 135–156. Springer, 1976.
- 32 Yavor Nenov, Robert Piro, Boris Motik, Ian Horrocks, Zhe Wu, and Jay Banerjee. RD-Fox: A highly-scalable RDF store. In *The Semantic Web – ISWC 2015, Lecture Notes in Computer Science*. Springer, 2015. doi:10.1007/978-3-319-25010-6\_1.
- 33 Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. *ACM Transactions on Database Systems (TODS)*, 34(3):1–45, 2009. doi:10.1145/1567274.1567278.
- 34 Marko A Rodriguez. The gremlin graph traversal machine and language (invited talk). In *Proceedings of the 15th symposium on database programming languages*, pages 1–10, 2015. doi:10.1145/2815072.2815073.
- 35 Francesc Rosselló and Gabriel Valiente. *Graph Transformation in Molecular Biology*, pages 116–133. Springer, 2005. doi:10.1007/978-3-540-31847-7\_7.
- 36 Francesc Rosselló and Gabriel Valiente. Chemical graphs, chemical reaction graphs, and chemical graph transformation. *Electronic Notes in Theoretical Computer Science*, 127(1):157–166, 2005. Proceedings of the International Workshop on Graph-Based Tools. URL: <https://www.sciencedirect.com/science/article/pii/S157106610500112X>.
- 37 Jiwon Seo, Stephen Guo, and Monica S. Lam. Socialite: Datalog extensions for efficient social network analysis. In *Proceedings of the 29th IEEE International Conference on Data Engineering (ICDE 2013)*, pages 278–289, 2013. doi:10.1109/ICDE.2013.6544832.
- 38 Evgeny Skvortsov. Logica project, 2023. URL: <https://logica.dev/>.
- 39 Evgeny Skvortsov, Yilin Xia, Bertram Ludäscher, and Shawn Bowers. Logica tgdk: Graph transformation, December 2025. URL: <https://tinyurl.com/LogicaGraphTransformation>.
- 40 Evgeny S. Skvortsov, Yilin Xia, Shawn Bowers, and Bertram Ludäscher. The logica system: Elevating SQL databases to declarative data science engines. In *International Workshop on the Resurgence of Datalog in Academia and Industry (Datalog-2.0)*, volume 3801 of *CEUR Workshop Proceedings*, pages 69–73, 2024. URL: <https://ceur-ws.org/Vol-3801/short5.pdf>.
- 41 Evgeny S. Skvortsov, Yilin Xia, and Bertram Ludäscher. Logica: Declarative data science for mere mortals. In *International Conference on Extending Database Technology*, pages 842–845, 2024. doi:10.48786/edbt.2024.84.
- 42 Cedric A.B. Smith. Graphs and composite games. *Journal of Combinatorial Theory*, 1(1):51–81, June 1966.
- 43 Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. doi:10.1137/0201010.
- 44 Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems: Volume II: The New Technologies*. W. H. Freeman & Co., USA, 1990.
- 45 Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The Well-founded Semantics for General Logic Programs. *Journal of the ACM*, 38(3):619–649, July 1991. doi:10.1145/116825.116838.
- 46 Gergely Varró, Andy Schurr, and Dániel Varró. Benchmarking for graph transformation. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, pages 79–88. IEEE, 2005.
- 47 Victor Vianu. Rule-based languages. *Annals of Mathematics and Artificial Intelligence*, 19(1):215–259, March 1997. doi:10.1023/A:1018907806177.
- 48 Jiacheng Wu, Jin Wang, and Carlo Zaniolo. Optimizing parallel recursive datalog evaluation on multicore machines. In *Proceedings of the 2022 International Conference on Management of Data*, pages 1433–1446, 2022. doi:10.1145/3514221.3517853.



# GeoSPARQL and SPARQL Benchmarking with GeoRDFBench Framework

Theofilos Ioannidis ✉ 🏠 

Dpt. of Informatics and Telecommunications, National and Kapodistrian University of Athens, Greece

Nikos Mamoulis ✉ 🏠 

Dpt. of Computer Science and Engineering, University of Ioannina, Greece

Manolis Koubarakis ✉ 🏠 

Dpt. of Informatics and Telecommunications, National and Kapodistrian University of Athens, Greece

---

## Abstract

We present the GeoRDFBench framework, whose purpose is to assist and streamline the benchmarking of geospatial semantic stores. We identify and formally define all benchmark components, extend them to represent their geospatial aspects, allow for the automatic mapping of datasets to graphs, provide a specialization hierarchy of queryset types for micro or macro experimental scenarios, even for modeling dynamically generated queries. Queries may define their expected resultset to enable automatic accuracy verification. Experiment behavior and execution logic is controlled by the execution specification, which dictates the action (run experiment or print ground queries) to take, the number of repetitions per execution type (cold, warm, continuous cold), the query repetition and experiment timeouts, the delay period before clearing caches, the aggregating function for reporting execution times, and the policy to follow upon cold execution time out. We decouple these declarative benchmark specifications from the framework's execution engine and serialize them as JSON files; this way, we increase their reuse (instantiation through

deserialization), experiment reproducibility and dissemination. Furthermore, these JSON specification libraries can be served remotely by a configurable REST API endpoint. We also model the Geospatial RDF store optional application and database server modules and manage their life-cycle (start, stop, restart) during experiment execution to achieve ideal cold cache query executions. In addition, we unify by generalization the repository and connection functionalities of the three most common RDF framework Java APIs offered by RDF stores: OpenRDF Sesame, Eclipse RDF4J and Apache Jena. At the same time GeoRDFBench allows queryset filtering, automatic system-dependent query namespace prefix generation and query rewriting when non GeoSPARQL spatial vocabularies are used. We provide for a quick learning start by implementing several geospatial RDF stores as separate runtime-dependent modules with repository generation and experiment execution scripts. RDF modules include: RDF4J with and without Lucene, GraphDB, Stardog, Strabon, OpenLink Virtuoso and Jena GeoSPARQL.

**2012 ACM Subject Classification** Information systems → Spatial-temporal systems; Information systems → Semantic web description languages

**Keywords and phrases** geospatial, semantic, benchmarking, framework

**Digital Object Identifier** 10.4230/TGDK.4.2.8

**Category** Resource

**Supplementary Material**

*Software (GeoRDFBench Framework, code):* <https://github.com/tioannid/geordfbench> [31]  
archived at `swb:1:dir:972135253c56ac42c3face1b01ab2ad360acea96`

*Service (GeoRDFBench Framework, web site):* <https://geordfbench.di.uoa.gr/> [32]

*Other (Paper Benchmark Results, Github):* [https://github.com/tioannid/geordfbench\\_results](https://github.com/tioannid/geordfbench_results) [36]  
archived at `swb:1:dir:cbc8f90f62d28e3d3136adc697cbe936dec01bf1`

*Other (Paper Benchmark Results, Zenodo):* <https://doi.org/10.5281/zenodo.15349539> [35]

*Software (GeoRDFBench Samples, code):* [https://github.com/tioannid/geordfbench\\_samples](https://github.com/tioannid/geordfbench_samples) [33]  
archived at `swb:1:dir:57f7a454372a393491bfed1715859b690acb5eee`

*Dataset (Geographica 2, dataset):* <https://doi.org/10.5281/zenodo.13283414> [34]



© Theofilos Ioannidis, Nikos Mamoulis, and Manolis Koubarakis;  
licensed under Creative Commons License CC-BY 4.0

Transactions on Graph Data and Knowledge, Vol. 4, Issue 2, Article No. 8, pp. 8:1–8:50



Transactions on Graph Data and Knowledge

TGDK Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



Received 2025-07-02 Accepted 2026-06-22 Published 2026-09-03

**Editors** Stefania Dumbrava, George Fletcher, Matteo Lissandrini, and Riccardo Tommasini

**Special Issue** Data Management for (Knowledge) Graphs

**Generative AI Declaration** No form of AI, generative or otherwise, has been used in any phase or step of design/development/testing/editing/formatting/etc., neither for the software presented in the paper not for the paper itself.

## 1 Introduction

Many projects [12, 15, 16] have shown that spatial and temporal aspects of Linked Open Data (LOD) are as important and critical, as thematic information, in order to guide decision making [49]. The number of graph database systems with varying degrees of spatial support have steadily increased for the last years [21, 46, 9, 7, 64, 37, 43, 19, 39, 2, 5] and their scalability has been tested for managing very large LOD datasets such as in the DBLP knowledge graph and respective SPARQL endpoint [1]. Selecting the most suitable system requires *evaluating the most recent versions of available systems on accessible infrastructure within the defined budget, with the desired queryloads run against datasets of appropriate size and content*.

There have been many efforts on automating some of the arduous and repetitive tasks of benchmarking. Parametric ontology-based synthetic generators [27, 10, 50, 25, 20] have assisted in creating datasets of desired size and attributes, while log-mining techniques [50, 74, 20] helped creating more application specific querysets. Benchmarking cloud platforms featuring distributed file systems, containerization technologies and intuitive web UIs have allowed reuse of implemented systems and workloads and ease of management [70]. In all cases however, the benchmark researcher has not been spared the effort to deal with system configuration and optimization, spatial indexing setup, detailed query execution, exception handling, and learning required technology stacks and platform APIs.

### Motivation for this work

Our geospatial benchmark experience on single node and Spark-based distributed RDF stores along with synthetic data and query generation [37, 9, 25] has led us to believe that the geospatial semantic store research area would greatly benefit from the introduction *of a lean but extensible geospatial benchmarking framework* with three main objectives.

First, it should *conceptualize and formalize geospatial semantic benchmarks and their execution environment making this knowledge as reusable as possible*, to enable users creating new benchmarks, modify existing ones in as fast, credible and repeatable way as possible.

Secondly, it should *model the RDF stores' architecture, the client RDF Java providers they offer and, on behalf of the system evaluators, implement as much of the configuration, optimization, module administration, data loading and query execution logic as possible* in order to minimize the effort of adding new RDF stores or upgrading existing ones for testing through the framework.

Thirdly, *the framework should be extensible*, meaning that, the benchmark components, such as querysets, datasets, execution model, the execution environment components, such as host, operating system, reporting sink, the RDF stores' architecture modules such as application and database servers and the client RDF Java providers should provide all commonly identified functionality and structural information but can be extended if needed in order to include new subclasses with modified behavior.

Our work, at this stage, focuses on benchmarking single node GeoSPARQL/SPARQL query engines through the console. It makes it easy to integrate new or updated RDF stores, provides a model for declaratively describing a benchmark's structure, behavior and its host's environment. It



helps mapping existing benchmarks or designing new ones, it is aware of and assists with the spatial aspects of systems and benchmark workloads and provides support for GeoSPARQL/SPARQL query execution exception handling. It also allows parallel experiment execution of implemented stores in the same node with or without containers<sup>1</sup>. Perpendicular to our direction, HOBBIT [70] provides a generalized architecture with message bus connected containers for benchmarking the entire Linked Data (LD) life cycle and non LD workloads and systems. However this generality also forces it to be completely agnostic and therefore provide no assistance to its users for systems' integration, dealing with the spatial aspects of the benchmark workloads or handling the specific GeoSPARQL/SPARQL query language structure, features and execution issues.

To the best of our knowledge, there is no previous work that combines a substantial part of the following features that are also the core contributions of our framework:

1. It features a runtime, which *abstracts and implements specification hierarchies for components required to setup and run an experiment on a geospatial RDF store*. Components include: *datasets, querysets, experiment execution, workloads, logging specifications, report sinks, operating systems and hosts*. The runtime contains the *JSON generator whose API enables the creation of serialized versions of all benchmark component specifications*. The framework comes pre-bundled with a *library of JSON specifications which include the Geographica 2 [37] benchmark's components, a PostgreSQL and an H2 [51] embedded implementation of the JDBC report sink and hosts with Linux and Windows operating systems*. The REST API endpoint module promotes a more distributed approach to the framework's operation, as JSON specifications can also be served remotely as URL calls.
2. The runtime *abstracts and generalizes repository connection functionality from the three best known RDF framework APIs: (i) OpenRDF Sesame, (ii) Eclipse RDF4J and (iii) Apache Jena*. At the same time, *it explicitly models the RDF store application and back-end modules by embedding their life-cycle management in the experiment execution*. It combines the above by *implementing class hierarchies of geospatial enabled RDF stores according to their basic module architecture and the RDF framework APIs they support*.
3. The framework *provides example implementations for geospatial RDF stores of different architectures and different RDF framework APIs*. These come in the form of Maven modules<sup>2</sup> and the list includes: *(i) RDF4J [67] with or without Lucene SAIL [68], (ii) GraphDB [57], (iii) Stardog [75], (iv) Strabon [46], (v) Virtuoso [21], (vi) Jena GeoSPARQL [61]*. These modules can act as *templates for other stores with similar architecture and in most cases the required code effort is trivial* as most of the heavy lifting has been pulled upwards in the class hierarchies of the runtime.
4. The runtime experiment executor features a *single experiment execution loop independent of the store architecture and RDF framework API used*. It enables *automatic result accuracy verification* for workloads that have embedded an expected resultset, while query execution accuracy results are persisted with other statistics and provides a *query translation hook*. The executor features *programmable timeout* and a *fine grained error handling* mechanism during each one of the query execution phases, which allows it to *measure results separately from scan errors*<sup>3</sup> and *minimize the execution abort scenarios*. It features *query result sampling in experiment logs* for verification and debugging purposes. It allows *statistics customization* and *synchronous or deferred experiment results' persistence* to a user-defined report sink.

<sup>1</sup> GeoRDFBench's site includes containerized images for demonstration purposes.

<sup>2</sup> GeoRDFBench framework is a Java Maven multi-module POM project.

<sup>3</sup> Usually due to invalid geometries or unsupported operators.

5. The framework can also be used for *SPARQL benchmarks*. LUBM [27] benchmark’s workload component is included in the JSON specification library.

The organization of the rest of the paper is as follows. Section 2 discusses related work. A GeoSPARQL/SPARQL benchmarking framework’s tasks and comparison criteria are detailed in Section 3. Section 4 presents the high level architecture of the framework while Section 5 explains GeoRDFBench’s runtime. Showcasing the JSON generator API follows in Section 6. Section 7 shows an evaluation of the framework and Section 8 compares GeoRDFBench with LITMUS, IGUANA benchmarking frameworks, the HOBBIT platform and the Geographica 2 benchmark. Finally, Section 9 presents conclusions and future work. Complementary and supporting information with code listings, benchmark logs and the experiment execution algorithm is provided in a list of appendices.

## 2 Related Work and Background

In this section, we present related work on graph and geospatial graph store categories, architecture, evaluation criteria and benchmarking.

### 2.1 Graph Store Categories

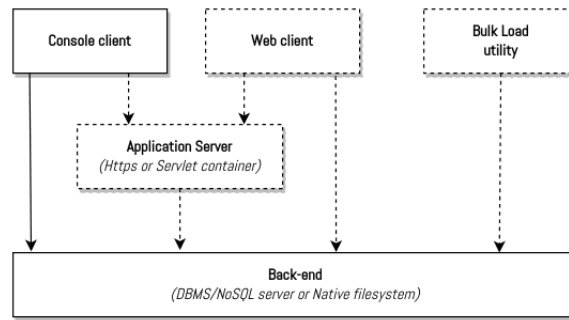
In general, Graph stores follow either the Resource Description Framework (RDF) or the Labeled Property Graph (LPG) approach. RDF as a data model has good expressivity, while featuring a standardized declarative query language SPARQL [80] and a standardized spatial vocabulary GeoSPARQL [65]. LPG stores, on the other hand, excel in graph traversal and path search for analytics and machine learning. But, until recently, they lacked standardization as there are several data models and languages from high-profile vendors and institutions, such as Neo4j’s Cypher [24], Apache TinkerPop Gremlin [71], the Oracle supported PGQL [79] and G-Core [3] from the Linked Data Benchmark Council (LDBC). As of April 2024, the first edition of the Graph Query Language (GQL) standard [38] is officially published and hopefully will allow language inter-operation with SPARQL, yet no standard spatial vocabulary has been proposed. Another issue is that some of these languages are declarative while others are procedural basically requiring expert knowledge specific for each system to efficiently query stored data. Neo4j’s proprietary procedural Cypher language provides native support only for POINT geometries <sup>4</sup> and only through the external plugin Neo4j Spatial [53] support is extended to other geometries. This plugin aligns closely with the concepts found in spatial database systems, such as the layer concept found in Shapefile and Open Street Map geographic data file formats and efficiently querying data require proficient and explicit procedural manipulation of all relevant functionality.

The interoperability of RDF and Property Graphs has been studied in [4] and the G2GML mapping language between the two models has been proposed in [13]. The RDF community on the other hand, also realizing that triples’ annotation with metadata, such as edge properties, can lead to data size blow-up and unnecessarily complex queries, defined the RDF\*[28] data model and the SPARQL\* language, which implement this sought out LPG feature and promote interoperability in the graph community. Neo4j’s neosemantic (n10s) [54] toolkit offers support for RDF data and its related vocabularies, but unfortunately not the GeoSPARQL vocabulary.

To conclude, at the time of writing of this paper, benchmarking geospatial graph databases with complex geometries support, in a declarative manner, is limited to the RDF model.

---

<sup>4</sup> <https://neo4j.com/docs/cypher-manual/current/values-and-types/spatial/>



■ **Figure 1** Geospatial Graph Store Architectures.

## 2.2 Geospatial Graph Store Architecture

The available geospatial graph stores feature a 3-tier architecture with standard and optional components. Their architecture is similar to that in Figure 1 which shows their standard and optional components.

All stores have some front-end application, usually a terminal or web-based **console**, through which the user can create repositories, load datasets to and run queries against them. Depending on the system, this console may communicate directly with the back-end or may relay requests to an application module acting as a proxy. The **application module** is usually an HTTPS server or servlet container, such as Nginx or Eclipse Jetty, with multi-user, load balancing and connection reuse as general capabilities. It also allows centralized semantic store configuration with sane default values for all unconfigured user sessions. Due to the large size of linked datasets, many systems, apart from the console embedded ingestion utilities, they offer dedicated **bulk loading utilities** which can speed up the import step. Such an example is the Preload and LoadRDF tools in Ontotext’s GraphDB.

For the **back-end module**, storage type and indexing methods are the usual differentiating factors. Some stores, mostly research-oriented and especially early ones, employ rigid architectures based on specific implementation recipes. For example, Parliament [7] uses the embedded key-value database Berkeley DB with a standard R-tree spatial index, Strabon [46] uses PostgreSQL and PostGIS with an R-tree over GiST [29] spatial index, uSeekM follows the same path but for spatial information only and native file storage is used for storing and managing thematic information with B+ trees, while Oracle Spatial And Graph [60] uses an R-tree spatial index on top of its proprietary industry leading RDBMS solution. More contemporary market-driven stores offer elastic architectures which effectively decouple modules from specific implementation choices by offering many compatible alternatives for each one of them. For example, Virtuoso offers virtual graphs over many well-known data and file formats, such as Excel, XML files and RDBMS sources.

## 2.3 Geospatial Graph Store Evaluation Criteria

The growth rate of LOD sizes questions the ability of graph databases to persist this big data, while at the same time it poses a critical challenge to the performance of these stores under query loads of interest [45, 9]. For spatial data, we face an additional challenge which is the approximate matching supported by the precision parameter of common spatial indexing algorithms, such as quad [23] and geohash [81] prefix trees, which basically controls how many results will match a spatial filter. Better accuracy requires more index storage and brings a storage and performance penalty, so most systems try to balance between the two. The spatial index algorithm and precision are either fixed for the store e.g., RDF4J, defined upon database creation e.g., Stardog

or dynamically defined even after database creation e.g., GraphDB. Setting up a system with lower precision, has the benefit of reduced storage size, bulk load time and query execution times. Therefore, we have 4 important check points that a geospatial semantic benchmark should measure when testing spatially enabled stores: (i) *equality of spatial index precision* for all systems under test, (ii) *bulk load ability* for huge dataset sizes, (iii) *query execution performance* for various query loads and (iv) *query execution accuracy*. The most valid query execution accuracy test is comparing the query resultset against the expected resultset (*gold standard*), as it is done in the Evaluation Storage module of HOBBIT. However, storage requirements for persisting large gold standard resultsets make this impractical as a general approach. Low selectivity queries against a 100M-triple dataset or even highly selective queries against a 10G-triple dataset can yield 10M-triple resultsets. A more general approach is to evaluate the system accuracy in a piecemeal fashion using a benchmark comprising several workloads. Small real-world or synthetic datasets with simple and highly selective queries that use each one of the operators of interest, make it easy to check accuracy and find implementation or configuration issues with a system. With the previous issues resolved<sup>5</sup>, we proceed with large real-world datasets with querysets of interest where for each query we compare the number of returned results against the expected number of results. Under the preconditions mentioned, this is a good indicator of the query execution accuracy since it is fast to verify and with low storage requirements which makes it both easy to persist and disseminate.

## 2.4 Benchmarking Graph Stores

Various SPARQL and GeoSPARQL benchmarks have been devised over the past 20 years to test the supported features and performance of graph stores. Well known SPARQL benchmarks include: LUBM [27], BSBM [10], DBpedia SPARQL benchmark (DBPSB) [50] and the Social Network Benchmark (SNB) [20], just to mention a few. GeoSPARQL benchmarks have been presented in [63, 61], the benchmark Geographica in [25], a smart city services related benchmark in [8], a compliance benchmark in [42] and Geographica 2 in [37].

Benchmarking is a notoriously *difficult, time-consuming, resource intensive, high complexity, multi-parameter and error prone process* even when human nature's bias is not present to favor one of the proposed solutions. Since graph stores are continuously evolving and offer improved efficiency and new capabilities, *it is also a process that needs to be **repeated regularly***, if the benchmark results are to reflect a valid image of the graph store ecosystem.

## 2.5 Benchmarking Frameworks

A *benchmarking framework* is a software platform that assists with: (i) the integration of systems of interest, (ii) integration of existing benchmarks, (iii) generation and customization of benchmark datasets and querysets, (iv) running experiments of a benchmark against one or more systems, (v) collecting experiments results and system logs, (vi) result analysis and finally (vii) easy experiment verification. Some of these features appeared as new ideas or automations included in different benchmarks, which however *should not create the impression that these benchmarks can be considered proper frameworks*.

In particular, several SPARQL and GeoSPARQL benchmarks have generalized or automated the queryset and dataset generation task. For example, LUBM, which focuses on reasoning, features a university ontology-based synthetic data generator able to scale to arbitrary sizes. DBPSB's

---

<sup>5</sup> Removing problematic queries or reconfiguring the system.

queryset creation process is based on querylog mining, clustering and SPARQL feature analysis, which is applied to the DBpedia knowledge base and shows that performance of triple stores is by far less homogeneous than suggested by non application-specific benchmarks. FEASIBLE [74] suggests an automatic approach for the generation of application-specific benchmark querysets (SELECT, ASK, DESCRIBE and CONSTRUCT) out of the application's query logs history, thus enhancing insights as to the real performance of triple stores employed for a given application. IGUANA [17] innovates by providing an execution environment which can measure the performance of RDF stores during data loading, data updates as well as under different loads and parallel requests. LITMUS [77] proposes uniform benchmarking of non-spatial data management systems supporting different query languages, such as SPARQL and Gremlin. Sparqlscope [6] is a dataset statistics-driven, parametric templates-based SPARQL query generator which creates a total of 105 queries for any given RDF dataset, covers a large subset of SPARQL 1.1 features and allows a feature at a time study. Geographica and Geographica 2 include an ontology-based geospatial synthetic generator able to create spatial datasets of arbitrary size and also generate the corresponding queryset with a user defined thematic and spatial selectivity. Kobe [44] cloud benchmarking engine for federated query processors includes the GeoFedBench [78] benchmark, which focuses on validation of the actual crop land usage against the Austrian land survey dataset.

## 2.6 Benchmarking Framework Platforms

These are multi-user environments where researchers can store and share datasets, querysets, execution results and system modules. They are designed for deployment to cloud infrastructures, with distributed file systems and containerization technologies. HOBBIT [70], the most complete of these platforms, extends the scope of benchmarking to the entire LD life-cycle [55], such as *link discovery* [40] employs intuitive web UIs and allows the integration of systems in various programming languages. For a comparison between GeoRDFBench and HOBBIT please refer to Section 8.

## 3 Comparison Criteria for GeoSPARQL Benchmarking Frameworks

In order to streamline the comparison of GeoSPARQL and/or SPARQL benchmarking solutions, a number of criteria categories need to be considered. Each of them comprises several criteria, however, we discuss the absolute necessary. GeoRDFBench's design aims at covering as many of these criteria as possible or at the very least be easily extensible to include additional ones.

### 3.1 Benchmarking Coverage

This discussion concerns benchmarking four (4) stages of working with RDF stores:

1. **RDF Data Ingestion (Loading and Indexing).** This stage comprises measuring how long it takes for an RDF store to load an RDF dataset to a new repository (*load time*) and how long it needs to index the loaded data (*indexing time*). The *ingestion time* is the summation of load and indexing times. When multiple ingestion methods are available (online, offline/bulk loaders) the most appropriate method for the input RDF dataset size should be chosen unless comparing all loaders is an explicit aim of the benchmark. The stage completes by measuring the repository size after loading (*load size*) and after indexing (*ingestion size*). The repository *index size* is the difference of ingestion size minus the load size. We need to note that several systems have loaders that report only the repository ingestion time and size and might only offer an opportunity to take statistics about secondary indexes, such as spatial ones.
2. **Query Execution Performance and Accuracy.** This stage focuses on measuring the total time (*total execution time*) a query takes in order to execute (*evaluation time*) and return the entire resultset (*scan time*). Three important execution variants need to be considered

though: (i) single query with all caches cleared (*cold cache*), (ii) single query with caches enabled by previous executions (*warm cache*) and (iii) cache indifferent execution of constant or randomized mixtures of queries (*continuous/batch/queryset/query mixes*). For deterministic queries the gold resultset or at least the expected number of results is known, therefore it is also necessary to measure if the query returned the correct number of results (*query accuracy*). This test is necessary because small deviations in the number of results can reveal problems with function implementation or rounding errors in the internal representation of information on data files and indexes.

3. **RDF Store Data Scalability and User Concurrency.** Nowadays, behavior predictability against unknown future loads (data size, number of users) carries important weight on organization decision making process about the most appropriate RDF solution for their needs. As such, the study of the behavior of RDF stores against increasing larger datasets (*scalability*) and number of users (*user concurrency*), which actually takes part in the previous phase, is presented as a separate phase.

Common knowledge dictates, that excluding embedded system databases, serving concurrent users is rarely a need or a problem if the underlying repository size is small to medium but becomes an issue when data size drastically increases. This is also verified in the conclusions of benchmarks performed with the IGUANA framework involving several RDF stores against real-world and synthetic workloads: “..while the triple stores we evaluated seem to scale up well to concurrent queries and updates, *they are all affected significantly by the size of the datasets*”. Therefore, we propose that a logical sequence is testing system scalability first and later on perform user concurrency testing in the scalable systems subgroup.

4. **Statistics Reporting and Logging.** In this final stage the aim is to generate various useful statistic measures (median, average, standard deviation, no of queries per hour, etc.) about each query, queryset or system that might help the user draw conclusions and/or shed light on problematic areas that need to be addressed with further investigation. The recorded information of the first three stages augmented with the statistics produced in this phase can then be organized and disseminated in the form of reports in various locations (file system, databases) and different formats (CSV, TSV, JSON, XML, RDF, tables). Based on these reports, graphs and charts might be generated according to the users preferences. Also, in dynamic or template based querysets it is very important to log the exact query instance that was executed and for which metrics were collected. Finally, system configurations and experiment host environment parameters need to be recorded as a form of “chain of evidence” when comparing systems on the same host or the same system in different hosts.

## 3.2 Geospatial Support

RDF stores can have full or partial support of GeoSPARQL or even alternative spatial vocabularies with equivalent function libraries. The evaluation of RDF stores’ spatial support is tested by the existence of the following points:

1. **Spatial Index Creation Time and Size.** As mentioned earlier, when possible, the time it takes to create the spatial index (*spatial indexing time*) and the additional size imposed by it to the repository (*spatial index size*) should be measured. In most cases, these two values are substantial compared to the repository load time and size. They represent an additional investment of valuable resources and should carry a weight factor in the final evaluation of whether the query execution benefits, they might offer, are worth it at all or if in some application use cases they are.
2. **Spatial Index Specification.** Each RDF store specifies a list (usually one only) of *spatial indexing algorithms* (quad, geohash, b-tree, etc.). Some of these algorithms have accuracy ranges and a default value (*accuracy value*) which dictates how detailed a spatial search in



different regions will be. Low accuracy values have a triple side-effect: (i) they might allow false-positive results to appear for a specific spatial filter, (ii) the search time is reduced and (iii) the spatial indexing time and index size are also reduced. Thus it affects the measured query accuracy, the total execution time, the ingestion time and size. It is therefore important, when systems' configurations allow it, to set all competing RDF stores' spatial searching capabilities as close as possible. Some systems though, such as, RDF4J, Stardog, etc., they do not offer a way to set the accuracy value, while others such as GraphDB offer dynamic reconfiguration of the spatial index.

3. **GeoSPARQL Support and Compliance.** GeoSPARQL provides a common vocabulary and language for spatial RDF stores to be tested side-by-side either for performance or for feature coverage (*compliance level*) measurements. When possible, GeoSPARQL querysets should be used to drive the relevant benchmarks on spatial triple stores. However, there were and still are systems that have partially implemented GeoSPARQL and missing functionality is offered by system-oriented spatial vocabularies<sup>6</sup> and library extensions of SPARQL. For such systems, GeoSPARQL querysets might need some level of translation to system-specific variants (*query translation*). Past benchmarking efforts actually followed this path in order to be more inclusive. We believe that benchmarking frameworks should cater for such translations (application hooks) from GeoSPARQL constructs to system-oriented SPARQL constructs making certain that they are applied consistently generating variants of the input GeoSPARQL querysets on the fly while reporting the query variant executed in each case.
4. **GeoSPARQL Spatial Indexing Challenges.** All GeoSPARQL engines do not automatically recognize the presence of `geo:wktLiteral` property in order to spatial index the WKT serializations of the geometry objects referred to. They may require that each dataset explicitly states the properties that point to the geometry serializations e.g., the RDF4J Lucene SAIL<sup>7</sup> requires that to properly work. Some other systems, such as Parliament may require inference be enabled and additional RDFS triple statements inserted to point to the sub-properties of GeoSPARQL's `geo:hasGeometry` property. A benchmarking framework should enable the user to configure these properties in the most convenient place, preferably in a declarative manner.
5. **Spatial Index Enabled GeoSPARQL Query Execution.** Even during query execution some systems like Parliament and GraphDB will enable their spatial index only when GeoSPARQL spatial predicates are used. In these cases benchmarks such as Geographica and Geographica 2 do offer equivalent querysets to make sure that these systems are given fair opportunity at showing their strengths. Benchmarking frameworks should offer querysets with bundled spatial function and predicate versions of their queries and make sure that the execution engine feeds the appropriate set to the system that requires it. For systems such as GraphDB which supports GeoSPARQL operations with and without the use of the spatial index the engine should allow the system to choose which one of the two options to use.

### 3.3 Semantic Web Standards Support

Apart from the specific GeoSPARQL support mentioned, benchmarks and relevant frameworks should also check if several other Semantic Web standards are supported by the RDF stores under test. We briefly mention the most important ones:

<sup>6</sup> Virtuoso supports `virtrdf:Geometry` literal and built-in `bif:xx` functions

<sup>7</sup> [https://rdf4j.org/javadoc/latest/org/eclipse/rdf4j/sail/lucene/LuceneSail.html#WKT\\_FIELDS](https://rdf4j.org/javadoc/latest/org/eclipse/rdf4j/sail/lucene/LuceneSail.html#WKT_FIELDS)



1. **Quads as well as Triples.** Many benchmarks and frameworks do not even mention or test if triple stores support quads. They focus entirely on triples and they are missing the evaluation and study of the relevant 4-level thematic indexing structures, e.g., RDF4J's "SP0C, POSC, COSP"<sup>8</sup> or Virtuoso's "OGPS, PGOS"<sup>9</sup>.
2. **Multiple RDF Serializations Support.** It follows from the above that *TriG*, the extension of Turtle serialization, which supports representing a complete RDF Dataset, is not tested by all benchmarks or given as an input option in benchmarking frameworks. The most common format for datasets is *N-Triples* which uses absolute IRIs and is the least space conserving one.
3. **SPARQL Query Namespace Prefixes.** Some benchmarking frameworks, in an attempt to reduce technical effort, narrow their focus and do not provide any assistance in building the appropriate SPARQL query namespace prefix header for each executing system. Queries with their header are commonly managed and consumed as a complete block either one query per file or line. However, if query translation is required for some systems, this effort becomes more difficult to implement (usually manually) and verify its outcome. The solution they propose is a different set of query files or folders for each system, which the user has to consider when building the experiment tasks or workloads. The actual namespace prefix header for each query is a merge of the specific query's prefix map, combined with the implied prefix map in the target dataset or some graphs in it and finally the system-oriented vocabulary and extension prefix map (*namespace prefix map merging*). An automated namespace merging implies separate handling of the query from its header which by design simplifies the query translation process. One should expect from benchmarking frameworks claiming to assist users to provide some sort of automated support for this sort of tasks.

### 3.4 Framework Automation Level

The most important feature of a benchmarking framework over specific benchmarks is the amount of assistance it provides to its users.

1. **Setting Up New Benchmarks.** Designing a new benchmark usually means selecting and crafting datasets, querysets and describing the execution logic of the experiments. Usually a researcher locates the real data sources and through data curation and appropriate conversions formulates the desired RDF real-world datasets which may be structured as a collection of RDF graphs. Querysets can be a collection of static queries that are designed to test specific aspects of the systems against the chosen datasets. If the collection comprises many queries which follow certain patterns then *template based queries* are used where static or dynamic replacement of template tokens occurs to create the *ground queries* and at the same time keeps the number of queryset entries very low. More dynamic scenarios might include *synthetic query generators* which follow some ontology-based logic. In these cases they might be paired with a *scalable synthetic data generator* based on the same ontology which can generate datasets of various sizes. Yet another very popular case is to use log mining techniques from application logs and through filtering and transformation process generate the final queryset. This technique however targets real-world datasets and specific application use cases. Benchmarking frameworks therefore should at the very least offer: (i) a scalable synthetic data and query generator, (ii) facilitate dataset integration and bundling, (iii) support for a query template mechanism that allows easy creation of queryset variants while keeping the size small and (iv) support the three query execution variants mentioned in Section 3.1.2.

---

<sup>8</sup> (S)ubject, (P)redicate, (O)bject, (C)ontext

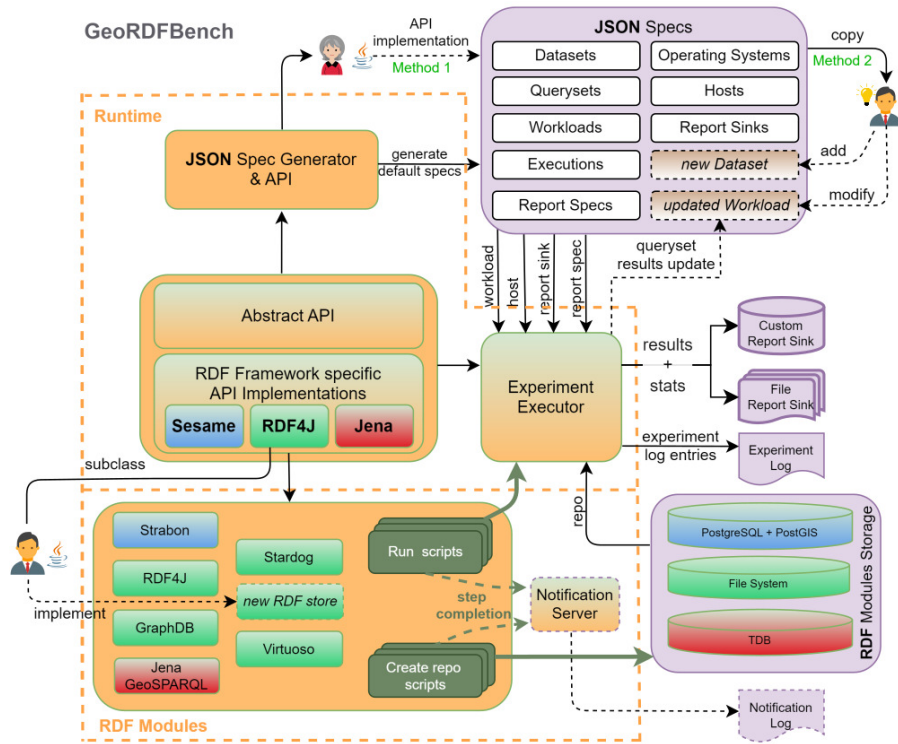
<sup>9</sup> (P)redicate, (G)raph, (O)bject, (S)ubject

2. **Updating Existing Benchmarks.** Updating an existing benchmark usually implies refreshing all participating systems to newer versions and probably adding new systems. The researcher faces an increased number of features e.g., new indexing algorithms, most of the time improvement of capabilities, a need for upgrading supporting client RDF library versions, changes in back-end architecture e.g., addition of an application or database server component, a new array of optimal system configuration options. A benchmarking framework should encode, as much of this information as possible for each supported system, in scripts, property files or using dynamic discovery in order to guide the user. It should also be host resource aware and allocate reasonable resources following the proposed performance guidelines of each system. This action assists the user in system configuration on the host environment which is a very laborious and sensitive task. Sub-optimal configuration of a system may result in biased results and wrong conclusions in the comparative evaluation report.
3. **Repeatable and Adaptable Experiment Execution.** With time, not only RDF stores evolve but also available host hardware and operating systems improve. Since benchmarking is a process that must be repeated to reflect the current status of the competition, it is necessary that benchmarking frameworks facilitate running the same benchmark in different environments. This can only be achieved by isolating the host and operating system aspects (*benchmarking environment*) that are relevant to the process, model them and create declarative specifications for each prospective experiment environment. This should ideally provide a plug and play behavior which the framework's executing engine should be able to adapt to. Aspects that are commonly used and affect the benchmarking process are host available RAM, disk and file locations for datasets and repositories, operating system cache control and console commands, etc. Finally a common scenario that benefits from experiment execution adaptability is when there is a need for running queryset subsets. The case might be that the experiments must focus on a specific area of interest, e.g. spatial joins with intersect operator, or that the target RDF stores under test have limited capabilities and cannot execute the full queryset. *Queryset filtering* for selective query execution at experiment run time promotes clarity and allows for maintaining well designed queryset libraries without having to deal with a haphazard collection of multiple variants of the same base querysets.
4. **Native and Dockerized Execution.** Until now, the previously mentioned problem had been addressed through the use of one or more containers (*dockerized execution*) which however use the same host operating system and only modify the allocated resources. Few frameworks allow *native execution* and testing of the systems and even fewer allow both types of operation. The problems of dockerization can be summed to the choice of using the **privileged mode** of operation or not. If not used, the framework is unable to perform cold cache query executions and if it does, it breaks the host-container isolation<sup>10</sup> creating security risks and instability in the host and other running containers. LITMUS is a framework that uses a single privileged container approach, IGUANA and GeoRDFBench use native mode of execution and the latter also allows single non privileged container operation for the initial benchmarking phases or training purposes.

### 3.5 Sustainability and Extensibility

We conclude with two obvious but neglected aspects of a benchmarking framework. The first question is whether this resource is actively supported, that is if there new features added or bugs fixed in the past 12-24 month period. Is there a plan for the medium and long-term maintenance of

<sup>10</sup><https://docs.docker.com/reference/cli/docker/container/run/#privileged>



■ **Figure 2** Architectural overview of GeoRDFBench Framework.

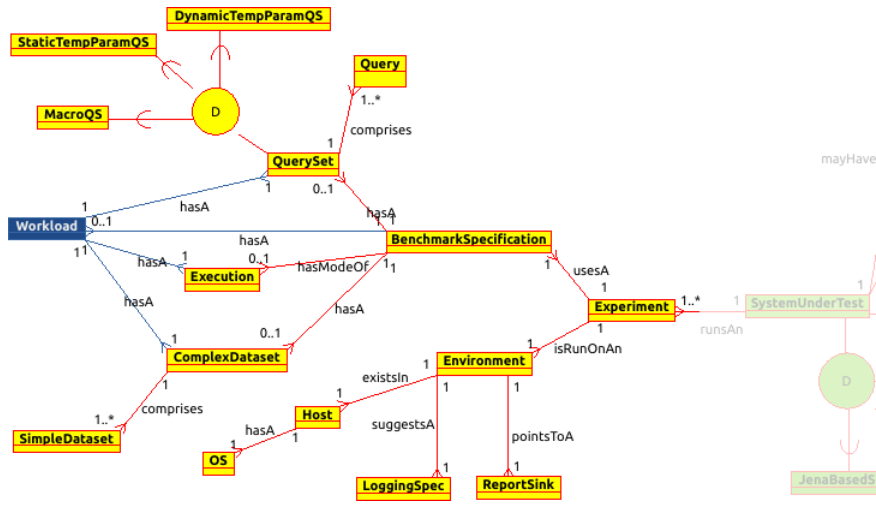
the resource (*sustainability*)? The next question is about the extensibility of the resource. Can it be easily extended or adapted by other users to support research and experimentation? Extending a framework which is *actively supported* is easier and can be more productive because questions can be readily answered by the developers. Another important factor is whether the technologies and methodologies in design and implementation are *open source*, *good quality* and *widely used*. The final and most crucial factor is of course if the resource was intentionally *designed to promote easy continuous evolution and extension*. Monolithic script-based approaches can rarely cover the last point as they require complete rewrite for changes that pertain to the execution logic, the introduction of template based query support, handling of system-depended vocabularies and query rewrites.

In the rest of this paper the above criteria are referenced by number in square brackets after the relevant text.

#### 4 GeoRDFBench: A Framework Simplifying Geospatial Semantic Benchmarking

In this section we present the high level architecture of the GeoRDFBench framework, which is shown in Figure 2.

The system consists of two main parts: the *runtime* and the *RDF modules*, depicted inside the dashed border. The runtime is the engine and fabric of the framework and it is responsible for generating JSON benchmark specifications and executing experiments initiated by the RDF modules' run scripts. The RDF modules is where pre-implemented and newly implemented RDF stores reside that can participate in experiments. Stores use their *repository creation script*



■ **Figure 3** EER diagram of Abstract API - Experiment Components.

to create repositories and import data to them. Each store’s *experiment run script* initiates a benchmark experiment and eventually invokes the runtime’s *experiment executor* component passing all required inputs which include, among others: the RDF store, the repository, the benchmark workload, the host where the experiment is conducted on and the report sink where experiment results and statistics will be stored. Both types of scripts send progress messages to the optionally enabled, remote or local, *notification server* which logs them and serves as a useful non-intrusive monitoring tool for the researcher.

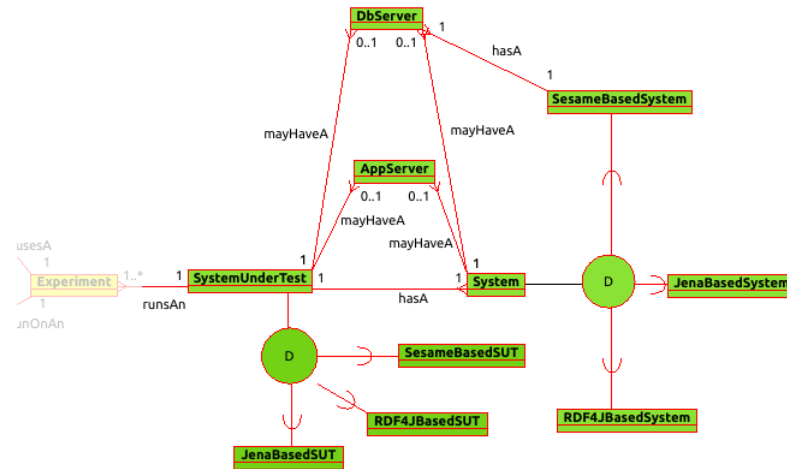
The default *JSON specifications* correspond to the Geographica 2 and LUBM benchmarks’ components and are generated by the *JSON Specs Generator*. This “starter dough” library is stored on the file system separately from GeoRDFBench code. The user can either implement new custom specifications using the JSON Specs Generator API (**Method 1**) or copy and modify existing ones (**Method 2**).

The *JSON Library REST API endpoint* module (not shown in Figure 2) can provide an alternate way for the experiment executor to acquire JSON specifications from a remote location such as a docker container or server and promotes both a more distributed approach to the framework’s operation and increased reusability of these resources.

## 5 GeoRDFBench Runtime: The Framework’s Engine

The GeoRDFBench runtime consists of four components: (i) the **Abstract API**, (ii) the **RDF Framework Specific API Implementations**, (iii) the **Experiment Executor** and (iv) the **JSON Specification Generator**.

In this section, the term *system* is used to refer to the repository functionality of a geospatial RDF store, while the term *system under test (SUT)* is used to refer to the system-host pair, along with management capabilities of the system’s application and database server status, if they are present. The SUT knows how and in which sequence to start and stop the application server, repository and database server of the system and to clear system caches. SUT is also the vehicle with which experiment timed queries are executed.



■ **Figure 4** EER diagram of Abstract API - System Under Test.

## 5.1 Abstract API

This is a core part of the GeoRDFBench’s benchmarking API. It abstracts the properties, functionalities and interactions of the *benchmark experiment components* and the *SUT*. On the conceptual level, the two simplified<sup>11</sup> Enhanced-ER (EER) diagrams, in Figure 3 and in a part of Figure 4, show the important entities, their specializations, how they are associated, along with the corresponding structural constraints (cardinality constraints) for these associations.

On the implementation level, the *Abstract API* creates class hierarchies for each component type, exposes common functionality with appropriate interfaces and uses abstract classes to pull up properties and provide default implementations for operations.

### 5.1.1 Experiment Components

Figure 3 depicts the experiment components, which are detailed below and which are logically organized into the *Benchmark Specification* and *Experiment Environment* groups.

#### 5.1.1.1 Benchmark Specification

This group includes all components necessary to describe a benchmark which are independent of the platform where the experiment runs. There are two forms available: (i) the *detailed form* uses independent component specifications and (ii) the *compact form* which uses the *workload* “container” concept to group several components. With the exception of the *workload*, all components described below are part of the *detailed form*.

#### Dataset

The entity **ComplexDataset**, in Figure 3, represents the “complex” or “composite” geospatial dataset which can comprise one or more “simple” geospatial datasets represented by the **SimpleDataset** entity [3.4.1]. The *simple dataset* specification contains: a logical name, the dataset relative path location, the filename, the RDF serialization format, a map of dataset relevant namespace

<sup>11</sup> Only important entities, specializations and relationships are depicted while attributes are omitted.

prefixes, a map of properties that link features with their geometries which represent their spatial extent e.g., `geo:hasGeometry`, a map of properties that link a geometric element with its WKT serialization e.g., `geo:asWKT` [3.2.3, 3.2.4], and the scaling factor used in case of a synthetic dataset. The *complex dataset* specification contains: a logical name, the dataset base path location, a map of contexts (named graphs) [3.3.1, 3.4.1] and the list of simple datasets comprising it. Experiments use only complex datasets.

## Queryset

GeoRDFBench follows the path of Geographica 2 and adopts the **micro** and **macro** experiment queryset categorization proposed in the Jackpine [66] geospatial database benchmark. Micro querysets simulate a lab-like controlled environment where the aim is to assess the performance and compliance of individual functions of interest e.g., a spatial function, and each query has a unique and autonomous purpose. The performance metrics of interest are query accuracy and the query response time over repeated executions with cold and warm caches [3.1.2]. Macro querysets, on the other hand, attempt to simulate real-world scenarios where the aim is to assess the expected performance of certain use cases e.g., “Rapid Mapping for Wild Fire Monitoring” in Geographica 2. They can comprise a random selection of *ground queries*<sup>12</sup> with arbitrary criteria or even form a dependency graph of ground and *template queries*<sup>13</sup> where the resultsets of specific ground queries’ execution operate as arguments for template parameters in order to instantiate the template queries before they execute [3.4.1]. The performance metrics of interest in these cases is mainly query throughput, irrespective of cache status, which translates to number of query executions in a period of time e.g., 1 hour [3.1.2].

The entity **QuerySet**, in Figure 3, represents the geospatial queryset which can comprise one or more queries represented by the **Query** entity. The *query* specification contains: a label, the GeoSPARQL query text which may contain replaceable tokens (template parameters), a flag that signals the existence of spatial predicate [3.2.3, 3.2.5] and the accuracy validation indicator. The accuracy indicator denotes whether the expected number of results returned by the query is dataset-dependent, template-dependent, or independent [3.1.2]. The *queryset* specification contains: a logical name, the queryset path location, a map of queryset relevant namespace prefixes [3.3.3], a map of fixed, static template, or dynamic template queries with arithmetic index and replacement maps that assist in creating ground queries from template queries. Experiments use only querysets, which *can be inclusively or exclusively filtered at run time* [3.4.3]. All **QuerySet** subclasses can handle ground queries but they differ in the flexibility they offer in template query parameter instantiation. The **StaticTempParamQS** subclass models sets of template parameter queries which have fixed parameter values for all queries. This is useful when very large literals e.g., a 2KB WKT polygon literal, need to be repeated in several queries, thus it allows for a succinct representation of the template queryset for viewing, storing and dissemination purposes while allowing for the exploded representation only during experiment execution [3.4.1]. The **DynamicTempParamQS** subclass models sets of template parameter queries which may have different value for each parameter for each query. This can be viewed as providing a declarative alternative for creating a simplified synthetic query generator [3.4.1]. While the previous subclasses are useful for micro experiment types, the **MacroQS** subclass and its specializations<sup>14</sup> are useful for macro experiment types.

<sup>12</sup> Queries with RDF terms only (IRIs, blank nodes and literals).

<sup>13</sup> Queries with replaceable tokens acting as placeholders for runtime substitution.

<sup>14</sup> Not shown in Figure 3 for simplicity reasons.

### Execution specification

The entity **Execution**, in Figure 3, describes which experiment action (*run*, *print*) to take, the query execution types (*cold*, *warm*, *cold\_continuous*) and number of repetitions per type, the query repetition timeout and total timeout for all repetitions, the delay period for synchronous clearing of system caches, the function to use for aggregating execution times in the standard file-system based statistics and the policy to follow when a cold execution times out [3.1.2, 3.4.3, 3.5]. The non-default *print* action triggers a pseudo-execution which generates the ground queryset for inspection purposes [3.1.4].

### Workload (compact form)

The entity **Workload**, in Figure 3, represents the compact form of the benchmark experiment description: *dataset + queryset + execution specification*. It promotes ease of use and consistent experiment execution [3.4.1-3].

#### 5.1.1.2 Experiment Environment

This group includes all experiment platform dependent components.

### Operating system

The entity **OS**, in Figure 3, represents the host's operating system and features a name, the shell command path, the commands for synchronizing cached data to persistent storage and the one for fully clearing caches (pagecache, dentries and inode) [3.4.2, 3.4.3].

### Host

The entity **Host**, in Figure 3, represents the hardware platform where the benchmark experiment is taking place [3.4.2, 3.4.3]. It has the host name, the operating system, IP address, total RAM (GBs), the base path for the actual dataset files, the base path for RDF store repository files and the base path for the standard file-system based reports and statistics.

### Report sink

In addition to the standard file-system based reports and statistics that GeoRDFBench generates by default, there is the option of collecting these statistics in an advanced report store such as a database [3.1.4]. The entity **ReportSink**, in Figure 3, describes the experiment result report store, where detailed statistics will be sent and custom reports generated. The default report store is a PostgreSQL JDBC implementation and has as properties, the driver name, hostname, alternate hostname, port, database name, user and password. Alternate hostname allows for having a fall-back database where results from extremely long running experiments can be saved.

The PostgreSQL report store has as default behavior the *deferred insertions* for query execution results, that involves an experiment result collector which flushes results upon experiment termination. In this way, we avoid synchronous result insertions to the report sink, which would be disrupted by the repetitive restarts of the database component for SUTs using the same DBMS as the report sink. A second lighter option is the H2 embedded database which offers synchronous experiment result recording. In both cases, the target report sink's database schema is automatically generated by the runtime using the *database definition script* bundled with each report sink's source code. This schema can also be generated with the help of the runtime-bundled *database generation SQL script*.



Customized reports are provided by several report views with aggregate statistics which exist in the database schema. These include the mean and median value of the total query execution time, the number of executions per query, cache and system tested. Additional statistics such as variance and standard deviation can be easily added to these report views either by altering the report view definition with the database client or more permanently by updating the `schemaInitScriptreport` static property of the relevant abstract class `JDBCRepSrc` or `EmbeddedJDBCRepSrc` in the GeoRDFBench's runtime.

### Logging or Report specification

The entity `LoggingSpec`, in Figure 3, allows customization of the number of resultset entries to be logged during the query execution scanning phase of the `Experiment Executor`. A positive non zero integer value allows for a sample of the results returned by each query to be recorded in the experiment log and can be used as a proof of concept that a system performs accurately or similar to other systems. Such a setting is useful in early benchmarking phases and can help identify, early on, issues with disabled plugins, external libraries, or with incorrect results by non-compliant function behavior [3.1.4]. A zero value, on the other hand, allows for very accurate calculation of the query response time and is useful in the final benchmarking phase.

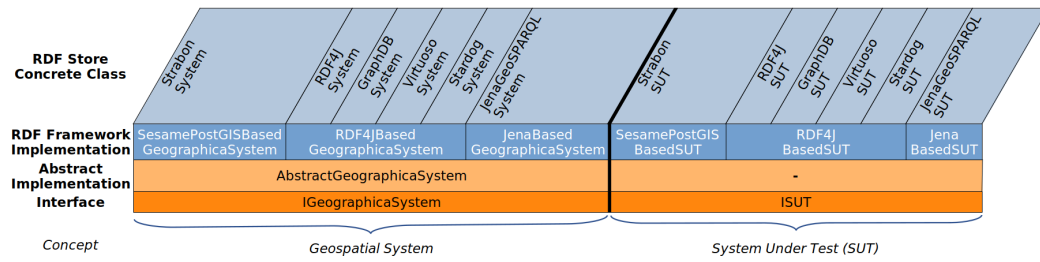
## 5.1.2 Systems and SUTs

In Figure 4, the parent concepts of system and system under test (SUT) components only, are also part of the Abstract API. The entity *System* represents an RDF framework independent geospatial semantic store and more specifically the repository aspect of it [3.4.3, 3.5]. It is described by a map of properties and their values, such as repository location and name, system relevant namespace prefixes, as well as various indexing parameters. It also has a connection property which allows query execution and a flag to denote whether the store has been initialized. The entity *SystemUnderTest* on the other hand represents a *System* with its optional application and database server components, represented by *AppServer* and *DbServer* respectively [3.4.3, 3.5].

On the implementation level, the Abstract API comprises two layers: (i) the *Geospatial System Abstraction Layer*, which is depicted in the lower left two hierarchy levels of Figure 5, and (ii) the *System Under Test (SUT) Abstraction Layer*, which is the lower right level of the same figure, both of which are explained below.

### Geospatial System Abstraction Layer

This layer comprises one interface that describes a geospatial RDF store and one abstract class that implements the RDF framework independent common functionality. The *Geospatial Graph System Interface* (`IGeographicaSystem`), is a contract that requires functions for: setting a map of system properties, system initialization, system termination and a function that returns system-specific namespace prefix mappings. The *Base Abstract Implementation* (`AbstractGeographicaSystem`) of `IGeographicaSystem`, is an abstract class that uses generics and encapsulates the system properties map, the initialization status, the generic repository “connection”, which is RDF Framework specific and the skeleton functionality to handle these. This generic “connection” corresponds to an appropriate RDF Framework abstraction that allows creating query instances on a system repository.



■ **Figure 5** Systems & SUT Class Hierarchies.

## System Under Test (SUT) Abstraction Layer

This layer comprises the generic *Geospatial Graph SUT Interface* (ISUT), which is a contract that requires functions for: retrieving the host, the generic “system”, execution and report specifications, starting and terminating the application and database server, making system dependent translations of the queryset and executing timed queries [3.3.3, 3.4.2, 3.5].

## 5.2 RDF Framework Specific APIs

The second part of the core API, on the conceptual level, is depicted in part of Figure 4 which includes the specializations of system and SUT. In a similar manner, system and SUT concepts have three child entities to model the corresponding three RDF framework specific concepts: *RDF4JBasedSystem*, *JenaBasedSystem*, *SesameBasedSystem*, *RDF4JBasedSUT*, *JenaBasedSUT* and *SesameBasedSUT* [3.4.2, 3.5].

On the implementation level, this part comprises: (i) the *RDF Framework Specific System Layer*, which is depicted by the left side of “RDF Framework Implementation” level of Figure 5, and (ii) the *RDF Framework Specific SUT Layer*, which is the right side of the same level, both of which are explained below.

### 5.2.1 RDF Framework Specific System Layer

This layer consists of three specializations of the `AbstractGeographicaSystem` class, one for each RDF framework supported by the GeoRDFBench runtime. Each class grounds the generic “connection” to the most appropriate interface or class of the RDF framework it implements. Since each framework has more than one major release, which commonly break backward compatibility, the exact version of each RDF framework supported by the runtime was based on its usage by RDF graph stores. The three specialization classes are:

#### Sesame API (`SesamePostGISBasedGeographicaSystem`)

Most scalable RDF solutions based on Sesame, use v2.6.x since support of the RDBMS Sail was deprecated after that. This Sail allowed graph stores to tap, among other things, into geospatial and other capabilities provided by well known DBMSs’ such as PostgreSQL with PostGIS. Therefore, this implementation adds: host, port, database name, user and password, to the system properties map and handles them appropriately. The generic “connection” type is replaced with class:

```
org.openrdf.repository.sail.SailRepositoryConnection
```

**RDF4J API (RDF4JBasedGeographicaSystem)**

Version 4.3.15 was selected since is widely supported by all contemporary systems. This implementation focuses on the NativeStore Sail and adds the repository base directory, repository name and indexes used. The generic “connection” type is replaced with interface:

```
org.eclipse.rdf4j.repository.RepositoryConnection
```

**Jena API (JenaBasedGeographicaSystem)**

Since only Jena GeoSPARQL uses this API, we simply chose a recent and frequently used Jena version in Maven Central [52], from the latest stable branch, which was 4.10.0. Jena Tuple Database (TDB2) [62] was preferred as the persistent storage option as it is more robust and can handle large update transactions. This implementation adds the repository base directory, repository name and injects the transaction functionality in the system initialization and termination code. The generic “connection” type is replaced with interface:

```
org.apache.jena.rdf.model.Model
```

**5.2.2 RDF Framework Specific SUT Layer**

This layer comprises three base implementations of the ISUT interface with corresponding abstract classes: `SesamePostGISBasedSUT`, `RDF4JBasedSUT` and `JenaBasedSUT`. Each class among other things handles the initialization and termination of system, application and database server components of the SUT either as a whole or on a component basis. They also invoke system-specific query translations, manage and monitor query execution which takes place in a separate thread, such as enabling timeout for the executing query and handling customized exceptions thrown by different RDF frameworks during the query evaluation phases [3.3.3, 3.4.2, 3.5].

**5.3 Experiment Executor**

The executor comprises the concrete `Experiment` and abstract `RunSUTExperiment` classes. The subclasses of `RunSUTExperiment` that RDF modules have to implement are the entry points for all experiment run scripts. The `RunSUTExperiment`, parses the script arguments that describe which JSON specifications (see Figure 2) need to be deserialized into experiment component instances, applies queryset filter if needed, configures the SUT with the above and finally launches the `Experiment` run loop, whose logic is described in Algorithm 1 of Appendix C [3.4.2, 3.4.3, 3.5].

Two actions, performed at the experiment construction time, are the namespace prefix map merging between the corresponding maps of the system, dataset and queryset along with system dependent queryset rewrites, in case non standard vocabularies are used, offering similar functionality [3.3.3, 3.4.2, 3.4.3].

**5.4 JSON Specification Generator**

This runtime component is a collection of runnable utility classes with no parameters, one for each experiment component type, which create all the specifications necessary to run the Geographica 2 benchmark. This geospatial benchmark employs the majority of dataset, queryset and execution specifications’ types. These JSON specifications are part of the project build tree and are readily available to the user [3.2.3, 3.3, 3.4.1, 3.4.2, 3.4.3].

The detailed component type hierarchies, create the need to handle many polymorphic instances which must be properly serialized, otherwise it will be impossible to deserialize them. For this purpose, the Jackson [22] JSON library is used to annotate interfaces and class hierarchies to simplify serialization and deserialization [3.5].

## 6 JSON Generator API - By Example

In this section, we demonstrate the use of the runtime JSON generator API for generating the *detailed form* for benchmark specifications, using as test case, the *Scalability* workload of the Geographica 2 GeoSPARQL benchmark [3.1.2, 3.1.3].

### Scalability Workload Description

This workload (see Table 1) is intended to evaluate geospatial RDF store scalability against increasingly larger real-world datasets with many complex geometries. There are 6 workload variants, each comprising one dataset with increasing number of triples (10K, 100K, 1M, 10M, 100M, 500M), a set with either 3 spatial function queries (1 selection and 2 joins) or with 3 equivalent spatial predicate queries<sup>15</sup> and a common experiment execution model. The execution model defines that each query shall be executed 3 times with COLD caches, 3 times with WARM caches and that the median of the 3 execution times shall be considered as the result for COLD and WARM executions. The maximum timeout period for each query is set to 24 hours and in case a query's COLD execution fails then all remaining COLD and WARM executions should be skipped. A 5000 msec delay is also specified after clearing caches and waiting for garbage collection. The below code samples demonstrate how to generate three different types of specifications, serialize them to JSON files and then deserialize them from the same files. Listing 13 encodes all the above.

■ **Table 1** Geographica 2 Scalability workloads.

| Workload         | Dataset          | Queryset                                 | Execution Spec |
|------------------|------------------|------------------------------------------|----------------|
| Scalability 10K  | scalability_10K  | scalabilityFunc<br>or<br>scalabilityPred | scalability    |
| Scalability 100K | scalability_100K |                                          |                |
| Scalability 1M   | scalability_1M   |                                          |                |
| Scalability 10M  | scalability_10M  |                                          |                |
| Scalability 100M | scalability_100M |                                          |                |
| Scalability 500M | scalability_500M |                                          |                |

### 6.1 Dataset generation

The following code creates the `scalability_10K` complex dataset specification object.

■ **Listing 1** Generate Dataset Specification.

```
public static GeographicaDS newScalabilityDS() {
    // create a simple dataset object with a single N-Triples file
    GenericGeospatialSimpleDS sds
        = new GenericGeospatialSimpleDS("scalability_10K", // simple dataset name
            "Scalability/10K", // relative directory where the file resides
            "scalability_10K.nt", NTRIPLES_STR); // the triples file
    // add to it any namespace prefixes used in the dataset file
    sds.addUsefulNamespacePrefix("lgo", "<http://data.linkedeodata.eu/ontology#>");
    // add to it any property used in the dataset that denotes feature geometry
    sds.addHasGeometry("scalabilityHasGeometry", "<http://www.opengis.net/ont/geosparql#hasGeometry>");
    // add to it any property used in the dataset that denotes WKT serialization
    sds.addAsWKT("scalabilityAsWKT", "<http://www.opengis.net/ont/geosparql#asWKT>");
    // create a complex dataset object with a single simple dataset object
    GeographicaDS gds =
```

<sup>15</sup> Systems, such as GraphDB and Parliament, use their spatial index only with spatial predicates.

```

        GeographicaDS(sds, // the simple dataset
            "", // context/graph IRI for the simple dataset
            0); // synthetic dataset scaling factor, 0 for non synthetic datasets
// serialize the complex dataset specification object to a JSON file
gds.serializeToJSON(new File(SCALABILITY_JSONDEF_FILE));
// deserialize a complex dataset object from a JSON file and return it
return DataSetUtil.deserializeFromJSON(SCALABILITY_JSONDEF_FILE);
}

```

The complex dataset comprises an N-Triples simple dataset file and Listing 11 details the generated JSON file.

## 6.2 Queryset generation

We also generate the first variant of the quervset, `scalabilityFunc`, which uses spatial functions.

■ **Listing 2** Generate Queryset Specification.

```
public static StaticTempParamQs scalabilityFuncQS() {  
    // read fixed Polygon from external file which is used in spatial selection query  
    String givenPolygon = readFile(SCALABILITY_EUROPE_POLYGON_FILE);  
    // initialize the map of useful general RDF related prefixes  
    Map<String, String> mapUsefulNamespacePrefixes = new HashMap<>();  
    // initialize the map of template parameters  
    Map<String, String> mapTemplateParams = new HashMap<>();  
    mapTemplateParams.put("FUNCTION", "sfIntersects");  
    mapTemplateParams.put("GIVEN_SPATIAL_LITERAL", givenPolygon);  
    // populate Graph prefixes map  
    Map<String, String> mapLiteralValues = new HashMap<>();  
    // populate template queries map  
    Map<Integer, IQuery> mapQry = new HashMap<>();  
    mapQry.put(0, new SimpleQuery("SC1_Geometries_Intersects_GivenPolygon",  
        "SELECT ?s1 ?o1 WHERE { \n ?s1 geo:asWKT ?o1 . \n lgo:(geof:FUNCTION(?o1, GIVEN_SPATIAL_LITERAL)). \n} \n",  
        false));  
    mapQry.put(1, new SimpleQuery("SC2_Intensive_Geometries_Intersect_Geometries",  
        "SELECT ?s1 ?s2 \nWHERE { \n ?s1 geo:hasGeometry [ geo:asWKT ?o1 ] ;\n lgo:has_code \"1001\"^^xsd:integer . \n\n ?s2 geo:hasGeometry [ geo:asWKT ?o2 ] ;\n lgo:has_code ?code2 . \n FILTER(?code2>5000 && ?code2<6000 &&  
?code2 != 5260) .\n FILTER(geof:FUNCTION(?o1, ?o2)). \n} \n",  
        false));  
    mapQry.put(2, new SimpleQuery("SC3_Relaxed_Geometries_Intersect_Geometries",  
        "SELECT ?s1 ?s2 \nWHERE { \n ?s1 geo:hasGeometry [ geo:asWKT ?o1 ] ;\n lgo:has_code \"1001\"^^xsd:integer . \n\n ?s2 geo:hasGeometry [ geo:asWKT ?o2 ] ;\n lgo:has_code ?code2 . \n FILTER(?code2 IN (5622, 5601, 5641,  
5621, 5661)) .\n FILTER(geof:FUNCTION(?o1, ?o2)). \n} \n",  
        false));  
    StaticTempParamQs scalabilityQS =  
        new StaticTempParamQs("scalabilityFunc", "", false, mapQry,  
            mapTemplateParams, mapUsefulNamespacePrefixes, mapLiteralValues);  
    // serialize the queryset specification object to a JSON file  
    scalabilityQS.serializeToJson(new File(SCALABILITY_FUNC_JSONDEF_FILE));  
    // deserialize a queryset objet from a JSON file and return it  
    return QuerySetUtil.deserializeFromJSON(SCALABILITY_FUNC_JSONDEF_FILE);  
}
```

The generated JSON file detailed in Listing 12 of Appendix A uses the `StaticTempParamQS` subclass which allows the user to model querysets with or without template parameters but with fixed parameter values for all queries.

### 6.3 Execution spec generation

The following code generates the Scalability execution specification object.

**Listing 3** Generate Execution Specification.

```
public static SimpleES newScalabilityES() {  
    // create a map with no of executions per execution type  
    Map<ExecutionType, Integer> execTypeReps = new HashMap<>();  
    execTypeReps.put(ExecutionType.COLD, 3);  
    execTypeReps.put(ExecutionType.WARM, 3);  
    // create a simple execution specification object
```

```

SimpleES sses = new SimpleES(execTypeReps,
    24 * 60 * 60, // 24 hours max duration per query execution
    7 * 24 * 60 * 60, // 7 days max duration for the experiment
    Action.RUN, // run experiments instead of printing ground queryset
    AverageFunction.QUERY_MEDIAN, // use median instead of mean
    BehaviourOnColdExecutionFailure.SKIP_REMAINING_ALL_QUERY_EXECUTIONS,
    5000); // 5000 msecs delay for clearing caches and garbage collection
// serialize the execution specification object to a JSON file
sses.serializeToJSON(new File(SCALABILITYEXECUTIONSPECJSONDEF_FILE));
// deserialize an execution spec object from a JSON file
// and return it
return ExecutionSpecUtil.deserializeFromJSON(SCALABILITYEXECUTIONSPECJSONDEF_FILE);
}

```

The generated serialized representation is detailed in Listing 13 and fully describes the experiment execution model in *Scalability Workload Description* presented earlier in this section.

In a similar manner, the user can generate the experiment environment specifications: host, operating system, report sink and logging. At this point we should remind that, as it was envisioned by design, the trivial method of copying and modifying an existing specification file with a standard text editor, will probably suffice for many use cases, once a library of JSON specifications is already available. Both methods of generating specifications are depicted in the upper part of Figure 2. For an example of generating the *compact form* of a benchmark specification, the user can refer to *GeoRDFBench Framework Samples* [33] application that is using as test case, the LUBM benchmark for SPARQL.

## 7 Experimental Evaluation

In this section, we present the process of running some of the Geographica 2 [37] benchmark scalability experiments with the help of the GeoRDFBench framework and present the results. The experiment environment, benchmark description and execution details are presented below.

### 7.1 Environment

#### Host Hardware

The hardware platform for the experiments was an Intel NUC8i7BEH box, Ubuntu 22.04.5 LTS with 32GB DDR4-2400MHz, a Samsung SSD NVMe 970 EVO Plus 500GB system disk and a secondary data disk Western Digital WDC WD20SPZX-75U 2TB mounted on `/data`. Both filesystems `/` and `/data` were formatted as “ext4”. All SUTs and their repository data were intentionally placed under the slower `/data` filesystem.

#### Installed Software

Virtuoso and GraphDB required that we install the corresponding server software under `/data`. In addition PostgreSQL v14.17 with PostGIS v3.2 was also installed, since Strabon requires it for creating spatial databases. The PostgreSQL “data\_directory” was set in `/data/pgdata/data`. The branch `releases/2.0.0-M1` of GeoRDFBench was used in `/data/geordfbench`. The project came ready with prebuilt binaries. The report sink schema was left to be automatically created by the first system to run experiments. Manually creating the same schema is provided through the `scripts/geordfbench.sql` script.

#### System-Version Selection

Some RDF modules included in GeoRDFBench allow, without effort, testing system configuration variations. In view of this, we chose the following **seven** (7) systems and system variations to participate in this demonstration [3.2.4]: (i) GraphDB v10.8.5, (ii) GraphDB v10.8.5 with

*GeoSPARQL* plugin enabled, *quad* prefix tree indexing algorithm and 11 precision value (iii) RDF4J v4.3.15, (iv) RDF4J v4.3.15 with *Lucene Sail* enabled for spatial indexing (v) Openlink Virtuoso Open Server (VOS) v7.2.14 (vi) Jena GeoSPARQL v4.10.0 with *STRtree*<sup>16</sup> spatial index and (vii) Strabon v3.3.3-SNAPSHOT. Additionally, we tested the second GraphDB variant against two versions of the querysets [3.2.5]: (i) with *spatial functions* which is the default for all systems and (ii) with *spatial predicates*, to exhibit the different optimization paths followed and the importance of testing all these alternatives offered for properly benchmarking spatially enabled RDF stores. We were not provided with a license renewal for Stardog<sup>17</sup> therefore we were unable to run experiments with the latest v8.2.2 of the module which is included with GeoRDFBench. However, we do include it in the description of other aspects of the benchmarking process.

Since each GeoRDFBench RDF module is a Java client to an RDF store (embedded or standalone server) the current implementations use a Java RDF Framework library to instrument the stores. We currently have GraphDB, RDF4J and Virtuoso using *Eclipse RDF4J*, Jena GeoSPARQL uses *Apache Jena* while Strabon uses *OpenRDF Sesame*.

■ **Table 2** Scalability datasets basic characteristics.

| Dataset | Size   | # of Features | # of Points | # of Lines | # of Polygons |
|---------|--------|---------------|-------------|------------|---------------|
| 10K     | 3.7 MB | 1,135         | 587         | 0          | 900           |
| 100K    | 20 MB  | 12,166        | 6,623       | 4,239      | 2,531         |
| 1M      | 194 MB | 118,161       | 46,781      | 45,238     | 29,200        |
| 10M     | 1.9 GB | 1,038,739     | 317,865     | 328,630    | 427,842       |
| 100M    | 19 GB  | 10,259,959    | 904,677     | 2,058,386  | 7,553,440     |
| 500M    | 97 GB  | 48,623,878    | 5,520,767   | 15,771,932 | 23,390,220    |

## 7.2 Benchmark description

The following experiment aims both at studying performance characteristics of the SUTs but more importantly exhibiting the usage of GeoRDFBench for running benchmarks. We chose the 10K, 100K, 1M and 10M variants of the Scalability workload (see Table 1) where the respective datasets contain 10K, 100K, 1M and 10M triples [3.1.3]. All share the same queryset which comprises three queries: (i) a spatial selection *SC1*, (ii) a low selectivity (heavy) spatial join *SC2* and (iii) a higher selectivity (lighter) spatial join *SC3* [3.2.3]. The queryset is an instance of the *StaticTempParamQS* class with one template parameter, which represents a file persisted polygon replaced during queryset instantiation, used in the spatial filter of query *SC1*. The basic characteristics of the datasets are described in Table 2.

The following listing shows the ground spatial function queryset for RDF4J:

■ **Listing 4** RDF4J Ground Spatial Function Queries & Prefix Header.

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geo-sf: <http://www.opengis.net/ont/sf#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX lgo: <http://data.linkedeodata.eu/ontology#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
```

<sup>16</sup> From the JTS library, a query-only R-tree using the Sort-Tile-Recursive (STR) algorithm

<sup>17</sup> Stardog requires a license otherwise experiments fail.



```

# Query 0 - SC1_Geometries_Intersects_GivenPolygon:
SELECT ?s1 ?o1 WHERE {
  ?s1 geo:asWKT ?o1 .
  FILTER(geof:sfIntersects(?o1, "POLYGON((23.708496093749996 37.95719224376526,
    ... 37.95719224376526))"^^<http://www.opengis.net/ont/geosparql#wktLiteral>)).
}
# Query 1 - SC2_Intensive_Geometries_Intersect_Geometries:
SELECT ?s1 ?s2 WHERE {
  ?s1 geo:hasGeometry [ geo:asWKT ?o1 ] ;
  lgo:has_code "1001"^^xsd:integer .
  ?s2 geo:hasGeometry [ geo:asWKT ?o2 ] ;
  lgo:has_code ?code2 .
  FILTER(?code2>5000 && ?code2<6000 && ?code2 != 5260) .
  FILTER(geof:sfIntersects(?o1, ?o2)).
}
# Query 2 - SC3_Relaxed_Geometries_Intersect_Geometries:
SELECT ?s1 ?s2 WHERE {
  ?s1 geo:hasGeometry [ geo:asWKT ?o1 ] ;
  lgo:has_code "1001"^^xsd:integer .
  ?s2 geo:hasGeometry [ geo:asWKT ?o2 ] ;
  lgo:has_code ?code2 .
  FILTER(?code2 IN (5622, 5601, 5641, 5621, 5661)) .
  FILTER(geof:sfIntersects(?o1, ?o2)).
}

```

For comparison, the next listing shows the spatial predicate queryset for GraphDB+P [3.2.5]:

■ **Listing 5** GraphDB Ground Spatial Predicate Queries & Prefix Header.

```

PREFIX ext: <http://rdf.useekm.com/ext#>
...
# Query 0 - SC1_Geometries_Intersects_GivenPolygon:
SELECT ?s1 ?o1 WHERE {
  ?s1 geo:asWKT ?o1 .
  ?s1 geo:sfIntersects "POLYGON((23.708496093749996 37.95719224376526,
    ... 37.95719224376526))"^^<http://www.opengis.net/ont/geosparql#wktLiteral> .
}
# Query 1 - SC2_Intensive_Geometries_Intersect_Geometries:
SELECT ?s1 ?s2 WHERE {
  ?s1 geo:hasGeometry ?g1 ;
  lgo:has_code "1001"^^xsd:integer .
  ?s2 geo:hasGeometry ?g2 ;
  lgo:has_code ?code2 .
  ?g1 geo:sfIntersects ?g2 .
  FILTER(?code2>5000 && ?code2<6000 && ?code2 != 5260) .
}
# Query 2 - SC3_Relaxed_Geometries_Intersect_Geometries:
SELECT ?s1 ?s2 WHERE {
  ?s1 geo:hasGeometry ?g1 ;
  lgo:has_code "1001"^^xsd:integer .
  ?s2 geo:hasGeometry ?g2 ;
  lgo:has_code ?code2 .
  ?g1 geo:sfIntersects ?g2 .
  FILTER(?code2 IN (5622, 5601, 5641, 5621, 5661)) .
}

```

GeoRDFBench, during experiment execution, *automatically constructs the prefix header to include system, queryset and dataset specific namespace prefixes* [3.3.3]. For example, while Strabon headers do not differ from the one presented above, the headers for GraphDB and Virtuoso also include the following system-specific namespace prefixes:

■ **Listing 6** GraphDB & Virtuoso Prefix Header Differences.

```

# GraphDB adds a few useful GeoSPARQL extensions based on the USeekM library
PREFIX ext: <http://rdf.useekm.com/ext#>
# Access to Virtuoso's built-in functions
PREFIX bif: <http://www.openlinksw.com/schemas/bif#>

```

To produce the ground queries of Listings 4 and 5 GeoRDFBench conveniently takes the necessary steps. First, the template parameters of the queries in Listing 12 are *replaced with their values during queryset deserialization*. After that, it *applies any system provided query*

translations in order to customize the ground queries [3.2.3, 3.4.1]. For example, Stardog emulates the GeoSPARQL `geof:sfIntersects` function with the combination of the non-standard `geof:relate` through the negation of `geo:disjoint` as exhibited below for query SC1:

■ **Listing 7** Stardog Queries SC1 Differences.

```
# Query 0 - SC1_Geometries_Intersects_GivenPolygon:
SELECT ?s1 ?o1 WHERE {
  ?s1 geo:asWKT ?o1 .
  ?relation geof:relate(?o1 "POLYGON((23.708496093749996 37.95719224376526,
... 37.95719224376526))"^^<http://www.opengis.net/ont/geosparql#wktLiteral> .
  FILTER(?relation != geo:disjoint) . }
```

Such system dependent logic, which may include different vocabularies, has been identified and implemented in the corresponding `translateQuery()` function of the relevant RDF Module SUTs for the user's convenience and provide a more consistent and trouble free experience. Stardog-SUT appropriately translates the unsupported GeoSPARQL `geof:sfWithin` and `geof:sfEquals` functions. GraphDBSUT handles the non-standard `geof:sfEquals` function behaviour. VirtuosoSUT maps `geof:sfWithin`, `geof:buffer`, `geof:distance`, `geof:sfEquals` functions to `bif:st_within`, `bif:st_point`, `bif:st_distance`, `bif:st_within`, the `geo:wktLiteral` data type to `virtrdf:Geometry` and handles default distance unit differences between functions.

The execution specification used for the scalability queryset is the same as in Listing 13.

### 7.3 Experiment Executions

As explained, we have 7 systems to test, however two variants of the scalability queryset need to run against the repositories of the GeoSPARQL enabled GraphDB. Therefore there is a total of **eight** (8) different system configurations to be tested against three increasingly bigger workloads. Since each system configuration needs to run against the four scalability workloads (10K, 100K, 1M, 10M) there are **thirty two** (32) experiment executions with unique database ID, system configuration name and configuration description are depicted in Table 3.

■ **Table 3** System Configuration Experiments.

| ExpIDs        | ConfigName           | System                            | ConfigDesc                           |
|---------------|----------------------|-----------------------------------|--------------------------------------|
| 1-3, 106      | <b>RDF4J</b>         | RDF4J                             | v4.3.15                              |
| 4-6, 111      | <b>JenaGeoSPARQL</b> | Jena GeoSPARQL                    | v4.10.0, <i>STRtree</i>              |
| 7-9, 108      | <b>GraphDB</b>       | GraphDB                           | v10.8.5                              |
| 10,11,14, 121 | <b>Virtuoso</b>      | Virtuoso Open Server              | v7.2.14, <i>2D R-tree</i>            |
| 15-17, 109    | <b>GraphDB+</b>      | GraphDB + <i>GeoSPARQL plugin</i> | v10.8.5, <i>Quad-11</i>              |
| 18-20, 110    | <b>GraphDB+P</b>     | GraphDB + <i>GeoSPARQL plugin</i> | v10.8.5, <i>Quad-11, Pred</i>        |
| 21-23, 107    | <b>RDF4J+</b>        | RDF4J + <i>Lucene Sail</i>        | v4.3.15, <i>Lucene spatial index</i> |
| 31-33, 124    | <b>Strabon</b>       | Strabon                           | 3.3.3-SNAPSHOT                       |

GeoRDFBench provides the following common tools [3.4.2, 3.5]: (i) the **environment preparation script** `geordfbench/scripts/prepareRunEnvironment.sh` which prepares the environment variables for running the RDF Module specific scripts for a given host, (ii) the **print environment script** `geordfbench/scripts/printRunEnvironment.sh` which allows reviewing the environment variables previously set.

Each RDF Module provides the following scripts which have largely the same functionality: (i) the **repository generation wrapper script** which creates all enabled repositories, imports RDF data into them and spatially indexes them, (ii) the **compact workload run script** which tests the system against the compact representation of a workload and (iii) the **detailed workload run script** which tests the system against the detailed representation of a workload.

## 8:26 Benchmarking with GeoRDFBench Framework

The repository generation wrapper scripts require the user provides all the required arguments for the system at hand and host critical locations, or that the environment variables have already been prepared with the environment preparation script, in which case the single required boolean parameter denotes whether to overwrite existing repositories or not. Each of these scripts comes, already setup, with a list of repositories to create automatically. The user will probably want to comment out the repositories that are not required or add new repositories to them.

The repository generation wrapper script calls the **repository generation script** for each repository [3.2.1, 3.2.2, 3.4.2]. This script is highly non-uniform as it engulfs each system's approach on optimizing data loading, thematic and spatial index building. For each system the most performant available loader (default/online vs bulk/offline) was selected even if that was a console tool instead of a Java library API call and the maximum statistics provided by it was collected. The key takeaways we discovered are: (i) Virtuoso's loader, does not provide a spatial index creation time, (ii) Stardog creates spatial index only during repo creation and requires different configuration settings for loading and querying, (iii) GraphDB Preload requires the server to be offline and cannot work with inference, (iv) JenaGeoSPARQL does not report a spatial index size because it does not persist a spatial index on disk. For the above reasons repository generation was not handled by the runtime API but pushed down to the module level and handled through scripts. When possible (RDF4J, Strabon) repository creation has been implemented at the module level and therefore can be performed programmatically through GeoRDFBench [3.5].

For the purpose of this work and to exhibit the use of these tools we created a single very simple **experiment execution script** for each one of the eight required executions. They basically call upon the GeoRDFBench common and the system-specific scripts in the correct order. They have an almost identical functionality with few minor additions for special cases. The log files generated by these scripts are three types: (i) `envvars_<sys>.log` which record the environment variables reported by the print environment script, (ii) `logCreateRepos_Scal10K_1M_<sys>.log` which store the repository generation wrapper script output and (iii) `RunWL<sys>Exp_Scal10K.log` which record the compact or detailed workload run script output. Below we see a compact (comment and whitespace sanitized) version of RDF4J's script:

### ■ Listing 8 RDF4J Experiment execution script.

```
#!/bin/bash
CWD='pwd'
cd /data/geordfbench/scripts
# environment preparation script sets environment variables for RDF4JSUT and nuc8i7beh host
source prepareRunEnvironment.sh nuc8i7beh RDF4JSUT "CreateRepo_Scalability10K_1M_RDF4J"
# print environment script logs environment variables for RDF4JSUT and nuc8i7beh host
./printRunEnvironment.sh >> /data/envvars_rdf4j.log

BASE_LOG_DIR=/data/LOGS
REPO_CREATION_LOG_DIR=${BASE_LOG_DIR}/RepoCreation/${ActiveSUT}
EXP_RUN_LOG_DIR=${BASE_LOG_DIR}/ExperimentRun/${ActiveSUT}
mkdir -p ${REPO_CREATION_LOG_DIR}; mkdir -p ${EXP_RUN_LOG_DIR}

cd ../RDF4JSUT/scripts/CreateRepos/
# Enable 100K and 1M scalability dataset generation
sed -i -e 's@levels=( "10K" )@levels=( "10K" "100K" "1M" )@g' createAllRDF4JRepos.sh
# repository generation wrapper script creates repos, loads data, spatial indexes them
./createAllRDF4JRepos.sh false 2>&1 | tee -a ${REPO_CREATION_LOG_DIR}/logCreateRepos_Scal10K_1M_RDF4J.log

DateTimeISO8601='date --iso-8601='date'' # Record current date
cd ../RunTests3/
# compact workload run script runs RDF4J against the Scalability 10K workload
./runWLTesForRDF4JSUT.sh -Xmx24g \
  -rbd ${RDF4JRepoBaseDir}/${EnvironmentBaseDir}/ -expdesc ${DateTimeISO8601}_RDF4JSUT_RunWL_Scal10K \
  -wl ${GeoRDFBenchJSONLibDir}/workloads/scalabilityFunc10K_WLoriginal_GOLD_STANDARD.json \
  -h ${GeoRDFBenchJSONLibDir}/hosts/nuc8i7behHOSToriginal.json \
  -rs ${GeoRDFBenchJSONLibDir}/reportspecs/simplereportspec_original.json \
  -rpsr ${GeoRDFBenchJSONLibDir}/reportsources/nuc8i7behHOSToriginal.json 2>&1 | tee -a
    ${EXP_RUN_LOG_DIR}/RunWLRDF4JExp_Scal10K.log
# compact workload run script runs RDF4J against the Scalability 100K workload
# compact workload run script runs RDF4J against the Scalability 1M workload
# ..
```

Excluding file and path locations, the difference of the experiment execution script for RDF4J compared to RDF4J+ variant is the line enabling the Lucene Sail. Similarly, the difference between the GraphDB and GraphDB+ is also a single line enabling the GeoSPARQL plugin which could require 2 more assignments if a different spatial algorithm and/or accuracy were needed. The above experiment execution script differences can be found in Appendix B Listings 14,15.

## 7.4 Experiment log files

### 7.4.1 Environment variables log

There is one such log for each system configuration [3.1.4]. It has a homogeneous output layout which comprises two sections. The first one contains environment variables common for all systems, such as, host name, IP address and max memory, locations for dataset source files, standard output results and GeoRDFBench installation-related. The second section consists of system-specific variables such as, version number, server installation (if present) and repositories' locations. Below we see part of the JenaGeoSPARQL output:

■ **Listing 9** JenaGeoSPARQL Environment variables.

```
All SUTs
-----
Environment = NUC8I7BEH
EnvironmentBaseDir = /data
GeoRDFBenchScriptsDir = /data/geordfbench/scripts
GeoRDFBenchJSONLibDir = /data/geordfbench/json_defs
DatasetBaseDir = /data/Geographica2_Datasets
...
ResultsDirName = 531f9c1#_2025-04-26_JenaGeoSPARQLSUT_CreateRepo_Scalability10K_1M_JenaGeoSPARQL
ActiveSUT = JenaGeoSPARQLSUT
ExperimentDesc = 531f9c1#_2025-04-26_JenaGeoSPARQLSUT_CreateRepo_Scalability10K_1M_JenaGeoSPARQL
SystemMemorySizeInGB = 32 GBs
JVM_Xmx = -Xmx24g

JenaGeoSPARQL SUT
-----
JenaBaseDir = /data/apache-jena-4.10.0
JenaGeoSPARQLRepoBaseDir = /data/JenaGeoSPARQL_4.10.0_Repos
Version = 4.10.0
```

### 7.4.2 Repository generation log

There is one such log for each system configuration [3.1.4]. It has the least homogeneous output since each system's preferred loader uses significantly different strategies which are reflected in the output. Apart from very useful details provided which may allow for fine-tuning each system, it provides two key pieces of information which are the *final repository size* and the *required time to complete repository creation, data loading and data indexing* for all enabled repositories. The RDF4JSUT+ output for the *scalability\_10K* repository can be found in Appendix B Listing 16.

### 7.4.3 Run workload log

Each such log concerns a single run of a system against a workload. It has homogeneous output layout which comprises three sections [3.1.4]. In the first section all validated arguments are listed, the JSON specifications (which can be either JSON files or URL calls to the JSON Library REST API endpoint) are deserialized to create the workload, host and system related components, the system query timeout is set dynamically based on the execution specification, if needed the queryset is translated according to the system and the final namespace prefix header is formed by merging dataset, queryset and system dependent prefixes. A sanitized version of the first section for Virtuoso's run against the Scalability 1M workload can be found in Appendix B Listing 17.

## 8:28 Benchmarking with GeoRDFBench Framework

The second section details each query's execution steps. It begins with the first iteration of the first query SC1 with COLD caches. The ground query is displayed, system caches are cleared since it is a COLD cache execution and after a delay for Java garbage collection, the experiment launches an `RDF4JbasedExecutor` instance in a new thread for executing the query in a time constrained manner. The child process reports back every 6 hours (21600000 msec) which is 25% of the total query timeout (24 hours)<sup>18</sup> specified in the execution specification. The executor reports all state transitions for the query execution and during the scanning phase reports the first 3 resultset entries according to the `simplereportspec_original.json` logging specification. Before exiting, the executor reports that it completed with no errors (COMPLETED-NONE), the query evaluation, scanning and total time in nano seconds, the number of results, the number of scan errors<sup>19</sup> and that the accuracy could not be verified, because the queryset specification provided did not include the expected number of results for this query. All query response times are net since they explicit aggregate only the query relevant portions of the code within the `run()` function of the executors' `Runnable` interface and exclude display, logging and when possible even exception handling logic. In Listing 10 we see a sanitized version of the second iteration of the execution of query SC2 (no 1) with COLD caches against the Scalability 100K workload.

■ **Listing 10** Virtuoso run log SC2 COLD Scalability 100K.

```
115549 [main] INFO ExperimentWorkload - |==> Executing query [1,
      SC2_Intensive_Geometries_Intersect_Geometries] (COLD, 1):
PREFIX bif: <http://www.openlinksw.com/schemas/bif#>
...
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?s1 ?s2
WHERE {
  ?s1 geo:hasGeometry [ geo:asWKT ?o1 ] ;
  lgo:has_code "1001"^^xsd:integer .
  ?s2 geo:hasGeometry [ geo:asWKT ?o2 ] ;
  lgo:has_code ?code2 .
  FILTER(?code2>5000 && ?code2<6000 && ?code2 != 5260) .
  FILTER(geof:sfIntersects(?o1, ?o2)).
}
...
120969 [main] INFO UbuntuJammyOS - Caches cleared after delay of 5000 msec
120970 [main] INFO VirtuosoSUT - Starting Virtuoso server...
133318 [main] INFO VirtuosoSystem - Initialized VirtuosoRepository has query timeout = 86400
133352 [main] INFO VirtuosoSUT - Starting QueryExecutor thread
133353 [main] INFO VirtuosoSUT - Waiting for QueryExecutor thread to finish
133353 [main] INFO VirtuosoSUT - Timeout progress step is 21600000 msec
133353 [Thread-5] INFO QueryRepResult - Transitioning (NOTSTARTED => STARTED)
133354 [Thread-5] INFO QueryRepResult - Transitioning (STARTED => PREPARING)
133355 [Thread-5] INFO QueryRepResult - Transitioning (PREPARING => EVALUATING)
135404 [Thread-5] INFO QueryRepResult - Transitioning (EVALUATING => EVALUATED)
135405 [Thread-5] INFO RDF4JbasedExecutor - s1 s2
135405 [Thread-5] INFO RDF4JbasedExecutor - <----->
135405 [Thread-5] INFO QueryRepResult - Transitioning (EVALUATED => SCANNING)
135405 [Thread-5] INFO RDF4JbasedExecutor - http://data.linkeddata.eu/osm/wales/places/id/335184
http://data.linkeddata.eu/osm/wales/transport/id/123360606
135405 [Thread-5] INFO RDF4JbasedExecutor - http://data.linkeddata.eu/osm/wales/places/id/335184
http://data.linkeddata.eu/osm/wales/transport/id/123360628
135405 [Thread-5] INFO RDF4JbasedExecutor - http://data.linkeddata.eu/osm/wales/places/id/335184
http://data.linkeddata.eu/osm/wales/transport/id/1396066034
135766 [Thread-5] INFO QueryRepResult - Transitioning (SCANNING => SCANNED)
135766 [Thread-5] INFO RDF4JbasedExecutor - <----->
135766 [Thread-5] INFO QueryRepResult - Transitioning (SCANNED => COMPLETED)
135766 [main] INFO VirtuosoSUT - Percentage of expired timeout is 0.0 %
135776 [main] INFO ExperimentWorkload - |<== Executed query [1, SC2_Intensive_Geometries_Intersect_Geometries]
(COLD, 1): <COMPLETED-NONE> 2050015972 + 360751232 = 2410767204 nsecs, 239 results, 0 scan errors -
ACCURACY NOT DETERMINED
```

<sup>18</sup>The Scalability query timeout is very high because of the very high query response times in the 500M dataset.

<sup>19</sup>Usually due to invalid geometries or unsupported operators.

In the third section, we have the result collector’s actions which involves flushing the [2 cache types \* 3 queries \* 3 iterations = 18] cached results in the report sink (PostgreSQL writes in *geordfbench* database in deferred mode, while in H2 results are written in synchronous mode). The collector also generates the standard result and statistics files in the file system and finally reports the gross script run time. A reduced version of the Virtuoso’s run against the Scalability 10K workload is shown in Appendix B Listing 18.

## 7.5 Evaluation Results

Tables 4-6 present the evaluation results for the GeoRDFBench experiments [3.1.1-3, 3.2.1].

### 7.5.1 Repository size

Virtuoso is the most scalable store of all, when taking into account that its repos include a *2D R-tree* persisted spatial index. Its 10M repo is smaller than Jena GeoSPARQL’s repo which does not persist a spatial index on disk and only 15% larger than the RDF4J’s and GraphDB’s repos which do not feature a spatial index at all. The fact that Virtuoso exhibits much larger repo sizes for small to medium datasets, points out that its internal database and index formats carry an almost constant overhead which is significant only in smaller databases and for more real case scenarios can safely be ignored. RDF4J’s *Nativestore* [69] (B-Tree indexed, no spatial index) performs best in small to medium dataset sizes and very well for tens of millions of triples which is on a par with its specification. GraphDB’s default configuration is storage efficient across all input dataset sizes. For small datasets it ranks average having a substantial overhead compared to RDF4J which however practically becomes irrelevant after the one million triples limit. In absolute terms it is the most scalable system for repository size, however we need to point out that similarly to RDF4J’s *Nativestore* there is no spatial index which will incur additional search costs for high selectivity spatial queries. JenaGeoSPARQL’s *TDB2* despite persisting 3 spatial indexes in memory and not on disk, seems to have a scalability issue. For smaller datasets it performs near the top of the competition, it then gradually degrades to an average score for datasets around one million triples and for bigger datasets it generates larger repositories even compared to the majority of systems with a persisted spatial index. By comparing Table 4 with Table 1 we also notice that the final repository sizes of JenaGeoSPARQL are almost identical to the input dataset sizes. In RDF4J+ the *Lucene* quad spatial index, increases the storage footprint by 45% on average with respect to RDF4J’s *Nativestore*. GraphDB+’s *GeoSPARQL plugin* uses the same *Lucene* library as RDF4J+ for spatial indexing, also increases the storage footprint by 45% but scales better than it in a similar manner that GraphDB outperformed RDF4J. The underlying reason for this behaviour is that despite GraphDB being built on top of RDF4J it brings in an augmented design and optimizations that start to pay off for larger dataset sizes. Strabon’s storage footprint is the largest of all system configuration due to the *PostGIS* database with an *R-tree over GiST* [29] spatial index.

### 7.5.2 Ingestion time

Table 5 presents the data ingestion times. When separate measurements for data loading and spatial indexing could be acquired they are provided. If spatial indexing was available but could not be measured the value was left empty while a hyphen (-) REPRESENTS stores that do not build a spatial index.

In absolute terms, RDF4J has the fastest ingestion times for small to medium dataset sizes and performs very well for tens of millions of triples. With the addition of *Lucene* spatial index RDF4J+ more than doubles the total ingestion time for smaller datasets and at best doubles it



## 8:30 Benchmarking with GeoRDFBench Framework

■ **Table 4** Scalability Workload Evaluation - Repository sizes (MB).

| System ConfigName | Spatial Index | 10K triples | 100K triples | 1M triples | 10M triples |
|-------------------|---------------|-------------|--------------|------------|-------------|
| RDF4J             | –             | 4           | 14           | 135        | 1262        |
| JenaGeoSPARQL     | memory        | 5           | 22           | 207        | 1977        |
| GraphDB           | –             | 28          | 38           | 144        | 1151        |
| Virtuoso          | disk          | 74          | 78           | 504        | 1338        |
| GraphDB{+,+P}     | disk          | 32          | 44           | 201        | 1637        |
| RDF4J+            | disk          | 7           | 21           | 205        | 1813        |
| Strabon           | disk          | 24          | 52           | 349        | 3078        |

for the largest one. JenaGeoSPARQL is second best for the first three datasets but its scalability potential starts to suffer at around ten million triples. For all GraphDB variants the data loading portion is almost identical and shows a time overhead for the first datasets. This is due to GraphDB’s *Preload* [59] offline tool. This two-phase strategy bulk loader has a large overhead, evident in the small and medium datasets, but performs increasingly better as dataset sizes grow. This is confirmed in the largest dataset where GraphDB exhibits the best scalability. The Lucene based *GeoSPARQL plugin* in GraphDB+ variant more than doubles the total ingestion time. Strabon’s three phase<sup>20</sup> bulk loader, *StrabonLoader*, greatly improves the data loading and indexing time over the standard OpenRDF Sesame based loader, however lags far behind the other systems. Its basic characteristic is that ingestion time is proportional to the dataset size and that is mainly due to the large time spent in phase one, RDF to CSV conversion, which is performed with the Redland raptor parser and serialization library. The average score of Virtuoso in the smaller datasets cannot overshadow the astonishing fact that it almost tied GraphDB’s no-spatial index performance while building a spatial index. To put this into perspective, for the 10M triple dataset all other systems required more than 95 sec for building just the spatial index, while Virtuoso completed data ingestion and spatial indexing in approximately the same time.

■ **Table 5** Scalability Workload Evaluation - Data ingestion (loading, spacial indexing) times (sec).

| System ConfigName | 10K triples |          |       | 100K triples |          |       | 1M triples |          |        | 10M triples |          |         |
|-------------------|-------------|----------|-------|--------------|----------|-------|------------|----------|--------|-------------|----------|---------|
|                   | Loading     | Indexing | Total | Loading      | Indexing | Total | Loading    | Indexing | Total  | Loading     | Indexing | Total   |
| RDF4J             | 0.62        | –        | 0.62  | 1.80         | –        | 1.80  | 11.14      | –        | 11.14  | 105.38      | –        | 105.38  |
| JenaGeoSPARQL     | 2.49        | –        | 2.49  | 3.91         | –        | 3.91  | 16.61      | –        | 16.61  | 121.90      | –        | 121.90  |
| GraphDB           | 14.89       | –        | 14.89 | 11.82        | –        | 11.82 | 17.59      | –        | 17.59  | 97.68       | –        | 97.68   |
| Virtuoso          | 15.87       | –        | 15.87 | 17.88        | –        | 17.88 | 28.07      | –        | 28.07  | 99.71       | –        | 99.71   |
| GraphDB{+,+P}     | 15.85       | 28.97    | 44.82 | 10.26        | 37.77    | 48.03 | 18.92      | 40.21    | 59.13  | 88.85       | 103.85   | 192.70  |
| RDF4J+            | 0.62        | 1.98     | 2.60  | 1.80         | 2.62     | 4.41  | 11.14      | 12.49    | 23.63  | 105.38      | 94.29    | 199.66  |
| Strabon           | 3.42        | 17.00    | 20.42 | 26.91        | 24.00    | 50.91 | 234.32     | 53       | 287.32 | 2504.14     | 371.00   | 2875.14 |

### 7.5.3 Accuracy & Response time

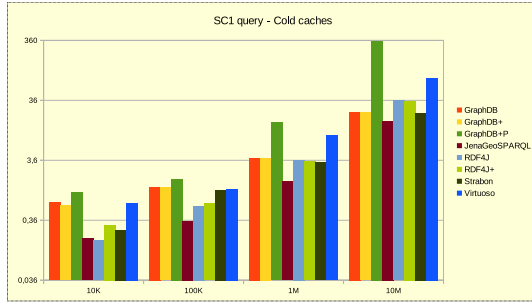
Table 6 shows the number of results returned and response times of the eight systems for each one of the workload, query and cache type combinations. The response times are the median values out of three executions. Figures 6-8 show chart diagrams with the system response times for the three queries [3.1.4]. All results were accurately calculated by all systems with one exception. For the spatial selection query **SC1**, for the Scalability 1M workload, Virtuoso calculated **80501** instead of 80500 results returned by all other systems. This is probably due to some rounding error in the 2D R-tree spatial indexing process and/or in the implementation of `geof:sfIntersects` GeoSPARQL function which mishandles a border result.

<sup>20</sup> RDF to CSV, CSV to PostGIS tables, indexing tables

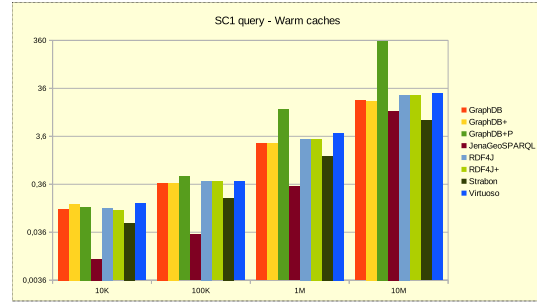


■ **Table 6** Scalability Workload Evaluation - Accuracy & Response Times.

| Cache | Query | Workload | RDF4J  |         | JenaGeoSPARQL |         | GraphDB |          | Virtuoso |         | GraphDB+ |          | GraphDB+P |          | RDF4J+ |         | Strabon |         |
|-------|-------|----------|--------|---------|---------------|---------|---------|----------|----------|---------|----------|----------|-----------|----------|--------|---------|---------|---------|
|       |       |          | # Res  | Time(s) | # Res         | Time(s) | # Res   | Time(s)  | # Res    | Time(s) | # Res    | Time(s)  | # Res     | Time(s)  | # Res  | Time(s) | # Res   | Time(s) |
| Cold  | SC1   | 10K      | 554    | 0.167   | 554           | 0.179   | 554     | 0.710    | 554      | 0.703   | 554      | 0.634    | 554       | 1.036    | 554    | 0.298   | 554     | 0.246   |
|       |       | 100K     | 6278   | 0.623   | 6278          | 0.350   | 6278    | 1.294    | 6278     | 1.202   | 6278     | 1.255    | 6278      | 1.747    | 6278   | 0.683   | 6278    | 1.148   |
|       |       | 1M       | 80500  | 3.639   | 80500         | 1.609   | 80500   | 3.918    | 80501    | 9.359   | 80500    | 3.943    | 80500     | 15.376   | 80500  | 3.408   | 80500   | 3.295   |
|       |       | 10M      | 442580 | 35.818  | 442580        | 16.371  | 442580  | 23.008   | 442580   | 84.408  | 442580   | 22.549   | 442580    | 351.622  | 442580 | 35.028  | 442580  | 22.129  |
|       | SC2   | 10K      | 2      | 0.135   | 2             | 0.090   | 2       | 3.320    | 2        | 0.575   | 2        | 0.575    | 2         | 3.439    | 2      | 1.031   | 2       | 0.254   |
|       |       | 100K     | 239    | 6.209   | 239           | 0.214   | 239     | 22.907   | 239      | 2.411   | 239      | 23.256   | 239       | 2.682    | 239    | 6.838   | 239     | 0.771   |
|       |       | 1M       | 813    | 23.995  | 813           | 4.392   | 813     | 213.015  | 813      | 21.658  | 813      | 207.580  | 813       | 26.186   | 813    | 25.539  | 813     | 3.373   |
|       |       | 10M      | 9160   | 568.296 | 9160          | 176.511 | 9160    | 1957.782 | 9160     | 189.685 | 9160     | 1886.617 | 9160      | 388.638  | 9160   | 499.867 | 9160    | 28.72   |
|       | SC3   | 10K      | 2      | 0.124   | 2             | 0.053   | 2       | 0.732    | 2        | 0.580   | 2        | 0.697    | 2         | 1.404    | 2      | 0.206   | 2       | 0.154   |
|       |       | 100K     | 239    | 6.136   | 239           | 0.183   | 239     | 8.816    | 239      | 2.415   | 239      | 8.802    | 239       | 6.238    | 239    | 6.719   | 239     | 0.695   |
|       |       | 1M       | 239    | 15.347  | 239           | 4.072   | 239     | 8.943    | 239      | 21.663  | 239      | 8.715    | 239       | 117.763  | 239    | 17.546  | 239     | 3.474   |
|       |       | 10M      | 239    | 215.219 | 239           | 176     | 239     | 41.842   | 239      | 192.008 | 239      | 40.461   | 239       | 1937.18  | 239    | 135.291 | 239     | 49.696  |
| Warm  | SC1   | 10K      | 554    | 0.115   | 554           | 0.010   | 554     | 0.111    | 554      | 0.142   | 554      | 0.141    | 554       | 0.122    | 554    | 0.101   | 554     | 0.056   |
|       |       | 100K     | 6278   | 0.419   | 6278          | 0.032   | 6278    | 0.384    | 6278     | 0.411   | 6278     | 0.379    | 6278      | 0.529    | 6278   | 0.411   | 6278    | 0.185   |
|       |       | 1M       | 80500  | 3.136   | 80500         | 0.322   | 80500   | 2.524    | 80501    | 4.229   | 80500    | 2.577    | 80500     | 13.122   | 80500  | 3.160   | 80500   | 1.399   |
|       |       | 10M      | 442580 | 25.941  | 442580        | 12.25   | 442580  | 20.091   | 442580   | 28.146  | 442580   | 19.794   | 442580    | 350.648  | 442580 | 25.307  | 442580  | 7.692   |
|       | SC2   | 10K      | 2      | 0.102   | 2             | 0.007   | 2       | 2.364    | 2        | 0.158   | 2        | 0.158    | 2         | 2.405    | 2      | 0.084   | 2       | 0.007   |
|       |       | 100K     | 239    | 5.991   | 239           | 0.079   | 239     | 22.781   | 239      | 1.717   | 239      | 22.300   | 239       | 0.997    | 239    | 6.689   | 239     | 0.011   |
|       |       | 1M       | 813    | 19.755  | 813           | 3.252   | 813     | 213.212  | 813      | 17.335  | 813      | 207.504  | 813       | 23.082   | 813    | 24.454  | 813     | 0.086   |
|       |       | 10M      | 9160   | 555.843 | 9160          | 158.535 | 9160    | 1950.94  | 9160     | 151.939 | 9160     | 1896.749 | 9160      | 396.026  | 9160   | 484.631 | 9160    | 0.471   |
|       | SC3   | 10K      | 2      | 0.101   | 2             | 0.007   | 2       | 0.110    | 2        | 0.161   | 2        | 0.111    | 2         | 0.416    | 2      | 0.105   | 2       | 0.006   |
|       |       | 100K     | 239    | 6.089   | 239           | 0.080   | 239     | 7.989    | 239      | 1.718   | 239      | 7.976    | 239       | 4.803    | 239    | 6.613   | 239     | 0.011   |
|       |       | 1M       | 239    | 11.303  | 239           | 3.288   | 239     | 8.086    | 239      | 17.437  | 239      | 8.085    | 239       | 116.961  | 239    | 15.621  | 239     | 0.078   |
|       |       | 10M      | 239    | 199.875 | 239           | 169.945 | 239     | 41.285   | 239      | 151.545 | 239      | 40.247   | 239       | 1980.143 | 239    | 116.536 | 239     | 1.144   |



(a)



(b)

■ **Figure 6** Response times – SC1 spatial selection query.

JenaGeoSPARQL and Strabon are the most performant systems in this comparison. JenaGeoSPARQL scores most points for the two smaller datasets independent of caches. It also comes on top for the 1M triple dataset for spatial selection queries independent of caches. With cold caches and spatial selection queries it performs the best even in the largest dataset. Strabon on the other hand is the strongest competitor for warm cache executions and for spatial join queries on larger datasets regardless of caching. RDF4J also got a point for the spatial selection query with cold caches against the smallest 10K dataset but we note that it performs predictably through out the entire spectrum. Warm cache times are on average 10% faster than cold ones, response times for higher selectivity join SC3 are smaller than lower selectivity join SC2 which means that filters are considered early in the query plan and finally response times scale in an analog manner when repository size grows. RDF4J+ provided improvements over RDF4J only in the spatial join queries and for the largest dataset. Improvement ranges between 12% for the low selectivity spatial join SC2 and 42% for the higher selectivity join SC3. Thus the extra repository size and additional ingestion time for RDF4J+ maybe justified for specific use cases. Virtuoso performs adequately but its behaviour is not entirely predictable. While response times scale in an analog manner when repository size grows and warm cache times are on average 20% faster than cold ones, the response times for higher selectivity join SC3 are similar to the ones for lower selectivity join SC2 which means that it has a low selectivity threshold for engaging filters early in the query plan. GraphDB exhibited two characteristics, lack of sensitivity to caching when selectivity is not low and the lowest response times for spatial joins with low selectivity such as SC2. Enabling the Lucene-based

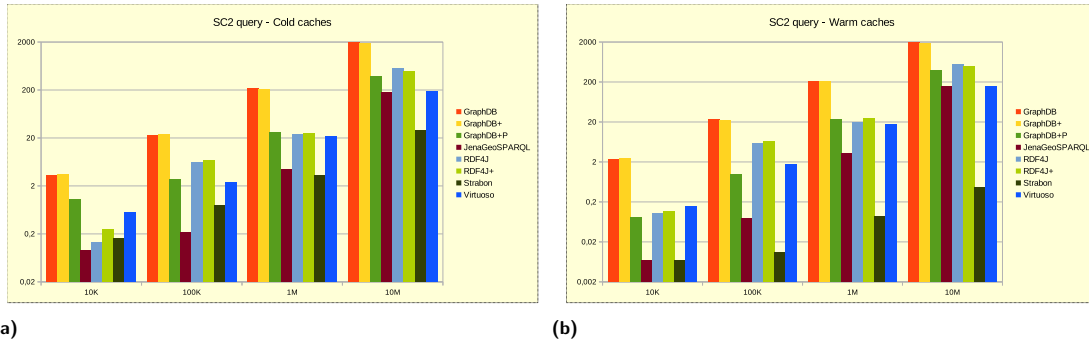


Figure 7 Response times – SC2 spatial join query.

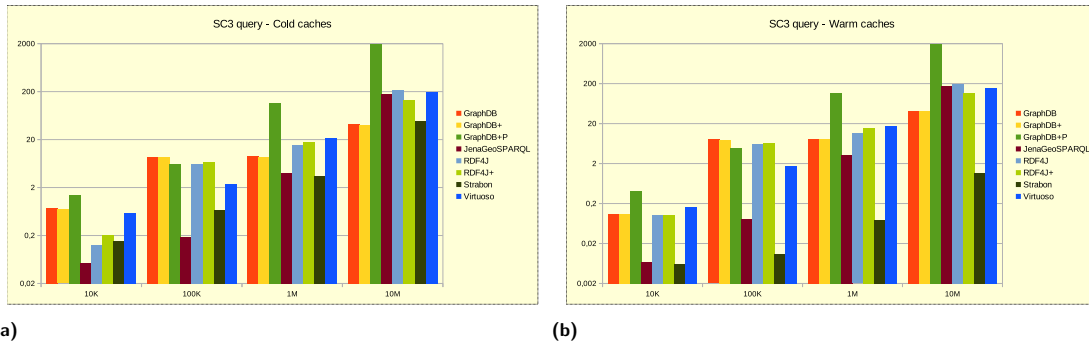


Figure 8 Response times – SC3 spatial join query.

GeoSPARQL plugin and using the standard spatial function querysets, the GraphDB+ offered borderline benefits to the worst scenarios with larger datasets. Using the spatial predicate version of the querysets, as instructed in the official documentation [58], made possible the effective use of the GeoSPARQL plugin’s spatial index and drastically improved the GraphDB+P variant’s SC2 response times by a factor of **x7**. However this negatively affected the times for SC1 and SC3 queries, in some cases by a factor of (**x7**). This variant also provided small improvements in its sensitivity to caching. Using the *variance* column of *vquery\_ordered\_aggrs\_4* database view on query response times, we can provide a few final notes concerning the consistency of the various systems tested. GraphDB+P on SC2 with cold caches and JenaGeoSPARQL on SC2 and SC3 irrespective of caches exhibited the highest variance especially in the two bigger 1M and 10M datasets. All other system, query, cache and dataset combinations had a consistent query response time across iterations.

The results of the above benchmark are available at Github [36] and Zenodo [35] for verification.

#### 7.5.4 Experiment Summary

Considering all previous measurements (ingestion time, repository size and response time) we conclude that Virtuoso and JenaGeoSPARQL behave in a more balanced and predictable manner than the rest of the system configurations. Therefore they are well suited as a first choice for use cases with a mix of spatial workloads. JenaGeoSPARQL performs better in the three smaller datasets while Virtuoso scales better over 10M triple datasets.

For use cases where query response time is of essence, spatial datasets are well above 1M triples and relatively static, Strabon is a good candidate. This is because despite its slow ingestion times and large repository sizes its response times with warm caches are at least **100x** faster

than the second contender. This is in sync with the Geographica 2 GeoSPARQL benchmark’s findings which affirmed Strabon’s query response scalability with datasets up to 500M triples (approximately 50 million complex geometries).

For use cases where ingestion time and repository size are critical factors GraphDB with the GeoSPARQL plugin disabled is a perfect candidate. Its lower performance on low selectivity spatial joins for larger datasets can be mitigated because of the versatility of the GeoSPARQL plugin. If the need arises then the plugin can be enabled with simple DML SPARQL statements providing a choice between two spatial indexing methods (quad, geohash) and adjustable precision. The user can then use spatial predicates instead of spatial functions for the low selectivity spatial joins while using spatial functions in the rest of their queries. At no extra cost the user can disable the plugin without losing the created spatial index.

RDF4J’s Nativestore has improved its scalability over its predecessor OpenRDF Sesame as its excellent performance for the first three datasets does not degrade by a significant factor for the 10M triple dataset. Its results position it as a very good choice for use cases with low to medium size spatial dataset needs and its SAIL layered architecture as an ideal base RDF store for research projects which can innovate with more scalable techniques. Unfortunately, the optional Lucene SAIL’s improvements in the largest dataset do not change RDF4J’s position compared to the competitors. With this in mind, we believe it does not justify the added repository size and ingestion time it requires.

## 7.6 Benchmarking with Docker – Examples

The GeoRDFBench web site [32] contains detailed tutorials and 4 Docker examples [3.4.4]. The “*Multiple stores against multiple Scalability Workloads*” example engages RDF4J, GraphDB and JenaGeoSPARQL against the Scalability 10K, 100K, 1M workloads and exhibits features, such as: (i) customization (**Method 2**) of queryset specification, (ii) query accuracy checking and (iii) custom sink reporting. In the “*Multiple stores against LUBM(1,0) Workload*” example GeoRDFBench tests RDF4J, GraphDB and JenaGeoSPARQL against the LUBM(1,0) SPARQL benchmark workload and exhibits features, such as: (i) generation (**Method 1**) of workload compact specification, (ii) query inclusion filtering, (iii) query accuracy checking and (iv) SPARQL benchmark compatibility. The “*Strabon and Virtuoso against the Synthetic Workload*” example engages Strabon and Virtuoso against the Geographica 2 *Synthetic-512* workload and exhibits features, such as: (i) query inclusion filtering, (ii) customization (**Method 2**) of execution specification, (iii) query timeout handling and (iv) custom sink reporting. All the above are shown in more detail in Appendix D Table 13.

## 7.7 Benchmarking Step-By-Step

In this part we will go through the docker<sup>21</sup> based example “*RDF4J, GraphDB and JenaGeoSPARQL against Scalability-10K, 100K, 1M Workloads*”. Basically, this is a subset of the experiments presented earlier. We will outline the steps taken and elaborate on some important aspects behind the scenes that might be of interest to users that would like to create their own custom benchmarks.

1. *Pulling the image*: The GeoRDFBench’s github repository has a docker registry from which we can pull the image `tioannid/geordfbench/multistore/scal10k_1m`.

```
$ docker pull ghcr.io/tioannid/geordfbench/multistore/scal10k_1m:latest
```

<sup>21</sup> A computer with Docker installation is required.

2. *Starting the container:* From the pulled image a container named *multiscal10k1m* can be started. We provide the hostname, IP address, RAM and number of virtual CPUs assigned to the container. We provide a mapping of the current host's working folder to the container's */src/* folder. Finally we expose the container's PostgreSQL listening port 5432 to the physical host's 5432 port and provide PostgreSQL's standard authentication credentials.

```
$ docker run -d -e POSTGRES_PASSWORD=postgres -p 5432:5432/tcp --hostname NUC8i7BEH \
--cpus="4" --memory="11g" --memory-swap="11g" \
--mount "type=bind,src='pwd',target=/src" --name multiscal10k1m \
ghcr.io/tioannid/geordfbench/multistore/scal10k_1m:latest
```

3. *Launch the experiments:* Through a terminal to the running container the user can launch the experiments and watch the creation of repos and experiment execution for each RDF store and each one of the 3 workloads.

```
$ docker exec -it multiscal10k1m /bin/bash
root@NUC8i7BEH:/data# ./startUpScript.sh
```

## 7.7.1 Docker Image Contents

### 7.7.1.1 Dockerfile

The docker image is built based on a *Dockerfile* which describes and installs the software prerequisites and takes any actions required to setup database server configuration and create any custom filesystem layout. The particular image installs Ubuntu Jammy, JDK 11, PostgreSQL 14 with its dependencies, Maven and Git. The actions it takes are:

- a. Setups a *systemd service* for PostgreSQL
- b. Copies under the container's */* folder the *startUpScript.sh*
- c. Copies under the container's */data* folder the following data:
  - i. GeoRDFBench Framework's source and binary code, under */data/geordfbench*
  - ii. The distribution binaries of the RDF stores to be tested, under */data/compressed*
  - iii. The RDF datasets required, under */data/Geographica2\_Datasets*
  - iv. Scripts to create the database for custom reports and statistics and run the experiments (repo creation, workload execution) for each RDF store, under the */data/scripts*

### 7.7.1.2 Start Up Script

This is a high level wrapper script that instruments the steps of the experiments' execution. It is also the best place to start understanding the experiment process in a top-down fashion. The script takes the following actions:

- a. Starts PostgreSQL service and create the database schema for custom reports and experiment statistics to be stored.
- b. For each RDF store calls a specific experiment script in */data/scripts/run\*.sh*

### 7.7.1.3 Experiment Execution Script

This is an script specific for each system, however it has the same logic which is summarized in the following steps:

- a. The environment preparation script *geordfbench/scripts/prepareRunEnvironment.sh* is called, which also prints and logs the environment variables set for the system.
- b. With the repository generation wrapper script creates the three Scalability repositories
- c. Invokes three times the compact workload run script of the system against each one of the Scalability workloads.

#### 7.7.1.4 Inspection Of Logs

The experiment execution script placed the logs under each system's base directory, but could very easily directed them at a custom location for centralized collection. In this case however the repository creation logs can be found in `/data/geordfbench/*SUT/scripts/CreateRepos` while the workload execution logs in `/data/geordfbench/*SUT/scripts/RunTests3`.

#### 7.7.1.5 Inspection Of Standard Statistics

The standard file-system based statistics that GeoRDFBench generates by default were placed in `/data/Results_Store` and the following sample listing clarifies the layout:

```
root@NUC8i7BEH:/data# tree -L 2 Results_Store/
Results_Store/
|-- RDF4JSUT
    |-- 2025-05-16_RDF4JSUT_RunWL_Scal100K
    |-- 2025-05-16_RDF4JSUT_RunWL_Scal10K
    '-- 2025-05-16_RDF4JSUT_RunWL_Scal1M
        '-- Scalability
            '-- 1M
                '-- RDF4JSUT-ExperimentWorkload
                    |-- 00-SC1_Geometries_Intersects_GivenPolygon-cold
                    |-- 00-SC1_Geometries_Intersects_GivenPolygon-cold-long
                    |-- 00-SC1_Geometries_Intersects_GivenPolygon-warm
                    |-- 00-SC1_Geometries_Intersects_GivenPolygon-warm-long
...

```

#### 7.7.1.6 Inspection Of Custom Report Store

The PostgreSQL database has been used as the report sink and all detailed experiment statistics have been sent to it. From the launched terminal that the startup script was run, we connect to the database and can query the basic reporting views. The `vqueryexecution3` shows detailed information (21 columns) about each iteration of query execution (iteration): the experiment, host, sut, queryset, dataset, query label, query no, cache type, query iteration, evaluation time, scan time, no of results, no of scan errors, evaluation flag, result exception, total time = evaluation + scan time, total time in seconds, was execution valid flag. The `vquery_ordered_aggrs_3` aggregates the above information (11 columns) and for query execution shows: experiment id, sut, queryset, dataset, query label, query no, was execution valid flag, cache type, no of iterations, mean of total time, median of total time. Users are free to modify the aggregate views in order to customize them to their needs e.g., adding more aggregate functions like variance and standard deviation for the total time or some other quantity collected during the experiments e.g., no of results.

```
root@NUC8i7BEH:/data# su - postgres
postgres@NUC8i7BEH:~$ psql
psql (14.17 (Ubuntu 14.17-0ubuntu0.22.04.1))
Type "help" for help.

postgres=# \c geordfbench
You are now connected to database "geordfbench" as user "postgres".
geordfbench=# SELECT * FROM public.vqueryexecution3;
geordfbench=# SELECT * FROM public.vquery_ordered_aggrs_3;
```

■ **Table 7** Benchmarking Coverage [3.1].

| Criteria                                  | 1. RDF Data Ingestion (Loading and Indexing)                                   | 2. Query Execution Performance and Accuracy                                                                                                                                      | 3. RDF Store Data Scalability and User Concurrency                                                                                        | 4. Statistics Reporting and Logging                                                                                                                                                                              |
|-------------------------------------------|--------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>GeoRDFBench Benchmarking Framework</b> | Yes, choice of best loader, split time and size for repo loading and indexing  | Yes, split query execution time for evaluation and scan phases, query fine error handling, query accuracy verification, cold/warm cache execution, continuous queryset execution | Yes, scalability workload (500M-triples 50M geometries), scalable synthetic data and query generator, simulated concurrent user execution | Yes, repo generation and query execution logs, system and host environment logs, sampling resultsets, customized report sinks (database and filesystem), custom statistics measurements, ground query generation |
| <b>LITMUS Benchmarking Framework</b>      | Partial, repository final size not reported                                    | Partial, no result accuracy, no query filtering, no query namespace prefixes handling, no query execution error handling/reporting, cold/warm execution                          | No, small datasets readily available                                                                                                      | Partial, file based reports and logs, no alternate report sink, plotting script available                                                                                                                        |
| <b>IGUANA Benchmarking Framework</b>      | Partial, repository final size and load time not reported, HTTP endpoints only | Adequate, evaluation from scan time cannot be distinguished, no result accuracy, query evaluation and scan errors not handled appropriately, continuous/warm execution           | Adequate, medium size datasets DBpedia 3.5.1 (232.5M triples) and SWDF (294.8K triples)                                                   | High, results stored as RDF or CSV files or endpoints, multiple statistics measurements                                                                                                                          |
| <b>HOBBIT Benchmarking Platform</b>       | No, user implemented                                                           | No, user implemented, warm cache execution, no query execution error handling/reporting                                                                                          | No, user provided                                                                                                                         | Yes                                                                                                                                                                                                              |
| <b>Geographica 2 GeoSPARQL Benchmark</b>  | Adequate                                                                       | Adequate, cold/warm cache execution, continuous queryset execution                                                                                                               | High, scalability workload (500M-triples 50M geometries), scalable synthetic data and query generator                                     | Adequate, repo generation and query execution logs, system and host environment logs, sampling resultsets, filesystem report and logs                                                                            |

## 8 Comparisons of GeoRDFBench

In this section we present key features, limitations and current level of support for the benchmarking frameworks LITMUS, IGUANA, the Geographica 2 benchmark and the benchmarking platform HOBBIT. This will help the reader understand some of the key contributions of GeoRDFBench over each one of the above solutions. In order to streamline this process, when relevant, the comparison criteria from Section 3 are referenced by number in square brackets after the relevant text. Tables 7-11 map each benchmarking solution's coverage (strength) of Section's 3 criteria.

### 8.1 LITMUS Benchmarking Framework

LITMUS is an early effort of creating a framework which aims at uniform benchmarking of data management solutions (DMS) supporting different query languages, such as SPARQL for RDF stores, Gremlin for LPG stores and SQL for traditional DBMSs. Its deliverables are: (i) *Gremlinator* [76], a SPARQL-to-Gremlin query translator, (ii) the `litmus-docker` [76] image for running experiments on 4 LPGs systems, 4 RDF stores and plotting the results and (iii) a Python utility which converts the SPARQL query JSON resultset from a DBpedia endpoint to GraphML.

It can be classified as a bare bones framework since it provides a basic level of automation with a collection of scripts, organized by system, for loading data and submitting queries in various languages such as Bash, Python and Gremlin Groovy. The main entry point is a large (approximately 2000 lines) monolithic Python script `run_script.py` where magic literals are used extensively e.g., LPG system folders are prepended with “g\_”, RFD stores with “r\_” and system scripts have to follow a rigid naming convention to distinguish actions that relate to them. There is no common engine to implement execution logic variants e.g., easily changing the order and number of cold or warm cache executions, handle queryset, dataset and/or system specific namespace prefixes. Instead there is one script for each system and benchmarking operation to test, however

■ **Table 8** Geospatial Support [3.2].

| Criteria                           | 1. Spatial Index Creation Time and Size | 2. Spatial Index Specification                             | 3. GeoSPARQL Support and Compliance                      | 4. GeoSPARQL Spatial Indexing Challenges | 5. Spatial Index Enabled GeoSPARQL Query Execution |
|------------------------------------|-----------------------------------------|------------------------------------------------------------|----------------------------------------------------------|------------------------------------------|----------------------------------------------------|
| GeoRDFBench Benchmarking Framework | Yes                                     | Yes, choice of method, accuracy and time of index creation | Yes + query translation for system-specific vocabularies | Yes                                      | Yes                                                |
| LITMUS Benchmarking Framework      | No                                      | No                                                         | No                                                       | No                                       | No                                                 |
| IGUANA Benchmarking Framework      | No                                      | No                                                         | No                                                       | No                                       | No                                                 |
| HOBBIT Benchmarking Platform       | No, user provided                       | No, user provided                                          | No, user provided                                        | No, user provided                        | No, user provided                                  |
| Geographica 2 GeoSPARQL Benchmark  | Yes                                     | Yes, choice of method, accuracy and time of index creation | Yes + query translation for system-specific vocabularies | Yes                                      | Partial                                            |

many of them are trivial (2-4 lines), repetitive (see each system’s cold and warm cache scripts) and offer a primitive parameterized wrapper script over each systems’ own utilities e.g., bulk loaders. Finally, there is a script for plotting the loading and execution times.

The “Parameters Measured” portion of the github `litmus-docker` repository mentions a large number of low-level (CPU/PMU) hardware level metrics for which there is no evidence of what practical use they offer in the graph store benchmarking evaluation. This is done by virtue of using the Linux profiler `perf` Python package in specific cloned scripts. Since this package explicitly states that it runs only on Linux<sup>22</sup> this basically implies that running LITMUS natively, for example in Windows and not in a docker container, will not have its full functionality.

The criterion category [3.1] “Benchmarking Coverage” is partially covered because of these missing features: (i) *repository final size is not reported*, (ii) there is no way to differentiate query evaluation time from scan time, (iii) there is *no result accuracy checking* measurement, (iv) there is *no query inclusion/exclusion filters* for running query subsets, (v) *cannot handle namespace prefixes at the system, queryset or dataset level*, (vi) query evaluation and/or scan errors are not reported, (vii) there is no active error handling and no related policy to respond to them, (viii) scalability cannot be tested with the available small datasets, (ix) all results are file based and no alternate report source is offered. The [3.2] “Geospatial Support” is not covered at all. Quads are not supported and since N-Triples and Turtle are the only supported RDF serializations only the criterion [3.3.2] from “Semantic Web Standards Support” is partially covered. The criterion category [3.4] “Framework Automation Level” is covered at a basic level since: (i) *query templates are not supported*, (ii) there is no common engine but a large monolithic entry point script, with multiple system relevant trivial and repetitive scripts using magic literals and (iii) it only supports *privileged dockerized operation* with Linux OS due to the use of the `perf` Python package. As a result setting up or updating benchmarks and adapting experiments requires extensive and meticulous modifications to all relevant system scripts in order to preserve integrity.

Finally, the [3.5] “Sustainability and Extensibility” criterion category is very weakly supported. The resource *is not sustainable* because: (i) the framework is not actively maintained and updated, as the paper was published in 2017 and there no commits for the past 8 years, (ii) the systems tested are deprecated e.g., Apache Jena v3.2.0 (released, Feb 2017), OrientDB 2.1.3 (released, October 2015), etc., (iii) the docker image fails to be built on a standard Linux Ubuntu 22.04.5 LTS, and (iv) the advent of the standard GQL for LPGs basically suggests that the supporting group’s research efforts should probably need to target a SPARQL-to-GQL converter.

## 8.2 IGUANA Benchmarking Framework

IGUANA aims to be a generic framework for benchmarking the read-write performance of triple stores. In its current status [26], it supports benchmarking SPARQL HTTP endpoints. In addition to querying it also allows for updates and stress testing endpoints by simulating users sending sets

<sup>22</sup>[https://docs.python.org/3/howto/perf\\_profiling.html](https://docs.python.org/3/howto/perf_profiling.html)



■ **Table 9** Semantic Web Standards Support [3.3].

| Criteria                           | 1. Quads as well as Triples | 2. Multiple RDF Serialization Support | 3. SPARQL Query Namespace Prefixes                 |
|------------------------------------|-----------------------------|---------------------------------------|----------------------------------------------------|
| GeoRDFBench Benchmarking Framework | Yes                         | Yes                                   | Yes, dynamic namespace prefix map merging          |
| LITMUS Benchmarking Framework      | No                          | Partial, N-Triples, Turtle            | No                                                 |
| IGUANA Benchmarking Framework      | Yes                         | Yes                                   | No                                                 |
| HOBBIT Benchmarking Platform       | No, user provided           | No, user provided                     | No, user provided                                  |
| Geographica 2 GeoSPARQL Benchmark  | Yes                         | Yes                                   | Adequate, non-dynamic namespace prefix map merging |

of queries independently of each other. It features a YAML file where all benchmark configuration is stored. This file contains the connections to RDF store endpoints, the description of tasks to run against endpoints, the metrics to measure and the destinations (CSV, RDF, endpoint) for report statistics. Its written in Java and is offered in two forms: (i) a Java jar file, and (ii) a native executable produced by compiling with GraalVM<sup>23</sup>. Despite its solid inception and implementation, focusing on SPARQL endpoint benchmarking bears a toll on the features IGUANA can support.

The criterion category [3.1] “Benchmarking Coverage” is adequately covered with some missing features: (i) only operates on HTTP endpoints, (ii) *repository final size is not reported* (iii) it *does not measure repository load times* (bulk loaders do not operate over HTTP), (iv) there is no way to differentiate query evaluation time from scan time, (v) there is *no result accuracy checking* measurement, (vi) query evaluation and/or scan errors are aggregated under “unknownException” statistics column along with HTTP related error codes, (vii) there can be no active error handling and no related policy to respond to them (e.g., ignore them and continue), (viii) scalability can borderline be tested with the available average size (232.5M triples) datasets. The [3.2] “Geospatial Support” is not covered at all. The criterion category [3.3] “Semantic Web Standards Support” is covered with the exception of criterion [3.3.3]. The criterion category [3.4] “Framework Automation Level” is covered at an adequate-high level because of its solid inception and implementation and the use of YAML specs. Missing features include: (i) query templates support is not generic but accommodates a specific scenario where query parameters are replaced by values returned from querying other live endpoints with related data, (ii) there is *no query inclusion/exclusion filters* for running query subsets, and (iii) provides native but not dockerized execution.

Finally, the [3.5] “Sustainability and Extensibility” criterion category is of medium strength. The resource seems to be borderline sustainable as it has active commits during the last 18 months and uses high quality libraries and techniques. It also seems to be quite easy to extend it by introducing new endpoints and workloads for testing, however there is no easy way to extend it in direct testing of GeoSPARQL/SPARQL engines through an RDF client library.

### 8.3 HOBBIT Benchmarking Platform

HOBBIT [70] is a very powerful but equally generic benchmarking framework. It *uses LD technologies, such as ontologies* to describe properties and parameters of benchmarks, systems and evaluation metrics and *uses OpenLink Virtuoso RDF store* as the platform’s persistent storage.

It has been used to benchmark non-LD applications and, among other things, allows it to run benchmarks throughout the LD life-cycle [73, 41]. This flexibility is achieved by containerizing the benchmarking components and using the RabbitMQ [48] message broker, which allows *byte array* “messages” to be exchanged between them. This *stripped from semantics representation* is exactly

<sup>23</sup><https://www.graalvm.org/>

what powers the general applicability of the HOBBIT framework, but *at the same time it carries a heavy price for the component developers who are obliged to take care all the application domain related critical tasks* without assistance from the platform APIs [30]. This functionality reusability at the container level also *creates dependencies of system containers' source code to modifications of the other ontologies*. For example, if new features are added to/removed from a benchmark's ontology, it not only triggers source code modification in the corresponding container but it also requires that all system adapter containers' source code needs to be aware of the new exposed benchmark features if one would like to test these systems against the updated version of the test. The *HOBBIT Java SDK* [72] provides some evidence <sup>24</sup> of the difficulties working with HOBBIT, such as: (i) demands a lot of reading, (ii) requires manual and error-prone work, (iii) does not allow to debug locally, etc. The HOBBIT Java SDK attempts to alleviate some of these problems.

In our view, this framework *targets a completely different user group* than the other frameworks and benchmarks in this work. The *average HOBBIT user* (according to designers' expectations) will be satisfied with reusing benchmarks and integrated systems in the form of containers provided by other *advanced HOBBIT users* and will simply modify the values of the models' exposed parameters. The premise is that there is such a **“group of the willing”** users that will be creating the relevant ontologies and containers for benchmarks and systems at the desired computer languages and updating containers when systems under test get newer versions. In reality the average HOBBIT user does not seem interested in embarking upon extending or modifying the relevant ontologies and containers of these benchmarks and systems and when they do so it is usually in the context of European funded sponsored challenges and mostly on the topic of Instance Matching or Link Discovery<sup>25</sup>.

The user of other benchmarking solutions to a lesser or greater (GeoRDFBench) extent is mainly interested in benchmarking RDF store performance in terms of data loading, data indexing, query accuracy and query execution times. They have expertise in Java or Python programming and want to find each RDF store's optimal configuration for each host's capabilities, try alternative data loading methods for different dataset sizes, experiment with different spatial indexing algorithms and accuracies offered by each geospatial RDF store, have some level of control over query execution in order to investigate exceptions either because of timeout expiration or the presence of malformed data or functionality compliance issues. As such, it is our view *that this framework is not directly competitive* with the other solutions and a direct comparison with quantitative criteria is neither possible nor useful.

From the criterion categories [3.1] “Benchmarking Coverage”, [3.2] “Geospatial Support” and [3.3] “Semantic Web Standards Support” the only criterion HOBBIT provides explicit support for is [3.1.4] “Statistics Reporting and Logging”. For every other criterion the user has to provide the implementation. “Framework Automation Level” category [3.4] is almost fully covered with a note that only dockerized execution is available which implies that cold cache executions are difficult to have. Finally, for the [3.5] “Sustainability and Extensibility” criterion category we verify that it is highly sustainable since it is actively maintained and supported with quality libraries, however, it is not easy to extend its ontology and APIs for directly supporting SPARQL/GeoSPARQL data loading and querying benchmarking.

<sup>24</sup>[https://github.com/hobbit-project/java-sdk-example/blob/master/SDK\\_vs\\_Standard\\_Way.pdf](https://github.com/hobbit-project/java-sdk-example/blob/master/SDK_vs_Standard_Way.pdf)

<sup>25</sup>[https://hobbit-project.github.io/challenge\\_overview.html](https://hobbit-project.github.io/challenge_overview.html)

## 8:40 Benchmarking with GeoRDFBench Framework

■ **Table 10** Framework Automation Level [3.4].

| Criteria                                  | 1. Setting Up New Benchmarks                                                                                                  | 2. Updating Existing Benchmarks                                                                                      | 3. Repeatable and Adaptable Experiment Execution                  | 4. Native and Dockerized Execution                                               |
|-------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------|----------------------------------------------------------------------------------|
| <b>GeoRDFBench Benchmarking Framework</b> | Yes, scalable GeoSPARQL data and query generator, template queries                                                            | Yes, declarative specs, reusability through class inheritance and script hierarchies, maven multi-module POM project | Yes, benchmarking environment specs, query filtering at execution | Yes, both on Linux and Windows OSs                                               |
| <b>LITMUS Benchmarking Framework</b>      | Basic, large monolithic entry point script, trivial repetitive scripts, magic literals, no common engine, ground queries only | Weak, updates require extensive and risky script modifications                                                       | Weak, adapting requires extensive and risky script modifications  | Adequate, privileged dockerized operation only, Linux only due to 'perf' package |
| <b>IGUANA Benchmarking Framework</b>      | Yes, non generic query templates                                                                                              | High, YAML specs, solid inception and implementation                                                                 | Adequate, YAML benchmark specs, no query filtering                | Adequate, native execution but with easy dockerization                           |
| <b>HOBBIT Benchmarking Platform</b>       | Yes                                                                                                                           | High, ontology based benchmark and system specs                                                                      | High                                                              | Adequate, dockerized execution only                                              |
| <b>Geographica 2 GeoSPARQL Benchmark</b>  | High, scalable GeoSPARQL data and query generator                                                                             | Partial, reusability through class inheritance and basic scripts                                                     | Weak, medium effort for updating benchmarks and systems           | Adequate, native execution only                                                  |

■ **Table 11** Sustainability and Extensibility [3.5].

| Criteria                                  | Sustainability and Extensibility                                                                                                                       |
|-------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>GeoRDFBench Benchmarking Framework</b> | Yes, actively maintained and supported, quality libraries, code base designed and implemented on OO principles                                         |
| <b>LITMUS Benchmarking Framework</b>      | Weak, last update 8 years ago, extending requires multiple risky script modifications                                                                  |
| <b>IGUANA Benchmarking Framework</b>      | Medium, active commits in the last 18 months, high quality libraries, easy to extend with new endpoints, difficult to extend for non HTTP benchmarking |
| <b>HOBBIT Benchmarking Platform</b>       | High, actively maintained and supported, quality libraries, difficult to extend ontology and APIs for SPARQL/GeoSPARQL cases                           |
| <b>Geographica 2 GeoSPARQL Benchmark</b>  | Medium, supported but not actively maintained, medium effort for updating benchmarks and systems                                                       |

## 8.4 Geographica 2 GeoSPARQL Benchmark

Geographica 2 benchmark is a set of well designed geospatial workloads and a set of RDF stores running against them following a specific set of steps and rules. However several areas were identified where much was left to expect for improvement. Abstraction and encapsulation [14, 11] were limited, leading to increased duplication of code and effort, especially when introducing new systems with a different RDF Java framework or with backward incompatible updated RDF Java framework version. Generalization-Specialization [14, 11] was limited to workload-specific queryset and experiment class hierarchies.

■ **Table 12** SLOC Comparison with Geographica 2.

| RDF Java Framework |         |       | OpenRDF Sesame |       | RDF4J     |       |             |       |              |       |             |       | Jena              |       |
|--------------------|---------|-------|----------------|-------|-----------|-------|-------------|-------|--------------|-------|-------------|-------|-------------------|-------|
| Module             | Runtime |       | Strabon SUT    |       | RDF4J SUT |       | GraphDB SUT |       | Virtuoso SUT |       | Stardog SUT |       | JenaGeoSPARQL SUT |       |
| CLOC Indicator     | Files   | Lines | Files          | Lines | Files     | Lines | Files       | Lines | Files        | Lines | Files       | Lines | Files             | Lines |
| Geographica 2      | 31      | 2249  | 2              | 333   | 3         | 835   | 3           | 584   | 4            | 511   | -           | -     | -                 | -     |
|                    | 79      | 10842 | 4              | 218   | 5         | 357   | 4           | 170   | 4            | 319   | 4           | 435   | 4                 | 133   |
| Increase (%)       | 155%    | 382%  | 100%           | -35%  | 67%       | -57%  | 33%         | -71%  | 0%           | -38%  |             |       |                   |       |

GeoRDFBench expands the scope of Universe of Discourse (UoD) [14] modeling to include all benchmark and experiment environment components, declaratively defines their specifications, decouples these from code by serializing them to disk, provides APIs and utilities to generate custom specifications and allows easy component creation from existing ones. Of paramount importance is the generalization of the repository and connection functionality of common RDF frameworks which allows significant reusability of code and easy integration of new systems with similar (in case of version upgrade) or different RDF frameworks.

As further quantitative evidence of the difference between Geographica 2 and GeoRDFBench a comparison is made between the two using the well established source code quality metric and effort predictor, the *Source Lines of Code (SLOC)* [56, 47]. More specifically, the utility **cloc** [18] measured the number of files and the *physical* SLOCs for the runtime and RDF modules for the

two systems. Table 12 presents the SLOC comparison between the two. The number of files and physical SLOCs of the GeoRDFBench runtime are **x2.54** and **x4.82** times the corresponding Geographica 2 ones. These numbers confirm the large scale of redesign and scope expansion that took place and which increased the opportunities for code reuse for RDF module implementation. This statement is consistent with the **reduction by half** of the number of physical SLOCs for the four common RDF modules: Strabon, RDF4J, GraphDB and Virtuoso. The newly introduced, Jena GeoSPARQL, required the least number of SLOCs because of its full GeoSPARQL compliance, while the newly introduced Stardog required the most effort since 50% of its SLOCs dealt with query translation to handle non-compliance to the GeoSPARQL standard. Also Stardog and Virtuoso required additional code to handle the server aspects of their architecture. RDF4Js SLOCs are relatively large because it encompasses the “dual” implementation of NativeStore with and without Lucene SAIL for spatial indexing.

With respect to compliance to the criteria categories in Section 3 we an adequate-high rating. The [3.1] “Benchmarking Coverage” is adequately covered with some missing features: (i) there is *no result accuracy checking* measurement, (ii) no query fine error handling. Scalability though is thoroughly provisioned with scalability workload and a scalable synthetic data and query generator. The [3.2] “Geospatial Support” is almost entirely covered with minor issues such as the limitation of query translation to specific systems and vocabularies and the lack of spatial predicate versions of querysets. The criterion category [3.3] “Semantic Web Standards Support” is covered almost entirely with the exception of non-dynamic namespace prefix map merging. The criterion category [3.4] “Framework Automation Level” is covered at an adequate level. Missing features include: (i) query templates support is not generic and not fully automated, (ii) limited reusability through class inheritance and basic script structure, (iii) provides native execution only, (iv) there is *no query inclusion/exclusion filters* for running query subsets, and (v) updating benchmark workloads and systems requires medium effort to complete. Finally, for the [3.5] “Sustainability and Extensibility” criterion category we verify that it is not very sustainable since it is supported but not actively maintained. Also, extending the solution requires medium to high effort to achieve.

## 9 Conclusions and Future Work

We presented the concepts, architecture and basic operation of the GeoRDFBench Framework, which aims to: (i) save the geospatial semantic benchmark researcher’s time and effort testing systems against benchmarks, (ii) minimize the margin for errors, (iii) increase reproducibility and results’ verification, while (iv) remaining extensible. Source code, running examples and instructions are provided in our repositories [31, 33] and site [32].

Future work will include: (i) implementing the remaining *Create, Update and Delete operations of the JSON Library endpoint* module, (ii) a *user interface module* paired with the endpoint which will assist with generating and modifying JSON specifications, (iii) support for the *Hadoop file system* and (iv) a *fourth Spark-based framework API*, so that Spark-based distributed GeoSPARQL solutions, such as **Strabo 2** [9] can be tested.

---

## References

- 1 Marcel R. Ackermann, Hannah Bast, Benedikt Maria Beckermann, Johannes Kalmbach, Patrick Neises, and Stefan Ollinger. The dblp knowledge graph and sparql endpoint. *Transactions on Graph Data and Knowledge*, 2(2):3:1–3:23, 2024. doi:10.4230/TGDK.2.2.3.
- 2 Altair. AnzoGraph DB web site, 2025. URL: <https://docs.cambridgesemantics.com/anzograph/v2.5/userdoc/Home.htm>.
- 3 Renzo Angles, Marcelo Arenas, Pablo Barceló, Peter A. Boncz, George H. L. Fletcher, Claudio Gutierrez, Tobias Lindaaker, Marcus Paradies, Stefan

- Plantikow, Juan F. Sequeda, Oskar van Rest, and Hannes Voigt. G-CORE: A core for future graph query languages. In Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein, editors, *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 1421–1432. ACM, 2018. doi:10.1145/3183713.3190654.
- 4 Renzo Angles, Harsh Thakkar, and Dominik Tomaszuk. Rdf and property graphs interoperability: Status and issues. *AMW*, 2369, 2019. URL: <https://ceur-ws.org/Vol-2369/paper01.pdf>.
- 5 Hannah Bast, Patrick Brosi, and Johannes Kalmbach. Efficient spatial joins on large geometry sets. In *Proceedings of the 33rd ACM International Conference on Advances in Geographic Information Systems*, pages 466–476, 2025. doi:10.1145/3748636.3762757.
- 6 Hannah Bast, Johannes Kalmbach, Christoph Ullinger, and Robin Textor-Falconi. Sparqlscope: A generic benchmark for comprehensive performance evaluation of sparql engines, May 2025. doi:10.5281/zenodo.15387492.
- 7 Robert Battle and Dave Kolas. Enabling the geospatial Semantic Web with Parliament and GeoSPARQL. *Semantic Web*, 3(4):355–370, 2012. doi:10.3233/SW-2012-0065.
- 8 Pierfrancesco Bellini and Paolo Nesi. Performance assessment of RDF graph databases for smart city services. *J. Vis. Lang. Comput.*, 45:24–38, 2018. doi:10.1016/j.jvlc.2018.03.002.
- 9 Dimitris Bilidas, Theofilos Ioannidis, Nikos Mamoulis, and Manolis Koubarakis. Strabo 2: Distributed management of massive geospatial rdf datasets. In *The Semantic Web–ISWC 2022: 21st International Semantic Web Conference, Virtual Event, October 23–27, 2022, Proceedings*, pages 411–427. Springer, 2022. doi:10.1007/978-3-031-19433-7\_24.
- 10 Christian Bizer and Andreas Schultz. The berlin sparql benchmark. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(2):1–24, 2009. doi:10.4018/IJSWIS.2009040101.
- 11 Grady Booch, Robert A Maksimchuk, Michael W Engle, Bobbi J Young, Jim Connallen, and Kelli A Houston. Object-oriented analysis and design with applications. *ACM SIGSOFT software engineering notes*, 33(5):29–29, 2008.
- 12 European Commission’s Joint Research Centre. INSPIRE Directive web site, 2007. URL: <https://inspire.ec.europa.eu/inspire-directive/2>.
- 13 Hirokazu Chiba, Ryota Yamanaka, and Shota Matsumoto. G2gml: Graph to graph mapping language for bridging rdf and property graphs. In *The Semantic Web–ISWC 2020: 19th International Semantic Web Conference, Athens, Greece, November 2–6, 2020, Proceedings, Part II 19*, pages 160–175. Springer, 2020. doi:10.1007/978-3-030-62466-8\_11.
- 14 Peter Coad, Edward Yourdon, et al. *Object-oriented analysis*, volume 2. Yourdon press New York, 1992.
- 15 European Commission. SWING Project web site, 2006. URL: <https://cordis.europa.eu/project/id/026514>.
- 16 European Commission. ExtremeEarth Project web site, 2019. URL: <http://earthanalytics.eu/>.
- 17 Lixi Conrads, Jens Lehmann, Muhammad Saleem, Mohamed Morsey, and Axel-Cyrille Ngonga Ngomo. Iguana: A generic framework for benchmarking the read-write performance of triple stores. In Claudia d’Amato, Miriam Fernandez, Valentina Tamma, Freddy Lecue, Philippe Cudré-Mauroux, Juan Sequeda, Christoph Lange, and Jeff Heflin, editors, *The Semantic Web – ISWC 2017*, pages 48–65, Cham, 2017. Springer International Publishing.
- 18 Albert Danial. cloc: v1.92 source code on Github, 2021. URL: <https://github.com/AlDanial/cloc>.
- 19 Alishiba Dsouza, Nicolas Tempelmeier, Ran Yu, Simon Gottschalk, and Elena Demidova. WorldKG: A world-scale geographic knowledge graph. In *CIKM ’21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, pages 4475–4484. ACM, 2021. doi:10.1145/3459637.3482023.
- 20 Orri Erling, Alex Averbuch, Josep Larriba-Pey, Hassan Chafi, Andrey Gubichev, Arnau Prat, Minh-Duc Pham, and Peter Boncz. The ldbc social network benchmark: Interactive workload. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 619–630, 2015.
- 21 Orri Erling and Ivan Mikhailov. Virtuoso: Rdf support in a native rdbms. In *Semantic web information management: a model-based perspective*, pages 501–519. Springer, 2009. doi:10.1007/978-3-642-04329-1\_21.
- 22 FasterXML. FasterXML Jackson source code on Github, 2024. URL: <https://github.com/FasterXML/jackson>.
- 23 Raphael A Finkel and Jon Louis Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4:1–9, 1974. doi:10.1007/BF00288933.
- 24 Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 international conference on management of data*, pages 1433–1445, 2018. doi:10.1145/3183713.3190657.
- 25 George Garbis, Kostis Kyzirakos, and Manolis Koubarakis. Geographica: A benchmark for geospatial rdf stores (long version). In *The Semantic Web–ISWC 2013: 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21–25, 2013, Proceedings, Part II 12*, pages 343–359. Springer, 2013. doi:10.1007/978-3-642-41338-4\_22.
- 26 DICE Group. Iguana source code on Github, 2024. URL: <https://github.com/dice-group/IGUANA>.
- 27 Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, 3(2-3):158–182, 2005. doi:10.1016/J.WEBSEM.2005.06.005.
- 28 Olaf Hartig. Foundations of rdf\* and sparql\*:(an alternative approach to statement-level metadata in rdf). In *AMW 2017 11th Alberto Mendelzon*



- International Workshop on Foundations of Data Management and the Web, Montevideo, Uruguay, June 7-9, 2017.*, volume 1912. Juan Reutter, Divesh Srivastava, 2017.
- 29 Joseph M. Hellerstein, Jeffrey F. Naughton, and Avi Pfeffer. Generalized search trees for database systems. In Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, editors, *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland*, pages 562–573. Morgan Kaufmann, 1995. URL: <http://www.vldb.org/conf/1995/P562.PDF>.
  - 30 HOBBIT. HOBBIT General API, 2017. URL: [https://hobbit-project.github.io/platform\\_api.html](https://hobbit-project.github.io/platform_api.html).
  - 31 Theofilos Ioannidis. GeoRDFBench Framework. Software, version 2.0.0, swbId: swb:1:dir:972135253c56ac42c3face1b01ab2ad360acea96 (visited on 2026-08-12). URL: <https://github.com/tioannid/geordfbench>, doi:10.4230/artifacts.27628.
  - 32 Theofilos Ioannidis. GeoRDFBench Framework Web Site. Service, version 2.0.0. (visited on 2026-08-12). URL: <https://geordfbench.di.uoa.gr/>, doi:10.4230/artifacts.27629.
  - 33 Theofilos Ioannidis. GeoRDFBench Framework Samples source code on Github, 2023. URL: [https://github.com/tioannid/geordfbench\\_samples](https://github.com/tioannid/geordfbench_samples).
  - 34 Theofilos Ioannidis. Geographica 2 Dataset doi, August 2024. doi:10.5281/zenodo.13283414.
  - 35 Theofilos Ioannidis. GeoRDFBench Framework Benchmark Results doi, May 2025. doi:10.5281/zenodo.15349539.
  - 36 Theofilos Ioannidis. GeoRDFBench Framework Benchmark Results on Github, 2025. URL: [https://github.com/tioannid/geordfbench\\_results](https://github.com/tioannid/geordfbench_results).
  - 37 Theofilos Ioannidis, George Garbis, Kostis Kyzirakos, Konstantina Bereta, and Manolis Koubarakis. Evaluating geospatial rdf stores using the benchmark geographica 2. *Journal on Data Semantics*, 10(3-4):189–228, 2021. doi:10.1007/S13740-021-00118-X.
  - 38 ISO/IEC. Information technology - Database languages - GQL, April 2024. URL: <https://www.iso.org/standard/76120.html>.
  - 39 Krzysztof Janowicz, Pascal Hitzler, Wenwen Li, Dean Rehberger, Mark Schildhauer, Rui Zhu, Cogan Shimizu, Colby K. Fisher, Ling Cai, Gengchen Mai, Joseph Zalewski, Lu Zhou, Shirley Stephen, Seila Gonzalez Estrecha, Bryce D. Mecum, Anna Lopez-Carr, Andrew Schroeder, Dave Smith, Dawn J. Wright, Sizhe Wang, Yuanyuan Tian, Zilong Liu, Meilin Shi, Anthony D'Onofrio, Zhining Gu, and Kitty Currier. Know, know where, knowwheregraph: A densely connected, cross-domain knowledge graph and geo-enrichment service stack for applications in environmental intelligence. *AI Mag.*, 43(1):30–39, 2022. doi:10.1609/aimag.v43i1.19120.
  - 40 Ernesto Jiménez-Ruiz, Tzanina Saveta, Ondrej Zamazal, Sven Hertling, Michael Roder, Irini Fundulaki, Axel Ngonga Ngomo, Mohamed Sherif, Amina Annane, Zohra Bellahsene, et al. Introducing the hobbit platform into the ontology alignment evaluation campaign. In *13th International Workshop on Ontology Matching (OM)*, volume 2288, pages 49–60, 2018. URL: [https://ceur-ws.org/Vol-2288/om2018\\_LTPaper5.pdf](https://ceur-ws.org/Vol-2288/om2018_LTPaper5.pdf).
  - 41 Milos Jovanovik. HOBBIT Examples source code on Github, 2017. URL: <https://github.com/hobbit-project/SpatialBenchmark>.
  - 42 Milos Jovanovik, Timo Homburg, and Mirko Spasić. A geosparql compliance benchmark. *ISPRS International Journal of Geo-Information*, 10(7):487, 2021. doi:10.3390/IJGI10070487.
  - 43 Nikolaos Karalis, Georgios M. Mandilaras, and Manolis Koubarakis. Extending the YAGO2 knowledge graph with precise geospatial knowledge. In Chiara Ghidini, Olaf Hartig, Maria Maleshkova, Vojtech Svátek, Isabel F. Cruz, Aidan Hogan, Jie Song, Maxime Lefrançois, and Fabien Gandon, editors, *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part II*, volume 11779 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2019. doi:10.1007/978-3-030-30796-7\_12.
  - 44 Charalampos Kostopoulos, Giannis Mouchakis, Antonis Troumpoukis, Nefeli Prokopaki-Kostopoulou, Angelos Charalambidis, and Stasinou Konstantopoulos. Kobe: Cloud-native open benchmarking engine for federated query processors. In *The Semantic Web: 18th International Conference, ESWC 2021, Virtual Event, June 6–10, 2021, Proceedings 18*, pages 664–679. Springer, 2021. doi:10.1007/978-3-030-77385-4\_40.
  - 45 Manolis Koubarakis, Konstantina Bereta, Dimitris Bilidas, Konstantinos Giannousis, Theofilos Ioannidis, Despina-Athanasia Pantazi, George Stamoulis, Seif Haridi, Vladimir Vlassov, Lorenzo Bruzzone, et al. From copernicus big data to extreme earth analytics. *Open Proceedings*, pages 690–693, 2019.
  - 46 Kostis Kyzirakos, Manos Karpathiotakis, and Manolis Koubarakis. Strabon: A semantic geospatial dbms. In *The Semantic Web-ISWC 2012: 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I 11*, pages 295–311. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-35176-1\_19.
  - 47 Davy Landman, Alexander Serebrenik, and Jürgen Vinju. Empirical analysis of the relationship between cc and sloc in a large corpus of java methods. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 221–230, 2014. doi:10.1109/ICSME.2014.44.
  - 48 Rabbit Technologies Ltd. RabbitMQ Broker web site, 2007. URL: <https://www.rabbitmq.com/>.
  - 49 Manolis Koubarakis, editor. *Geospatial data science: a hands-on approach based on geospatial technologies*. ACM Books, 2023.
  - 50 Mohamed Morsey, Jens Lehmann, Sören Auer, and Axel-Cyrille Ngonga Ngomo. Dbpedia sparql benchmark—performance assessment with real queries on real data. In *The Semantic Web-ISWC 2011: 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I 10*, pages 454–469. Springer, 2011. doi:10.1007/978-3-642-25073-6\_29.

- 51 Thomas Mueller. H2 Database) web site, 2005. URL: <https://h2database.com/>.
- 52 MvnRepository. Maven Central Repository web site, 2024. URL: <https://mvnrepository.com/repos/central>.
- 53 neo4j. Neo4j Spatial source code on Github, 2025. URL: <https://github.com/neo4j-contrib/spatial>.
- 54 neosemantics. neosemantics (n10s) source code on Github, 2025. URL: <https://github.com/neo4j-labs/neosemantics>.
- 55 Axel-Cyrille Ngonga Ngomo, Sören Auer, Jens Lehmann, and Amrapali Zaveri. Introduction to linked data and its lifecycle on the web. *Reasoning Web. Reasoning on the Web in the Big Data Era: 10th International Summer School 2014, Athens, Greece, September 8-13, 2014. Proceedings 10*, pages 1–99, 2014. doi:10.1007/978-3-319-10587-1\_1.
- 56 Vu Nguyen, Sophia Deeds-Rubin, Thomas Tan, and Barry Boehm. A sloc counting standard. In *Cocomo ii forum*, volume 2007, pages 1–16. Cite-seer, 2007.
- 57 Ontotext. GraphDB, 2024. URL: <https://graphdb.ontotext.com/>.
- 58 Ontotext. GraphDB geosparql, 2024. URL: <https://graphdb.ontotext.com/documentation/10.7/geosparql-support.html#geosparql-support>.
- 59 Ontotext. GraphDB ImportRDF geosparql, 2024. URL: <https://graphdb.ontotext.com/documentation/10.8/command-line-tools.html#importrdf>.
- 60 Oracle. Spatial And Graph web site, 2019. URL: <https://docs.oracle.com/en/database/oracle/oracle-database/19/spatial-and-graph.html>.
- 61 Taha Osman and Gregory Albiston. Geosparql-jena: Implementation and benchmarking of a geosparql graphstore. In *23rd European Conference on Knowledge Management Vol 2*. Academic Conferences and publishing limited, 2022.
- 62 Alisdair Owens, Andy Seaborne, Nick Gibbins, and m.c Schraefel. Clustered tdb: A clustered triple store for jena, November 2008.
- 63 Kostas Patroumpas, Giorgos Giannopoulos, and Spiros Athanasiou. Towards geospatial semantic data management: strengths, weaknesses, and challenges ahead. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 301–310, 2014. doi:10.1145/2666310.2666410.
- 64 Matthew Perry, Ana Estrada, Souripriya Das, and Jayanta Banerjee. Developing geosparql applications with oracle spatial and graph. In *SSN-TC/OrdRing@ ISWC*, pages 57–61, 2015. URL: <https://ceur-ws.org/Vol-1488/paper-06.pdf>.
- 65 Matthew Perry and John Herring. OGC GeoSPARQL - A Geographic Query Language for RDF Data. OGC Implementation Standard OGC 11-052r4, Open Geospatial Consortium, September 2012. URL: <http://www.opengis.net/doc/IS/geosparql/1.0>.
- 66 Suprio Ray, Bogdan Simion, and Angela Demke Brown. Jackpine: A benchmark to evaluate spatial database performance. In *2011 IEEE 27th International Conference on Data Engineering*, pages 1139–1150. IEEE, 2011. doi:10.1109/ICDE.2011.5767929.
- 67 RDF4J. RDF4J, 2024. URL: <https://rdf4j.org/>.
- 68 RDF4J. RDF4J lucene geosparql, 2024. URL: <https://rdf4j.org/documentation/programming/geosparql/>.
- 69 RDF4J. RDF4J NativeStore API Doc, 2024. URL: <https://rdf4j.org/javadoc/4.3.16/org/eclipse/rdf4j/sail/nativerdf/NativeStore.html>.
- 70 Michael Röder, Denis Kuchelev, and Axel-Cyrille Ngonga Ngomo. HOBBIT: A platform for benchmarking big linked data. *Data Sci.*, 3(1):15–35, 2020. doi:10.3233/ds-190021.
- 71 Marko A. Rodriguez. The gremlin graph traversal machine and language (invited talk). In *Proceedings of the 15th Symposium on Database Programming Languages*. ACM, October 2015. doi:10.1145/2815072.2815073.
- 72 Michael Röder. HOBBIT Java SDK web site, 2018. URL: <https://github.com/hobbit-project/java-sdk>.
- 73 Michael Röder. HOBBIT Examples source code on Github, 2024. URL: <https://github.com/hobbit-project/hobbit.examples>.
- 74 Muhammad Saleem, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. Feasible: A feature-based sparql benchmark generation framework. In *The Semantic Web-ISWC 2015: 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I 14*, pages 52–69. Springer, 2015. doi:10.1007/978-3-319-25007-6\_4.
- 75 Stardog. Stardog, 2024. URL: <https://www.stardog.com/>.
- 76 Harsh Thakkar. litmus-docker source code on Github, 2017. URL: <https://github.com/LITMUS-Benchmark-Suite/litmus-docker>.
- 77 Harsh Thakkar. Towards an open extensible framework for empirical benchmarking of data management solutions: Litmus. In *The Semantic Web: 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28-June 1, 2017, Proceedings, Part II 14*, pages 256–266. Springer, 2017. doi:10.1007/978-3-319-58451-5\_20.
- 78 Antonis Troumpoukis, Stasinos Konstantopoulos, Giannis Mouchakis, Nefeli Prokopaki-Kostopoulou, Claudia Paris, Lorenzo Bruzzone, Despina Athanasia Pantazi, and Manolis Koubarakis. Geofedbench: A benchmark for federated geosparql query processors. In *ISWC (Demos/Industry)*, pages 228–232, 2020. URL: <https://ceur-ws.org/Vol-2721/paper558.pdf>.
- 79 Oskar van Rest, Sungpack Hong, Jinha Kim, Xuming Meng, and Hassan Chafi. PGQL: a property graph query language. In Peter A. Boncz and Josep Lluís Larriba-Pey, editors, *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems, Redwood Shores, CA, USA, June 24 - 24, 2016*, page 7. ACM, 2016. doi:10.1145/2960414.2960421.
- 80 W3C. SPARQL 1.1 Query Language web site, 2013. URL: <https://www.w3.org/TR/sparql11-query/>.
- 81 Wikipedia. Geohash wiki page, 2008. URL: <https://en.wikipedia.org/wiki/Geohash>.



## A Scalability Workload Serialized JSON Specifications

This appendix presents the detailed form of the three most important JSON specifications for the Scalability 10K workload as they were generated by the JSON generator API code samples in Section 6. Listings 11, 12, 13 show the serialized representations of the specifications for the Scalability 10K complex dataset, the spatial functions variant of the Scalability queryset and the Scalability experiment execution respectively.

**Listing 11** Serialized Dataset Specification.

```
{ "classname" : "gr.uoa.di.rdf.Geographica3.runtime.datasets.complex.impl.GeographicaDS",
  "name" : "scalability_10K",
  "relativeBaseDir" : "Scalability/10K",
  "simpleGeospatialDataSetList" : [ {
    "name" : "scalability_10K",
    "relativeBaseDir" : "Scalability/10K",
    "dataFile" : "scalability_10K.nt",
    "rdfFormat" : "N-TRIPLES",
    "mapUsefulNamespacePrefixes" : {
      "geo" : "<http://www.opengis.net/ont/geosparql#>",
      "rdf" : "<http://www.w3.org/1999/02/22-rdf-syntax-ns#>",
      "owl" : "<http://www.w3.org/2002/07/owl#>",
      "geof" : "<http://www.opengis.net/def/function/geosparql/>",
      "lgo" : "<http://data.linkedeodata.eu/ontology#>",
      "xsd" : "<http://www.w3.org/2001/XMLSchema#>",
      "rdfs" : "<http://www.w3.org/2000/01/rdf-schema#>",
      "geo-sf" : "<http://www.opengis.net/ont/sf#>"
    },
    "mapAsWKT" : {
      "scalabilityAsWKT" : "<http://www.opengis.net/ont/geosparql#asWKT>"
    },
    "mapHasGeometry" : {
      "scalabilityHasGeometry" : "<http://www.opengis.net/ont/geosparql#hasGeometry>"
    }
  } ],
  "mapDataSetContexts" : {
    "scalability_10K" : ""
  },
  "n" : 0 }
```

**Listing 12** Serialized Queryset Specification.

```
{ "classname" : "gr.uoa.di.rdf.Geographica3.runtime.querysets.complex.impl.StaticTempParamQS",
  "name" : "scalabilityFunc",
  "relativeBaseDir" : "",
  "hasPredicateQueriesAlso" : false,
  "mapQueries" : {
    "0" : {
      "label" : "SC1_Geometries_Intersects_GivenPolygon",
      "text" : "SELECT ?s1 ?o1 WHERE { \n ?s1 geo:asWKT ?o1 . \n FILTER(geof:FUNCTION(?o1, GIVEN_SPATIAL_LITERAL)). \n} \n",
      "usePredicate" : false,
      "expectedResults" : -1
    },
    "1" : {
      "label" : "SC2_Intensive_Geometries_Intersect_Geometries",
      "text" : "SELECT ?s1 ?s2 \nWHERE { \n ?s1 geo:hasGeometry [ geo:asWKT ?o1 ] ;\n lgo:has_code\n \"1001\"^^xsd:integer . \n ?s2 geo:hasGeometry [ geo:asWKT ?o2 ] ;\n lgo:has_code ?code2 . \n\n FILTER(?code2>5000 && ?code2<6000 && ?code2 != 5260) .\n FILTER(geof:FUNCTION(?o1, ?o2)). \n} \n",
      "usePredicate" : false,
      "expectedResults" : -1
    },
    "2" : {
      "label" : "SC3_Relaxed_Geometries_Intersect_Geometries",
      "text" : "SELECT ?s1 ?s2\nWHERE { \n ?s1 geo:hasGeometry [ geo:asWKT ?o1 ] ;\n lgo:has_code\n \"1001\"^^xsd:integer . \n ?s2 geo:hasGeometry [ geo:asWKT ?o2 ] ;\n lgo:has_code ?code2 .\n\n FILTER(?code2\n IN (5622, 5601, 5641, 5621, 5661)) .\n FILTER(geof:FUNCTION(?o1, ?o2)). \n} \n",
      "usePredicate" : false,
      "expectedResults" : -1
    }
  },
  "mapUsefulNamespacePrefixes" : { } ,
```

## 8:46 Benchmarking with GeoRDFBench Framework

```
"mapTemplateParams" : {  
  "GIVEN_SPATIAL_LITERAL" : "\"POLYGON((23.708496093749996 37.95719224376526,22.906494140625  
    40.659805938378526,11.524658203125002 48.16425348854739,-0.1181030273437499  
    51.49506473014367,-3.2189941406250004 55.92766341247031,-5.940856933593749  
    54.59116279530599,-3.1668090820312504 51.47967237816337,23.708496093749996  
    37.95719224376526))\"<http://www.opengis.net/ont/geosparql#wktLiteral>",  
  "FUNCTION" : "sfIntersects"  
},  
"mapGraphPrefixes" : { } }
```

**Listing 13** Serialized Execution Specification.

```
{ "classname" : "gr.uoa.di.rdf.Geographica3.runtime.executionspecs.impl.SimpleES",  
  "execTypeReps" : {  
    "COLD" : 3,  
    "WARM" : 3  
  },  
  "maxDurationSecsPerQueryRep" : 86400,  
  "maxDurationSecs" : 604800,  
  "action" : "RUN",  
  "avgFunc" : "QUERY_MEDIAN",  
  "onColdFailure" : "SKIP_REMAINING_ALL_QUERY_EXECUTIONS",  
  "clearCacheDelaySecs" : 5000 }
```

## B GeoRDFBench Execution Script and Logs

This appendix details the differences of the experiment execution script between RDF4J vs RDF4J+ and GraphDB and GraphDB+ variants as well as, part of the experiment Scalability 10K repository generation for RDF4JSUT+ and part of the execution log of Virtuoso against the Scalability 1M workload.

**Listing 14** diff(RDF4J, RDF4J+) Experiment execution scripts.

```
$ diff -U 1 runRDF4J_Scal10K_1M.sh runRDF4J_Lucene_Scal10K_1M.sh  
...  
@@ -4,4 +4,5 @@  
source prepareRunEnvironment.sh nuc8i7beh RDF4JSUT "CreateRepo_Scalability10K_1M_RDF4J"  
+ export EnableLuceneSail=true  
./printRunEnvironment.sh >> /data/envvars_rdf4j.log  
...
```

**Listing 15** diff(GraphDB, GraphDB+) Experiment execution scripts.

```
$ diff -U 1 runGraphDB_Scal10K_1M.sh runGraphDB_GeoSPARQL_Scal10K_1M.sh  
...  
@@ -3,2 +3,5 @@  
source prepareRunEnvironment.sh nuc8i7beh GraphDBSUT "CreateRepo_Scalability10K_1M_GraphDB"  
+ export EnableGeoSPARQLPlugin=true  
+ # export IndexingAlgorithm = quad  
+ # export IndexingPrecision = 11  
./printRunEnvironment.sh >> /data/envvars_graphdb.log  
...
```

**Listing 16** RDF4JSUT+ repository generation log.

```
...  
Running script with syntax:  
./createAllRDF4JRepos.sh false /data/Geographica2_Datasets /data/RDF4J_4.3.15_LuceneRepos/server -Xmx24g true  
192.168.1.44 3333 /data/Geographica2_Datasets/Scalability/scalabilityDSGen.sh  
/data/Geographica2_Datasets/Scalability/scalability500MRefDS.nt.gz  
Script start time: Mon 28 Apr 2025 04:21:18 pm EEST  
/data/RDF4J_4.3.15_LuceneRepos/server dir does not exist. Creating it now ...  
Checking/Creating scalability 10K dataset ... Scalability 10K dataset already exists  
Generating scalability 10K repository ...  
./createRDF4JRepo.sh /data/RDF4J_4.3.15_LuceneRepos/server scalability_10K false "spoc,posc" N-TRIPLES  
/data/Geographica2_Datasets/Scalability/10K -Xmx24g true "http://www.opengis.net/ont/geosparql#asWKT"
```

```

192.168.1.44 3333
CREATE_REPO_ARGS = createman "/data/RDF4J_4.3.15_LuceneRepos/server" "scalability_10K" "FALSE" "true"
                    "spoc,posc" "http://www.opengis.net/ont/geosparql#asWKT"
LOAD_REPO_ARGS = dirloadman "/data/RDF4J_4.3.15_LuceneRepos/server" "scalability_10K" "N-TRIPLES"
                    "/data/Geographica2_Datasets/Scalability/10K" true
0 [main] INFO RDF4JSystem - No LocalRepositoryManager instance present, creating a new one.
68 [main] INFO RDF4JSystem - Creating NativeStore base sail with spoc,posc indexes and forceSync = false
71 [main] INFO RDF4JSystem - Adding Lucene sail on top of NativeStore
73 [main] INFO RDF4JSystem - Lucene sail will spatially index properties:
    http://www.opengis.net/ont/geosparql#asWKT
216 [main] INFO RDF4JSystem - Creating new repository object for repo id = scalability_10K
775 [main] INFO RDF4JSystem - Initializing new repository object for repo id = scalability_10K
835 [main] INFO RepoUtil - RDF4J created with manager lucene repo
    "/data/RDF4J_4.3.15_LuceneRepos/server/repositories/scalability_10K" in 74 msecs
835 [main] INFO RDF4JSystem - Closing connection...
[main] INFO RDF4JSystem - Repository closed.
1 [main] INFO RDF4JSystem - Loading file scalability_10K.nt ...
2500 [main] INFO RDF4JSystem - Finished loading file scalability_10K.nt in 2492 msecs
2506 [main] INFO RepoUtil - RDF4J loaded with manager all files from
    "/data/Geographica2_Datasets/Scalability/10K" to repo
    "/data/RDF4J_4.3.15_LuceneRepos/server/repositories/scalability_10K" in 2599 msecs
2506 [main] INFO RDF4JSystem - Closing connection...
2508 [main] INFO RDF4JSystem - Repository closed.
RDF4J repository "/data/RDF4J_4.3.15_LuceneRepos/server/repositories/scalability_10K" has size: 7MB

```

■ Listing 17 Virtuoso run log start section for Scalability 1M.

```

...
0 [main] INFO RunVirtuosoExperimentWorkload - args[0] = -surl
1 [main] INFO RunVirtuosoExperimentWorkload - args[1] = http://localhost:1111
1 [main] INFO RunVirtuosoExperimentWorkload - args[2] = -susr
1 [main] INFO RunVirtuosoExperimentWorkload - args[3] = dba
1 [main] INFO RunVirtuosoExperimentWorkload - args[4] = -spwd
1 [main] INFO RunVirtuosoExperimentWorkload - args[5] = dba
1 [main] INFO RunVirtuosoExperimentWorkload - args[6] = -rbd
2 [main] INFO RunVirtuosoExperimentWorkload - args[7] = virtuoso-opensource-7.2.14/repos
2 [main] INFO RunVirtuosoExperimentWorkload - args[8] = -expdesc
2 [main] INFO RunVirtuosoExperimentWorkload - args[9] = 2025-04-27_VirtuosoSUT_RunWL_Scal1M
2 [main] INFO RunVirtuosoExperimentWorkload - args[10] = -wl
2 [main] INFO RunVirtuosoExperimentWorkload - args[11] =
    /data/geordfbench/json_defs/workloads/scalabilityFunc1M_WLoriginal.json
2 [main] INFO RunVirtuosoExperimentWorkload - args[12] = -h
2 [main] INFO RunVirtuosoExperimentWorkload - args[13] =
    /data/geordfbench/json_defs/hosts/nuc8i7behHOSToriginal.json
2 [main] INFO RunVirtuosoExperimentWorkload - args[14] = -rs
2 [main] INFO RunVirtuosoExperimentWorkload - args[15] =
    /data/geordfbench/json_defs/reportspecs/simplereportspec_original.json
2 [main] INFO RunVirtuosoExperimentWorkload - args[16] = -rpsr
2 [main] INFO RunVirtuosoExperimentWorkload - args[17] =
    /data/geordfbench/json_defs/reportsources/nuc8i7behHOSToriginal.json
5 [main] INFO RunVirtuosoExperimentWorkload - |==> Experiment related options
9 [main] INFO RunVirtuosoExperimentWorkload - Experiment description: 2025-04-27_VirtuosoSUT_RunWL_Scal1M
9 [main] INFO RunVirtuosoExperimentWorkload - |==> Workload, Host, Report related options
9 [main] INFO RunVirtuosoExperimentWorkload - Workload specs configuration JSON file:
    /data/geordfbench/json_defs/workloads/scalabilityFunc1M_WLoriginal.json
10 [main] INFO RunVirtuosoExperimentWorkload - List of queries to include in the run: all
10 [main] INFO RunVirtuosoExperimentWorkload - Host configuration JSON file:
    /data/geordfbench/json_defs/hosts/nuc8i7behHOSToriginal.json
10 [main] INFO RunVirtuosoExperimentWorkload - Report specs configuration JSON file:
    /data/geordfbench/json_defs/reportspecs/simplereportspec_original.json
11 [main] INFO RunVirtuosoExperimentWorkload - Report source specs configuration JSON file:
    /data/geordfbench/json_defs/reportsources/nuc8i7behHOSToriginal.json
11 [main] INFO RunVirtuosoExperimentWorkload - |==> Repository/Store options
11 [main] INFO RunVirtuosoExperimentWorkload - Virtuoso Server endpoint URL: http://localhost:1111
11 [main] INFO RunVirtuosoExperimentWorkload - Virtuoso server username: dba
12 [main] INFO RunVirtuosoExperimentWorkload - Virtuoso server password: dba
12 [main] INFO RunVirtuosoExperimentWorkload - |==> System options
12 [main] INFO RunVirtuosoExperimentWorkload - BaseDir: virtuoso-opensource-7.2.14/repos
284 [main] INFO RunVirtuosoExperimentWorkload - {SimpleGeospatialWL: ScalabilityFunc, ,
    {GeographicaDS: scalability_1M, Scalability/1M,
    {GenericGeospatialSimpleDS: scalability_1M, N-TRIPLES}}
    {StaticTempParamQS: scalabilityFunc, false, SimpleES{ COLD=3, WARM=3, action=RUN, maxduration=604800 secs,
        repmaxduration=86400 secs, func=QUERY_MEDIAN }}}
590 [main] INFO RDF4JBasedSUT - Reading VirtuosoSUT properties from file :
    jar:file:/data/geordfbench/VirtuosoSUT/target/VirtuosoSUT-2.0.0-SNAPSHOT.jar!/virtuoso.properties
603 [main] INFO VirtuosoSUT - Initializing..

```

## 8:48 Benchmarking with GeoRDFBench Framework

```
603 [main] INFO VirtuosoSUT - Starting Virtuoso server...
15634 [main] INFO VirtuosoSystem - Uninitialized VirtuosoRepository created with query timeout = 0
15634 [main] INFO VirtuosoSystem - Initialized VirtuosoRepository has query timeout = 86400
...
15713 [main] INFO ExperimentWorkload - VirtuosoSystem-dependent translation of the queryset scalabilityFunc
15713 [main] INFO ExperimentWorkload - VirtuosoSystem-dependent namespace prefixes merged with the prefixes of
      queryset scalabilityFunc
15714 [main] INFO VirtuosoSUT - Closing..
15715 [main] INFO VirtuosoSystem - Closing connection...
15715 [main] INFO VirtuosoSystem - Repository closed.
20728 [main] INFO VirtuosoSUT - Stopping Virtuoso server...
```

### ■ Listing 18 Virtuoso run log Scalability 10K results.

```
289581 [main] INFO PostgreSQLRepSrc - Deferred mode for PostgreSQLRepSrc was enabled. 18 records were flushed
289582 [main] INFO GenericExperimentResultsCollector - Export statistics in
      "/data/Results_Store/VirtuosoSUT/2025-04-27_VirtuosoSUT_RunWL_Scal10K/
      Scalability/10K/VirtuosoSUT-ExperimentWorkload"
289740 [main] INFO GenericExperimentResultsCollector - Created non existing directory
289760 [main] INFO GenericExperimentResultsCollector - Statistics printed:
      ./00-SC1_Geometries_Intersects_GivenPolygon-cold
289772 [main] INFO GenericExperimentResultsCollector - Statistics printed:
      ./00-SC1_Geometries_Intersects_GivenPolygon-cold-long
...
289777 [main] INFO GenericExperimentResultsCollector - Statistics printed:
      ./02-SC3_Relaxed_Geometries_Intersect_Geometries-warm
289777 [main] INFO GenericExperimentResultsCollector - Statistics printed:
      ./02-SC3_Relaxed_Geometries_Intersect_Geometries-warm-long
289778 [main] INFO GenericExperimentResultsCollector - Cache COLD
289778 [main] INFO GenericExperimentResultsCollector - Query 0
289779 [main] INFO GenericExperimentResultsCollector - Rep 0 <COMPLETED-NONE> 634086428 + 114910498 =
      748996926 nsecs, 554 results, 0 scan errors
289779 [main] INFO GenericExperimentResultsCollector - Rep 1 <COMPLETED-NONE> 579085856 + 123844201 =
      702930057 nsecs, 554 results, 0 scan errors
289779 [main] INFO GenericExperimentResultsCollector - Rep 2 <COMPLETED-NONE> 444039862 + 123960680 =
      568000542 nsecs, 554 results, 0 scan errors
289779 [main] INFO GenericExperimentResultsCollector - Query 1
...
289780 [main] INFO GenericExperimentResultsCollector - Query 2
...
289780 [main] INFO GenericExperimentResultsCollector - Cache WARM
289780 [main] INFO GenericExperimentResultsCollector - Query 0
...
289780 [main] INFO GenericExperimentResultsCollector - Query 1
289780 [main] INFO GenericExperimentResultsCollector - Rep 0 <COMPLETED-NONE> 158021881 + 383204 =
      158405085 nsecs, 2 results, 0 scan errors
289780 [main] INFO GenericExperimentResultsCollector - Rep 1 <COMPLETED-NONE> 157411100 + 355432 =
      157766532 nsecs, 2 results, 0 scan errors
289780 [main] INFO GenericExperimentResultsCollector - Rep 2 <COMPLETED-NONE> 157869590 + 304380 =
      158173970 nsecs, 2 results, 0 scan errors
289780 [main] INFO GenericExperimentResultsCollector - Query 2
...
289781 [main] INFO RunVirtuosoExperimentWorkload - End ScalabilityFunc
Start time = Sun 27 Apr 2025 07:45:49 pm EEST
End time = Sun 27 Apr 2025 07:50:39 pm EEST
```

## C GeoRDFBench Experiment Common Execution Logic

This appendix presents the execution logic of the experiment executor component of the GeoRDF-Bench runtime which is shared by all implemented RDF modules.

■ **Algorithm 1** Experiment execution loop.

---

```

procedure RUN
  querySetIterations  $\leftarrow$  0                                ▷ queryset iteration counter
  problematicQueries.clear()                                ▷ queries that raised exception
  while max experiment duration has not been exceeded do
    queryset initialization                                    ▷ dynamic or file-driven
    for all COLD executions do                                ▷ COLD runs
      for all queries in filtered queryset do
        for all repetitions do
          init SUT after clearing caches
          coldResult  $\leftarrow$  sut.runTimedQuery()
          add coldResult to resCollector,reportSink
          close SUT
          if coldResult raised exception then
            skip remaining query repetitions
          end if
        end for
      end for
    end for
    for all WARM executions do                                ▷ WARM runs
      for all queries in filtered queryset do
        if COLD run raised exception then
          warmResult  $\leftarrow$  coldResult
          add warmResult to resCollector,reportSink
          skip to next query
        end if
        init SUT after clearing caches
        pseudorunResult  $\leftarrow$  sut.runTimedQuery()
        for all repetitions do
          warmResult  $\leftarrow$  sut.runTimedQuery()
          add warmResult to resCollector,reportSink
          close SUT
          if warmResult raised exception then
            skip remaining query repetitions
          end if
        end for
      end for
    end for
  end for

```

---

## 8:50 Benchmarking with GeoRDFBench Framework

### Algorithm 1 Experiment execution loop (cont.).

---

```

for all COLDCONTINUOUS executions do                                ▷ CONT runs
  if querySetIterations = 0 then                                     ▷ init system once
    init SUT after clearing caches
  end if
  for all queries in filtered queryset do
    if query is contained in problematicQueries then
      skip to next query
    end if
    coldContResult ← sut.runTimedQuery()
    add coldContResult to resCollector,reportSink
    if coldContResult raised exception then
      add query to problematicQueries
    end if
  end for
end while
close SUT and
flush report sink deferred records
resCollector.exportStatistics()
end procedure

```

---

## D Docker Examples Feature Matrix

This appendix presents the features exhibited in each one of the docker examples found in the GeoRDFBench web site [32].

Table 13 Docker Examples Feature Matrix.

| Example    | Name           |                                    | Scalability 10K Workload With RDF4J | RDF4J, GraphDB and JenaGeoSPARQL against Scalability-{10K, 100K, 1M} Workloads | Strabon and Virtuoso against The Synthetic Workload | RDF4J, GraphDB and JenaGeoSPARQL against LUBM(1, 0) Workload |
|------------|----------------|------------------------------------|-------------------------------------|--------------------------------------------------------------------------------|-----------------------------------------------------|--------------------------------------------------------------|
|            | Type           |                                    | Docker                              | Docker                                                                         | Docker                                              | Docker                                                       |
| Example    | Host OS        |                                    | Linux                               | Windows 10                                                                     | Linux                                               | Windows 10                                                   |
|            | Docker         | <i>Image</i>                       | Manual Build                        | Pre-build from Github Registry                                                 | Pre-build from Github Registry                      | Pre-build from Github Registry                               |
|            |                | <i>Simulated Host</i>              | NUC8i7BEH                           | NUC8i7BEH                                                                      | NUC8i7BEH                                           | NUC8i7BEH                                                    |
| Experiment | Startup Script |                                    | Automatic with Container Start      | Manual Script Start                                                            | Manual Script Start                                 | Manual Script Start                                          |
| Benchmark  | Name           |                                    | Geographica 2                       | Geographica 2                                                                  | Geographica 2                                       | LUBM                                                         |
|            |                | <i>Representation</i>              | Compact                             | Compact                                                                        | Detailed                                            | Compact                                                      |
|            |                | <i>Name</i>                        | Scalability                         | Scalability                                                                    | Synthetic                                           | LUBM(1, 0)                                                   |
|            | Workload       | <i>Dataset 1</i>                   | 10K                                 | 10K GOLD STANDARD                                                              | Scaling Factor N=512                                | Scaling Factor N=1, GOLD STANDARD                            |
|            |                | <i>Dataset 2</i>                   |                                     | 100K                                                                           |                                                     |                                                              |
| RDF Stores |                | <i>Dataset 3</i>                   |                                     |                                                                                |                                                     |                                                              |
|            | 1              |                                    | RDF4J                               | RDF4J                                                                          | Strabon                                             | RDF4J                                                        |
|            | 2              |                                    |                                     | GraphDB                                                                        | Virtuoso                                            | GraphDB                                                      |
| Features   | 3              |                                    |                                     | JenaGeoSPARQL                                                                  |                                                     | JenaGeoSPARQL                                                |
|            |                | <i>Result Accuracy Check</i>       |                                     | Yes                                                                            | Yes                                                 | Yes                                                          |
|            |                | <i>Query Inclusion Filter</i>      |                                     | Yes (Queryset)                                                                 | Yes (Execution)                                     | Yes                                                          |
|            |                | <i>Specification Customization</i> |                                     |                                                                                |                                                     | Yes                                                          |
|            |                | <i>JSON Generator API</i>          |                                     |                                                                                |                                                     | Yes                                                          |
|            |                | <i>Auto Query Prefix Header</i>    | Yes                                 | Yes                                                                            | Yes                                                 | Yes                                                          |
|            |                | <i>System Query Translation</i>    |                                     |                                                                                | Yes                                                 |                                                              |
|            |                | <i>Query Timeout Handling</i>      |                                     |                                                                                | Yes                                                 |                                                              |
|            |                | <i>Custom Sink Reporting</i>       |                                     | Yes                                                                            | Yes                                                 | Yes                                                          |
|            |                | <i>GeoSPARQL Benchmarking</i>      | Yes                                 | Yes                                                                            | Yes                                                 |                                                              |
|            |                | <i>SPARQL Benchmarking</i>         |                                     |                                                                                |                                                     | Yes                                                          |

# BLINKG: A Benchmark for LLM-Integrated Knowledge Graph Generation

Carla Castedo ✉ 

Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS),  
Universidade de Santiago de Compostela, Spain

Enrique Iglesias ✉ 

L3S Research Center Germany, Hannover, Germany

Manuel Lama ✉ 

Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, Spain  
Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS),  
Universidade de Santiago de Compostela, Spain

Alberto Bugarín-Diz ✉ 

Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, Spain  
Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS),  
Universidade de Santiago de Compostela, Spain

Maria-Esther Vidal ✉ 

L3S Research Center Germany, Hannover, Germany  
TIB Leibniz Information Centre for Science and Technology, Hannover, Germany  
Leibniz University, Hannover, Germany

David Chaves-Fraga ✉ 

Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, Spain  
Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS),  
Universidade de Santiago de Compostela, Spain

---

## Abstract

Generating Knowledge Graphs (KGs) remains one of the most time-consuming and labor-intensive tasks for knowledge engineers, as they need to identify semantic equivalences between input data sources and ontology terms. While declarative solutions (e.g., RML, SPARQL-Anything) have helped to generalize this process, aligning input schema elements with ontology terms still involves intricate transformations and requires considerable manual effort. With the advent of Large Language Models (LLMs), there is growing interest in leveraging their capabilities to assist KG engineers. Although some studies have explored using LLMs to automate KG construction, there is still no standardized framework for assessing how effectively they establish correspondences between data schemes and onto-

logy concepts. Therefore, in this paper, we propose BLINKG, a benchmark designed to evaluate the mapping capabilities of LLMs in constructing KGs from heterogeneous data sources. The benchmark includes a set of scenarios with increasing complexity, based on real-world use cases. We conduct an extensive experimental evaluation of several state-of-the-art LLMs using BLINK and observe that they already offer promising solutions. However, their performance remains limited in complex scenarios. Thanks to this benchmark we can already assess the current capabilities of LLMs for KG construction. Additionally, we define a set of requirements for achieving (semi)automated (LLM-driven) KG construction, opening new research lines in this area.

**2012 ACM Subject Classification** Information systems → Data management systems; Information systems → Semantic web description languages; Computing methodologies → Ontology engineering

**Keywords and phrases** Knowledge Graph Construction, Benchmarking, Mapping Languages, Large Language Models

**Digital Object Identifier** 10.4230/TGDK.4.2.9

**Category** Resource



© Carla Castedo, Enrique Iglesias, Manuel Lama, Alberto Bugarín-Diz, Maria-Esther Vidal, and David Chaves-Fraga;  
licensed under Creative Commons License CC-BY 4.0

*Transactions on Graph Data and Knowledge*, Vol. 4, Issue 2, Article No. 9, pp. 9:1–9:35



Transactions on Graph Data and Knowledge

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



**Related Version** *Full Version:* <https://arxiv.org/abs/2605.19518>

**Supplementary Material** *Dataset:* <https://doi.org/10.5281/zenodo.15971734>

*Software:* <https://github.com/citiususc/blinkg>

archived at `swb:1:dir:70c919275904a265a8d761adf4797f02724d2171`

**Funding** The members from University of Santiago de Compostela are funded by the Agencia Estatal de Investigación (Spain) (PID2023-149549NB-I00 & CPP2024-011786), the Xunta de Galicia – Consellería de Cultura, Educación, Formación Profesional e Universidades (Centro de investigación de Galicia accreditation 2024–2027 ED431G-2023/04) and the European Union (European Regional Development Fund – ERDF).

**Acknowledgements** We want to thank Jhon Toledo, Ana Iglesias-Molina, Oscar Corcho and Daniel Garijo for the discussions and ideas behind the proposed benchmark. We also want to thank Javier Garea for his support and feedback in the use of BLINKG

**Received** 2025-07-16 **Accepted** 2026-03-18 **Published** 2026-09-03

**Editors** Stefania Dumbrava, George Fletcher, Matteo Lissandrini, and Riccardo Tommasini

**Special Issue** Data Management for (Knowledge) Graphs

**Generative AI Declaration** The authors used OpenAI to assist with system implementation, figure preparation based on experimental results, and language polishing (including grammar and style checking). All outputs were carefully reviewed and validated by the authors.

## 1 Introduction

Knowledge engineering (KE) covers tasks required to make data and knowledge computationally accessible, tasks that traditionally depend on extensive manual work by domain experts and knowledge engineers. Large Language Models (LLMs) are increasingly used to support these processes, such as formulating competency questions [44], taking on expert roles in ontology engineering methodologies [15, 53], and verbalizing knowledge graph queries [37]. The growing number of LLM-based KE approaches has also motivated new benchmarking and evaluation initiatives like Alharbi et al. [2], Garijo et al. [17], Herwanto et al. [19], Rebboud et al. [43], and Tsaneva et al. [49] to assess their current capabilities and limitations.

Constructing KG from heterogeneous data sources is one of the most time-consuming and manual tasks that a knowledge engineer must perform [14]. The difficulty of the task lies in the need to understand the ontology terms and map the input data to these terms [39]. In some cases, the process can be relatively simple (e.g., mapping a table Sport to an ontology class Sport), but in many situations it requires a deeper understanding of the data and the domain. Often, domain experts are needed to correctly interpret implicit semantics and resolve ambiguities between the data and the ontology. In large-scale, real-world projects [7, 45], this task is often a bottleneck, with the mapping phase frequently exceeding six person-months [7].

Several solutions have recently explored using LLMs to reduce the effort required for semantic alignment between data sources and ontologies. For example, in Hofer et al. [22], the authors propose an automated KG construction pipeline where RML mappings [27] are generated by an LLM using the ontology and source data as inputs. In Schmidt et al. [46], the generation of YARRRML mappings [21] with LLMs is explored in the manufacturing domain. Similarly, [R2]RML-ChatGPT [42] presents a framework to refine mapping rules using ChatGPT. Recently, Freund et al. [16] evaluates its proposal (the *ReMap* tool) against LLM-based RML [27] mapping generation. However, all these solutions have been evaluated across diverse scenarios and use cases, using different parameters and metrics, which makes it difficult to compare them fairly. In

the SW and KG community, SemTab<sup>1</sup> is positioned as an initiative and challenge for automatic annotation of tabular data, providing its own benchmark [28] for testing solutions that automate table interpretation. However, it typically targets an already created knowledge graph (e.g., Wikidata) rather than a non-populated ontology, focuses on three specific tasks (Cell Entity Annotation, Cell Property Annotation, and Cell Type Annotation), and most of the approaches are implemented using end-to-end pipelines, limiting traceability and transparency.

In this context, to assess the potential of LLMs for generating semantic mappings from heterogeneous data sources to ontology terms, we present BLINKG, a **Benchmark for LLM-Integrated Knowledge Graph Generation**. The goal is to support the generation of explicit and declarative mappings, providing traceability and transparency in the knowledge graph construction process, as opposed to black-box end-to-end approaches. The main contributions of this work are summarized as follows:

- **C1:** A comprehensive and domain-agnostic framework composed of multiple scenarios, gold standards, and evaluation metrics designed to assess the behavior of LLMs in KG construction
- **C2:** Three progressively complex scenarios that reflect real-world challenges in KG construction, each covering a representative set of typical mapping tasks.
- **C3:** Open and reusable resources, fostering reproducibility and facilitating adoption and extension by the community<sup>2</sup>.
- **C4:** A comprehensive evaluation of six state-of-the-art LLMs, analyzing their performance and generalization capabilities, focusing on traceability and transparency.
- **C5:** Practical recommendations, lessons learned, and new research directions for advancing the (semi-)automation of LLM-driven knowledge graph construction.

The paper is structured as follows: Section 2 describes the related work on the automation of KG Construction, with special focus on novel proposals that use LLMs. Section 3 describes the BLINKG benchmark, with the proposed scenarios and metrics. Section 4 presents the experimental evaluation of our benchmark over state-of-the-art LLMs, and Section 5 reports the main points of discussion. We finalize the paper with the conclusions and future work in Section 6.

## 2 Related Work

Despite recent advances in automatic knowledge extraction, the creation of knowledge graphs remains an inherently manual and resource-intensive process. The main challenge lies in establishing semantic correspondences between heterogeneous data sources and the concepts defined in the target ontologies. In complex scenarios, this process requires not only a knowledge engineer but also a domain expert who can accurately identify and validate these mappings. Although declarative tools help streamline and simplify the process [4], it remains a challenging task that heavily relies on domain expertise. Existing work can be roughly grouped into (i) approaches that focus on mapping generation, mostly focused on relational databases as input sources, (ii) benchmarks and datasets that evaluate partial aspects of the KG Construction process, and (iii) more recent LLM-based systems, which exhibit different strengths and failure modes from traditional methods.

With the introduction of R2RML [10] and Direct Mapping [3], W3C recommendations for defining mappings between relational databases (RDB) and ontologies in the context of Ontology-Based Data Access (OBDA) [51], a number of approaches emerged aiming to automate the creation

<sup>1</sup> <https://github.com/sem-tab-challenge>

<sup>2</sup> <https://github.com/citiususc/blinkg>

of these mapping rules. Most of these approaches followed the directives of Direct Mapping, which defines a set of rules to produce a plain RDF representation of a relational database without considering any ontology. MIRROR [12], D2RQ [5], and Ontop [6] follow a similar approach, extracting from the Relational Database (RDB) schema a target ontology and the mapping correspondences. On the other hand, AutoMap4OBDA [47] and BootOX [29] consider an input ontology and generate actual R2RML mappings from the RDB. However, these solutions are limited to relational databases and rely heavily on heuristic-based approaches for mapping generation, which significantly constrains their applicability and flexibility in broader Knowledge Graph Construction (KGC) workflows that involve multiple formats, complex transformations, and non-relational sources.

All these proposals used RODI for the evaluation, a benchmark for RDB-to-ontology mapping generation [38]. RODI is designed to evaluate the quality of system-generated relational-to-ontology mappings, aiming to provide a generic and comparable evaluation framework for mapping generation systems. RODI employs an end-to-end evaluation approach, assessing mapping utility by comparing the accuracy of SPARQL query results over the generated RDF data against reference SQL query results on the original relational database. Similar to our proposal the benchmark incorporates diverse test scenarios from domains such as scientific conferences, geographical data, and oil and gas exploration. These scenarios are configured with databases, ontologies, and specific query workloads designed to test a variety of mapping challenges, including naming conflicts, structural RDB heterogeneity (e.g., normalization, denormalization, class hierarchies, key conflicts, and dependency conflicts). However, the authors acknowledge certain limitations in the scope of evaluation, particularly concerning the complexity of data transformations. RODI explicitly excluded complex data transformations such as unit conversions, string cleaning, or string compositions from its default benchmark configurations. This exclusion was justified by the observation that, at the time of its publication, “no current relational-to-ontology mapping generation system implements any such transformation functionality”. Therefore, the benchmark primarily focused on evaluating systems’ capabilities in RDB-schema-level matching and mapping rule generation, often resolvable through heuristic techniques, rather than addressing more complex data-level transformations or advanced relationships beyond basic structural mappings. In summary, RODI focuses on relational-to-ontology mappings, primarily evaluating schema-level correspondences and basic structural mappings over relational databases. BLINKG shares this emphasis on mapping generation, but generalises it beyond RDBs to multiple data formats (CSV, JSON, XML) and decomposes the mapping problem into a richer set of subtasks (e.g., class selection, subject generation, property selection, data reference identification, functions, joins), enabling more fine-grained and format-agnostic evaluation.

Beyond relational databases, the recent SemTab challenge<sup>3</sup> introduces a collection of tabular datasets [28] aimed at automatically matching them to external knowledge graphs such as DBpedia and Wikidata. Several solutions have been proposed, leveraging techniques ranging from heuristic rules and fuzzy matching (e.g., JenTab [1], DAGOBAB [24], and MTAB4D [32]) to knowledge graph embeddings (e.g, TorchicTab [11]). However, SemTab is primarily designed as a benchmark for evaluating end-to-end systems that annotate tables against a pre-existing knowledge graph, rather than aligning them with an ontology. Moreover, most of these systems do not produce explicit, declarative mappings, making it difficult to understand or reproduce how the annotations are actually generated.

In this context, Large Language Models (LLMs) have emerged as a promising solution to automate the transformation of (semi)structured data into KGS, leveraging their advanced natural language understanding capabilities. Several recent studies have explored the use of LLMs for

---

<sup>3</sup> <https://www.cs.ox.ac.uk/isg/challenges/sem-tab/>

■ **Table 1** Comparison between RODI, SemTab, and BLINKG in terms of their main focus, evaluation strategy, dependency on existing KGs, supported input formats, granularity and expected mapping outputs.

|                         | RODI            | SemTab                     | BLINKG            |
|-------------------------|-----------------|----------------------------|-------------------|
| Main focus              | RDB-to-ontology | Table/column/entity annot. | Data-to-ontology  |
| Evaluation              | SPARQL vs. SQL  | Links to KG                | Explicit Mappings |
| Granularity             | Graph-level     | Annotation-level           | Task-level        |
| Requires existing KG    | No              | Yes                        | No                |
| Supported formats       | RDB             | Tabular data               | Any               |
| Expected Mapping Output | R2RML           | Not explicit               | Table/RML         |

generating declarative mapping rules. For instance, Schmidt et al. [46] investigate LLM-assisted and context-enhanced YARRRML mapping generation in the manufacturing domain, addressing the challenge of consolidating inventory data in large companies such as Bosch. Their approach supports schema evolution and data integration through semi-automated mapping generation, and they report both qualitative and quantitative evaluations.

In the construction sector, Höltgen et al. [23] examine how LLMs can be used to convert relational infrastructure data into RDF using a four-step pipeline: SQL query generation, SQL-to-ontology mapping, R2RML mapping generation, and final RDF materialization. Although their results indicate that current LLMs struggle to fully automate this pipeline – due to limitations in handling R2RML syntax and a tendency to hallucinate ontology elements – models like GPT-4o show promising performance, especially under few-shot prompting. In particular, they note the model’s ability to infer implicit relationships in join tables.

Similarly, Hofer et al. [22] explore the use of LLMs for generating RML files from JSON data in the context of movie knowledge graphs. Their evaluation, using data from the IMDB and a target movie ontology, shows that Claude 3 Opus outperforms GPT-4 in mapping accuracy, while earlier models (e.g., Claude 2.1, GPT-3.5, Gemini-Pro) often produce invalid outputs, highlighting the models’ difficulty in handling precise syntax and semantically constrained tasks. Freund et al. [16] introduce ReMap, a deterministic reverse engineering pipeline to generate RML mappings from non-RDF source data and an expected RDF graph. In their evaluation, mappings generated by LLMs were used as a comparison baseline to assess the performance of ReMap. Additionally, Val-Calvo et al. [50] developed OntoGenix, an LLM-powered pipeline for automating ontology development that also generates mappings and RDF data.

While these studies highlight the growing interest in using LLMs for mapping generation, their evaluation strategies remain inconsistent and fragmented. Each proposal defines its own tasks, datasets, and success criteria, making it difficult to compare results or generalize findings. Moreover, most of the evaluations focus primarily on syntactic correctness such as the validity of the generated RML or R2RML files, rather than on the semantic quality or relevance of the mappings. There is a lack of shared tasks, reference datasets, and standardized evaluation parameters. This fragmentation underscores the need for a comprehensive and systematic benchmark like BLINKG, which provides a common ground for evaluating LLM-generated mappings across diverse scenarios, with clear tasks, gold standards, and metrics.

From this discussion, three insights emerge. First, RODI evaluates mappings indirectly, by measuring query-answering quality over the constructed RDF graph, so mapping errors are only observed through their impact on SPARQL results. In contrast, BLINKG focuses on the mapping layer itself: it evaluates the individual mapping decisions (e.g., class, property, subject template, data reference) before materialisation, providing direct evidence of where systems succeed or fail.

Second, SemTab assumes an existing knowledge graph and concentrates on entity linking and annotation against that graph, whereas BLINKG targets the prior step of mapping heterogeneous data sources to an ontology, without requiring a pre-populated KG; in this sense, SemTab and BLINKG are complementary, but they address different parts of the pipeline. Third, BLINKG generalises these ideas beyond relational databases to multiple data formats (CSV, JSON, XML) and offers explicit gold standards and metrics for a richer set of mapping-related subtasks, which makes it suitable for systematically evaluating both LLM-based and non-LLM systems under a common, mapping-centric perspective. In Table 1, we present an overview of this comparison between RODI and SemTab with BLINKG.

### 3 BLINKG: A Benchmark for LLM-Integrated Knowledge Graph Generation

To evaluate the performance of automation in mapping generation between ontology and input sources, we designed a benchmark comprising three different scenarios that reflect varying levels of alignment between data sources and ontology schemes: (i) Basic, where column names and ontology terms are lexically similar; (ii) Schema-aligned, where source schemes follow the structure of the ontology but with less obvious lexical overlap; and (iii) Schema-distant, where the source schema is semantically related to the ontology but structurally and lexically distant. These scenarios allow us to systematically test the ability of LLMs to generalize across different mapping difficulties. We also define a set of metrics to evaluate the quality of the generated mappings, considering precision, recall, and F1-score at the level of the task.

#### 3.1 Knowledge Graph Construction Tasks

To define the tasks considered in this benchmark, we conducted a thorough review of: i) existing related initiatives, such as SemTab [28]; ii) the capabilities defined by declarative mapping languages, mainly extracted from the complete analysis presented in Iglesias et al. [26] and the mapping features presented in Chaves-Fraga et al. [8]. Based on this analysis, we identified the following key tasks:

- **Ontology Class Identification:** This task involves identifying the class of the instance to be generated. For example, if the ontology defines the class `foaf:Person` and the input data contains a `person-id`, the output would include a triple such as: `ex:person/{id} rdf:type foaf:Person`. This task is related to the Column Type Annotation (CTA) task proposed in SemTab [28]. However, a key distinction is that input files frequently contain information about multiple types of entities (e.g., denormalized CSV files, XML trees, or JSON documents). Meanwhile, CTA is made at the table level.
- **Subject Generation:** This task refers to the generation of the subject in RDF triples. It typically involves two key aspects: i) Following best practices for URI creation [41], usually by combining the base URI (e.g., `http://example.org`) with the class label (e.g., `Person`); and ii) Identifying data references that ensure instance uniqueness within the dataset, similar to a primary key (PK) in relational databases. In this work, we refer to subject generation as *simple* when a single data reference is sufficient to identify an instance, and as *complex* when multiple data references must be combined, analogous to composite primary keys in relational databases. While this task is somewhat related to the Cell Entity Annotation (CEA) task in SemTab, it is fundamentally different: CEA focuses on linking a cell value to an existing entity in a KG, whereas the task here is about creating new entities.
- **Ontology Property Identification:** This task involves selecting the ontology property that links either a data reference or a related entity class, ensuring the property's domain matches the identified class. For example, given a column named `fullname` and an ontology

property `foaf:name` whose domain is `foaf:Person`, it is very likely that they are semantically equivalent. This task resembles the Column Property Annotation (CPA) task in SemTab; however, our approach leverages ontology restrictions (property domains), whereas CPA relies on already instantiated entities to discover such properties in the target KG.

- **Data Reference<sup>4</sup> Identification:** This task consists of identifying the reference to the input data, such as a CSV column, a JSONPath/XPath expression, or an SQL query, that will be used to construct triples. These references are typically used as the object of a triple when the identified property is a datatype property. Similarly to subject generation, we refer to object generation as *simple* when a single data reference suffices to construct the object value, and as *complex* when multiple data references need to be combined via standard string concatenation to obtain the final literal used in the triple.
- **Related Entity Class and Joins:** For object properties defined in the ontology, this task involves specifying how the subject of the related entity (i.e. the range of the property) is generated and under which conditions the relationship is established (i.e., the join condition). For instance, continuing with our earlier example, the triple `ex:person/1 foaf:works ex:organization/1` would be generated for the object property `foaf:works` (domain: `foaf:Person`, range: `foaf:Organization`) under the condition that `person.org` equals `organization.id`.
- **Language Annotation:** This task entails assigning the correct language tag to text literals based on their content and context. For each string value in the dataset, such as labels, comments, or descriptions, the system must detect the natural language and annotate it with the appropriate tag (e.g., `@en` for English, `@es` for Spanish).
- **Datatype Annotation:** This task consists of determining and assigning to each literal the most appropriate RDF datatype IRI, often drawn from XSD but not restricted to it, based on its lexical form and semantic intent. For example, strings matching ISO-8601 date patterns might be annotated as `xsd:date`, integer numeric values as `xsd:integer`, boolean-like values as `xsd:boolean`, or any custom datatype.
- **Transformation Functions:** This refers to any transformation that must be applied to data references before triple generation. For instance, dates may need to be transformed from heterogeneous input formats (e.g., "26/11/2025" or "11-26-2025") into a canonical representation such as `xsd:date` literals, or boolean-like codes (e.g., 1/0, "Y"/"N") may need to be normalised to explicit `true/false` values before triple generation. Additionally, one common task is to convert enumerated values (e.g., SQL ENUMs) into standardized taxonomies, such as those represented using SKOS, where each enumerated code is systematically mapped to the URI of the corresponding SKOS concept (e.g., mapping "OPEN" or "RESTRICTED" in a procurement system to the appropriate SKOS concepts in a controlled vocabulary). This is particularly relevant in scenarios where KGs leverage standardized datasets (e.g., authority tables from the European Commission that provide standardized taxonomies for public procurement procedures, country names and codes, or currencies<sup>5</sup>). These kinds of transformations are aligned with the function mapping capabilities defined in the RML-FNML module<sup>6</sup>, which explicitly supports the specification of such data transformation functions in mapping documents.

<sup>4</sup> The concept is defined in the RML ontology: <http://w3id.org/rml/core/spec#reference-rml-reference>

<sup>5</sup> <https://op.europa.eu/en/web/eu-vocabularies/authority-tables>

<sup>6</sup> <https://w3id.org/rml/fnml/spec>



■ **Table 2** Summary of ontology metrics and input data characteristics.

| Scenarios | Classes | Data prop. | Object prop. | SKOS | Input features                          |
|-----------|---------|------------|--------------|------|-----------------------------------------|
| 1         | 3       | 8          | 3            | 0    | Artificial data based in RML test cases |
| 2         | 17      | 49         | 10           | 10   | GTFS-Madrid’s Metro feed                |
| 3         | 11      | 4          | 10           | 3    | Real life data from CODICE              |

■ **Table 3** Features distributed by each scenario. Scenario 1 is divided in several atomic cases, while Scenarios 2 and 3 represent realistic KG construction scenarios.

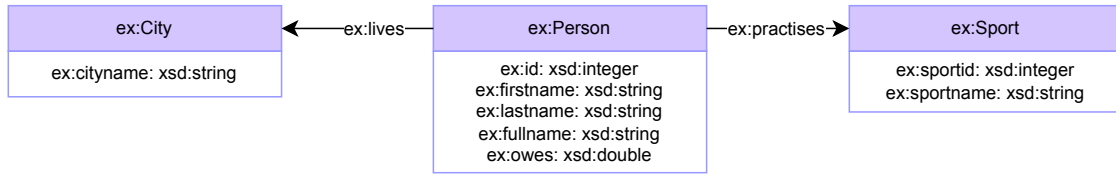
| Features/Scenarios          | Scenario 1 |    |    |    |    |    |    |    | Scenario 2 | Scenario 3 |
|-----------------------------|------------|----|----|----|----|----|----|----|------------|------------|
|                             | 1A         | 1B | 1C | 1D | 1E | 1F | 1G | 1H | GTFS       | PPDS       |
| One data reference          | x          | x  | x  | x  | x  | x  | x  | x  | x          | x          |
| Two or more data references |            | x  | x  | x  | x  | x  | x  | x  | x          | x          |
| Complex object generation   |            |    |    | x  |    | x  | x  |    | x          | x          |
| Simple subject generation   | x          | x  | x  |    | x  |    |    | x  | x          | x          |
| Complex subject generation  |            |    |    | x  |    | x  | x  |    | x          | x          |
| Self join                   |            |    | x  |    |    |    | x  |    | x          | x          |
| Join                        |            |    |    |    | x  | x  | x  |    | x          | x          |
| Two or more input sources   |            |    |    |    |    | x  | x  | x  | x          | x          |
| Datatypes generation        |            |    |    | x  |    |    | x  | x  | x          | x          |
| Language annotations        |            |    |    |    |    |    | x  | x  |            |            |
| Transformation Functions    |            |    |    |    |    |    |    |    | x          | x          |
| Distant Schemes             |            |    |    |    |    |    |    |    |            | x          |

Although we have split the work into separate tasks for clarity, experts tackle them as a single, intertwined challenge: parsing diverse data formats, mapping fields to the right classes and properties, picking accurate datatypes and language tags, and applying any needed transformations. Pulling this off usually demands deep domain knowledge, meticulous schema analysis, and plenty of careful judgment. Additionally, it would be possible to include more advanced tasks based on the capabilities of newer declarative mapping languages (e.g., RML [27]), such as list generation (RML-CC), statements about statements (RML-Star), or advanced input/output descriptions (RML-IO), we decided not to include them in this first version of the benchmark. We consider these to be advanced features, and our current focus is on understanding how proposed solutions perform on the core aspects of KG construction. These more complex features are planned for inclusion in a future version of the benchmark, when more tools are expected to support and process such constructs efficiently.

### 3.2 Benchmark Scenarios

For the BLINKG benchmark, we define three scenarios designed to evaluate the performance of automation proposals, categorizing them by difficulty. Each scenario comprises multiple use cases that adhere to its difficulty level. All use cases within a scenario share a common ontology and controlled vocabularies, ensuring the comparability of the results. Each case is accompanied by one or more datasets in various formats, as well as an expected output that serves as the baseline for evaluation. Table 2 presents the statistics and data sources of each scenario while Table 3 provides an overview of the features included in each scenario, which are described in the following sections.





■ **Figure 1** A basic ontology used for the Scenario 1.

### Scenario 1: Basic Knowledge Graph Construction

This scenario comprises a total of eight atomic cases, and it is inspired by the features defined in RML-core [36] and RML test cases [20]. Its objective is to evaluate the creation of KGs for basic use cases with simple input sources. An ontology was created to accommodate all of them, shown in Figure 1. It is comprised of three classes, related by two object properties, and for each class, there are several data properties.

Input data is provided for each use case in three different formats with the same content: CSV, JSON, and XML. The selection of one format over the other will be an evaluation choice. Regardless of the selected format, the resulting KG will remain the same.

In this scenario, the aim is to test basic and atomic behavior, so all the cases are chosen for this purpose. Case 1A seeks to test the ability to generate a unique mapping, which relates a single column to a data property. The entry refers to the name of an entity whose class is a **ex:Person**. Since there is insufficient information to conclude whether it is a first name or a last name, the property to be joined with is **ex:fullname**. As this is the only information available, the subject URI is generated by the same column, acting as a primary key.

Use case 1B increases the complexity of the previous one, performing two mappings on the same entity instead of one, using the same file source in both cases. Each mapping involves a single column and a data property. In addition to the previous use case, the **ex:id** property would be linked this time, changing the primary key from the column **Name** to the column **ID**. As a result, the subject URI would be generated by the **ID** column.

For use case 1C, two mappings are considered again, coming from the same file, but referring to entities from two different classes, **ex:Person** and **ex:Sport**. Thus, each mapping relates to a single data property of each of the entities involved, **ex:fullname** and **ex:sportname**. The URIs of the subjects are generated by the corresponding column for each of the entities. By having two entities from different classes, they can be related to each other through an object property, **ex:practises**, employing a self-join of the triples mapping.

For the 1D use case, four mappings are performed over entities from class **ex:Person**. Three of them directly relate a column to a data property, such as the first name, last name, or owed amount. The other one links **ex:fullname** with a literal generated by concatenating the columns for first name and last name. As there is no identifier, the primary key will be established as the concatenation of the first name and last name. The comprehension of the URI concept would be tested, as there are two repeated rows, which refer to the same entity, so the same URI must be considered for both rows. Finally, data types will be checked for consistency with the ontology, as more complex types such as floating-point numbers are included.

For use case 1E, two entities would be involved, **ex:Person** and **ex:Sport**, but this time the source files would be different for each entity. Students and sports would be in different files. This does not change the linking of data properties but adds a new difficulty to the establishment of an object property between entities. To achieve that, a conditional join would be necessary that states explicitly the equivalence between the **Sport** column in the student file and the **ID** column in the sport file.

Use case 1F explores an alternative way of establishing an object property between entities, by using an additional file that contains the information of the relation. The entities are the same as in the previous use case, but the relation between them is established in a third file. This file contains the ID of the student and the ID of the sport, so two conditional joins would be necessary between the ID column of the student file and the `Student_ID` column of the relation file, and between the ID column of the sport file and the `Sport_ID` column of the relation file.

For use case 1G, entities from both classes `ex:Person` and `ex:City` are considered. The added difficulty in this use case lies in linking the object property `ex:livesIn` correctly. There are two possible ways of doing it: by a self-join within the same file or by a conditional join between two different files. In this case, unlike the previous ones, the direction to take is not so direct and clear. To generate the mapping, the LLM must be able to avoid being influenced by ambiguity and focus on decision-making. Additionally, the use of language tags for literals is tested, as the city name is a string with language annotations. Although language information is not provided, the context suggests that all city names are in English.

Use case test 1H evaluates the ability to add language annotations to the same data properties of a single entity. Specifically, it involves two instances of the class `ex:City`, each representing the same city but with names in different languages (English and Spanish). These language differences must be expressed using language tags, not by creating separate entities for each language version. The city identifier serves as the primary key, so the subject URI must remain the same across both language annotations.

In summary, Scenario 1 provides eight atomic cases over a minimal three-class ontology and plain and simple inputs to validate each core mapping task: from class and subject generation to property identification, joins, datatype, and language annotations. By keeping the setup basic yet varied, we can pinpoint the strengths and weaknesses of automatic pipelines on these foundational building blocks of KG construction.

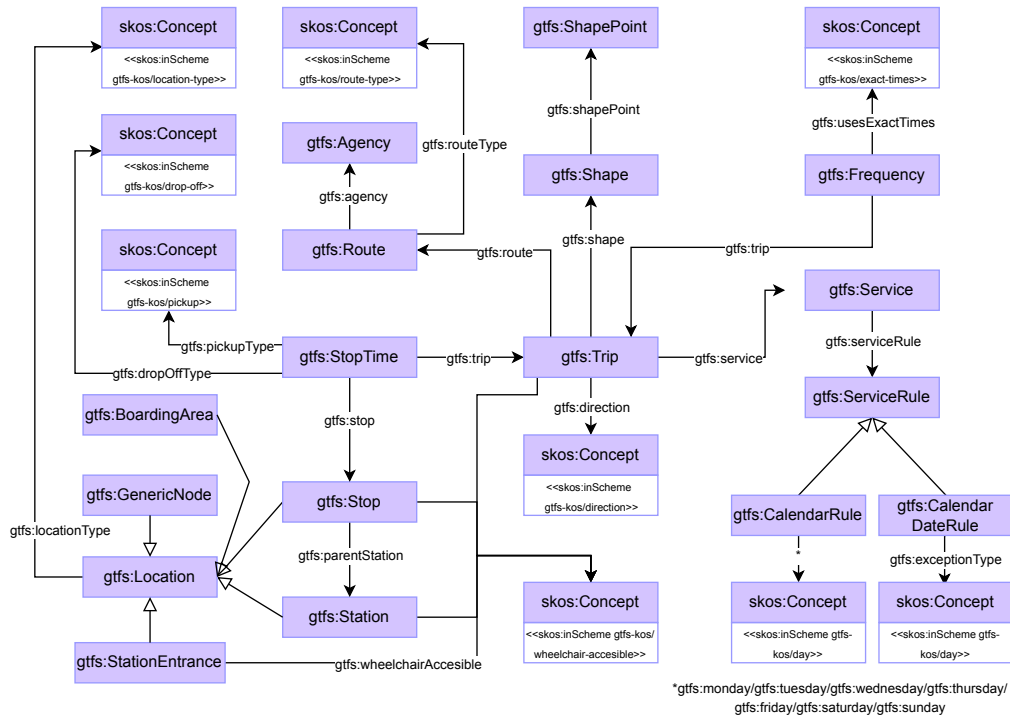
## Scenario 2: Schema-Aligned Knowledge Graph Construction

This scenario focuses on building a KG in the transportation domain. We drew inspiration from the GTFS-Madrid-Bench [9] use case, a benchmark designed to evaluate performance throughout the KG construction process. Unlike Scenario 1, which isolates individual tasks, this one presents a more realistic and complex setting where those tasks frequently overlap and interact. However, the main feature of this scenario is the very high alignment between the input data and the ontology. The ontology itself was built bottom-up directly from the official GTFS specification<sup>8</sup>. In other words, the ontology can be seen as a near-mirror of the GTFS specification, yet it diverges in important ways as OWL and RDF supports richer modeling constructs (e.g. complex class hierarchies, property restrictions, controlled vocabularies) that GTFS's flat, tabular-based format cannot express.

The goal of this scenario is to assess whether automated solutions operating in a context where input schemes and the ontology closely align can still understand ontology peculiarities, such as richer modeling constructs, for correctly transforming the input data into a KG. From a technical perspective, it has the objective of testing, in addition to everything included in Scenario 1 (see Table 3), the handling of transformation functions and the management of a larger amount of data, as in a more realistic use case. The ontology considered is represented in Figure 2, and has seventeen classes that are described in Table 4. The input data are 10 files extracted from the

---

<sup>8</sup> <https://gtfs.org>



■ **Figure 2** The Linked-GTFS Ontology<sup>7</sup>, used for Scenario 2. We omit datatype properties for the sake of simplicity. White arrows indicate that the source class is an `rdfs:subClassOf` of the target class, whereas black arrows represent object properties between classes.

GTFS-Madrid-Bench [9] database<sup>9</sup>. Two examples of input sources (stops and routes) are shown in Listing 1 and Listing 2. The information contained in them covers the entire set of classes and properties considered in this scenario.

The main difference from Scenario 1 is the introduction of more complexity at the ontology level, such as class hierarchies and constraints on classes and properties. The ontology introduces two distinct hierarchies: one for describing locations (stops, stations, station entrances, boarding areas, and generic nodes) and another for modeling service calendars. The calendar hierarchy maps directly to GTFS, since the specification defines two separate input files (`calendar.txt` and `calendar_dates.txt`), each corresponding to one of these classes. In contrast, the location hierarchy depends on the value of the `location_type` column in `stops.txt` (see Listing 1), adding complexity to the mapping rules for these subclasses. Notably, the same `location_type` value also determines inter-location relationships via the `parent_station` property, which entails a self-join on `stops.txt` to generate different subclasses.

In terms of constraints, many properties such as `gtfs:id`, apply to multiple classes, forcing its domain to be defined as the union of those classes and increasing the complexity of linking classes to input sources. We also propose the use of value restrictions on datatype properties. For example, the `gtfs:longName` of class `gtfs:Route` must start with an uppercase letter and consist of an alphanumeric string. All such restrictions may be encoded as transformation functions that ensure incoming data conforms to the ontology’s rules before KG generation.

<sup>9</sup> Note that the benchmark provides support for different data formats such as RDB, CSV, XML, or JSON

■ **Table 4** Class descriptions for ontology of Scenario 2.

| Class            | Description                                                               |
|------------------|---------------------------------------------------------------------------|
| Agency           | Transit companies with service.                                           |
| BoardingArea     | Location where passengers can board and/or alight vehicles.               |
| CalendarRule     | Service dates specified using a weekly schedule with start and end dates. |
| CalendarDateRule | Exception dates for the services.                                         |
| Frequency        | Trip gap for frequency service or condensed schedule.                     |
| GenericNode      | A location within a station, not matching any other location type.        |
| Location         | Places where vehicles pick up or drop off riders.                         |
| Route            | Group of trips that are displayed to riders as a single service.          |
| Service          | Set of ServiceRules.                                                      |
| ServiceRule      | Rule that associates dates with services (CalendarRule/CalendarDateRule). |
| Shape            | Rules for mapping vehicle travel paths.                                   |
| ShapePoint       | One point in a Shape.                                                     |
| Station          | Large transit location that may contain multiple Stops.                   |
| StationEntrance  | Location where passengers can enter or exit a station from the street.    |
| Stop             | Physical location where a vehicle stops or leaves.                        |
| StopTime         | Times that a vehicle arrives at and departs from stops for each trip.     |
| Trip             | Sequence of two or more stops that occur during a specific time period.   |

■ **Listing 1** Excerpt of the input source stops in CSV.

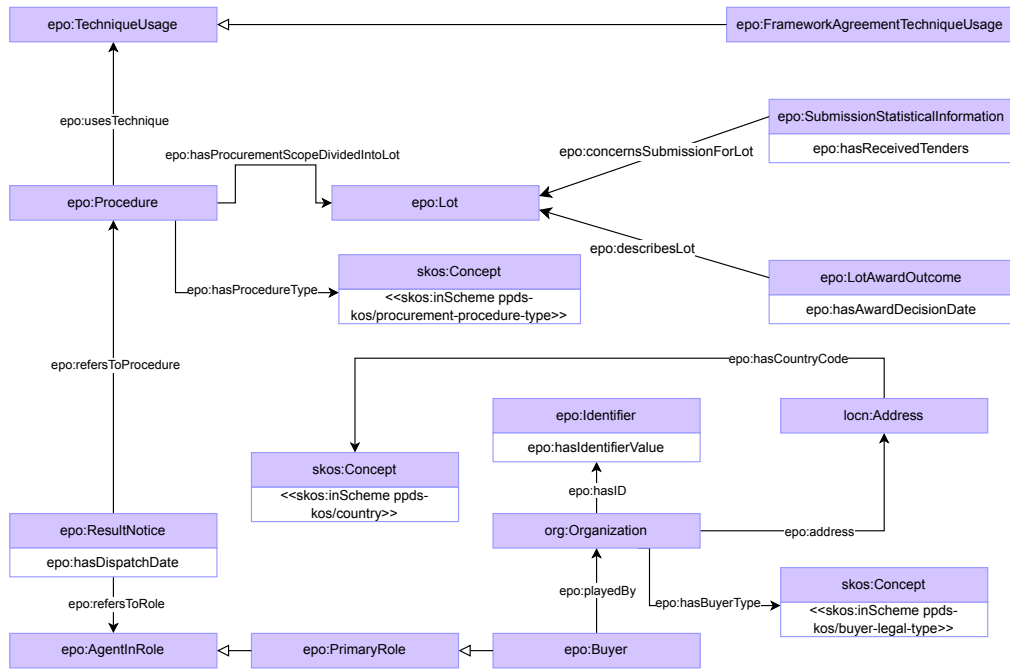
```
stop_id,stop_code,name,stop_desc,lat,lon,location_type,parent_station,wheelchair_boarding
acc411,1,Plaza de Castilla,P. de la Castellana 189,40.46682,-3.68918,0,est9021,0
acc41040,1,Ascensor,Pz. de Castilla 9,40.46555,-3.68877,2,est9021,1
acc41043,1,Intercambiador Superficie,P. de la Castellana 191 B,40.46728,-3.68915,2,est9021,0
acc41044,1,Ascensor,P. de la Castellana 189,40.46702,-3.68918,2,est9021,0
acc412,1,Castellana impares,P. de la Castellana 189,40.46722,-3.68952,2,est_90_21,0
acc413,1,Bravo Murillo,C. de Bravo Murillo 377,40.46629,-3.69036,1,est_90_21,1
```

Finally, the scenario incorporates controlled vocabularies. When input data contains enumerated types (data types defined by a fixed set of named values), these are usually transformed into independent taxonomies outside the ontology, often leveraging the W3C SKOS standard vocabulary<sup>10</sup>. By extracting the enumerated fields from GTFS, we generated eleven controlled vocabularies: Wheelchair Accessibility, Location Type, Route Type, Pickup, Drop-off, Direction ID, Bikes Allowed, Timepoint, Service Availability, Exception Type, and Exact Times. The essential task in this phase is to interpret each input value and map it to the corresponding concept within its scheme. For example, GTFS defines three numeric codes for bicycle access (0

<sup>10</sup> <https://www.w3.org/TR/skos-reference/>

■ **Listing 2** Excerpt of the input source routes in CSV.

```
route_id,agency_id,route_short_name,route_long_name,route_type,route_color,route_text_color
41,CRTM,1,Pinar de Chamartín-Valdecarros,1,2DBEFO,FFFFFF
410,CRTM,10,Hospital del Norte-Puerta del Sur,1,005AA9,FFFFFF
411,CRTM,11,Plaza Elíptica-La Fortuna,1,009B3A,FFFFFF
412,CRTM,12,MetroSur,1,A49800,FFFFFF
```



■ **Figure 3** The subset of the e-Procurement Ontology used for the Scenario 3. White arrows indicate that the source class is an `rdfs:subClassOf` of the target class, whereas black arrows represent object properties between classes.

– no information; 1 – allowed; 2 – not allowed), and our controlled vocabulary represents these as URIs: `wheel-chair-kos:no-information`, `wheel-chair-kos:no-information:accessible`, `wheel-chair-kos:no-information:inaccessible`<sup>11</sup>. The main challenge is understanding the meaning of these codes (often provided without context) and assigning them to the correct taxonomy entries. In the KG construction pipeline, this can be handled by transformation functions that convert raw input data into valid controlled vocabulary values.

This scenario brings everything together in a more realistic setting: a GTFS-based ontology that mirrors the standard but adds OWL-style class hierarchies, property constraints, and controlled vocabularies. Beyond the tasks from Scenario 1, we now also put transformation functions and a more complex environment to the test. The goal is to see if automated pipelines can tackle these added challenges and still build a robust KG.

### Scenario 3: Schema-Distant Knowledge Graph Construction

This scenario is the most complex scenario, focusing on building a KG for public procurement. It follows Scenario 2’s methodology but pushes it further by operating in a realistic setting where tasks frequently interact and overlap. The key difference is that, unlike Scenario 2, the input data models and the ontology are not closely aligned. Although they cover the same domain and share similar concepts, their schemes and lexical descriptions diverge significantly, meaning that deep domain expertise is required to create the relationships between the input sources and the provided ontology.

The ontology for Scenario 3 is drawn from the eProcurement Ontology (ePO)<sup>12</sup>, the official, pan-European data model for public procurement maintained by the European Publications Office. Given the domain’s complexity, ePO is organized into 13 ontology modules, each targeting a

<sup>11</sup> `wheel-chair-kos:` is the prefix of <http://transport.linkeddata.es/kos/wheelchair-accessible/>

<sup>12</sup> <https://github.com/OP-TED/ePO>

■ **Table 5** Class descriptions for ontology of Scenario 3.

| Class                            | Description                                                                   |
|----------------------------------|-------------------------------------------------------------------------------|
| AgentInRole                      | Ties an agent to a part they play in a given situational context.             |
| Buyer                            | Role of an agent that awards a contract and/or purchases items.               |
| FrameworkAgreementTechniqueUsage | Technique that establishes the terms governing contracts to be awarded.       |
| Identifier                       | String to distinguish uniquely one instance of an object.                     |
| Lot                              | Division of the services to be procured, allowing the award of contracts.     |
| LotAwardOutcome                  | Result concerning the Lot attributed by the awarder.                          |
| PrimaryRole                      | A primary role within the procurement process that ties an agent to a part.   |
| Procedure                        | Set of administrative activities conducted to conclude one or more contracts. |
| ResultNotice                     | Announcement of the award or non-award of a contract by a buyer.              |
| SubmissionStatisticalInformation | Statistical information about submissions on a given Lot.                     |
| TechniqueUsage                   | Methods used for conducting procurement procedure.                            |

specific aspect of the procurement lifecycle, from contract notices and award procedures to financial executions and legal frameworks. For our purposes, we extracted a focused subset of classes from the epo-core module, which alone defines nearly 150 classes and is under active, continuous development. This ontology supports multiple initiatives, such as the EU Public Procurement Data Space (PPDS)<sup>13</sup>, which aims to harmonize heterogeneous procurement datasets from all EU Member States into a common semantic layer based on ePO. A key objective of PPDS is to support the standardized computation and comparison of public procurement transparency indicators, as published in the Single Market Scoreboard<sup>14</sup>. Moreover, each Member State typically operates its own national procurement platform, with distinct data formats, schemas, and publication workflows. As a result, manually integrating all these sources with respect to the ePO ontology is extremely costly, error-prone, and must be tailored to the specific characteristics of each country.

The ontology used in this scenario is shown in Figure 3. It comprises 13 classes that capture details of public procurement processes, their outcomes, award procedures, and the organizations involved as contractors. Their definitions are also provided in Table 5. Furthermore, each procurement process can be divided into multiple lots, each of which may follow a distinct awarding procedure. From a technical standpoint, this ontology features deep, nested class hierarchies such as `epo:Buyer`  $\rightarrow$  `epo:PrimaryRole`  $\rightarrow$  `epo:AgentRole` and, despite a reduction in datatype properties compared to Scenario 2, it defines a large set of object properties to capture the rich relationships inherent in the public procurement domain. Nevertheless, we observe repetitive restriction patterns w.r.t. Scenario 2, such as properties reused across multiple classes and explicit cardinality constraints imposed on certain classes. Regarding controlled vocabularies, ePO reuses those provided by the Publications Office (the so-called authority tables<sup>15</sup>), so input data must be mapped and linked to these external taxonomies. As shown in Figure 3, this scenario leverages three controlled vocabularies: Procedure Types<sup>16</sup>, Country Codes<sup>17</sup>, and Buyer Types<sup>18</sup>.

<sup>13</sup> [https://single-market-economy.ec.europa.eu/single-market/public-procurement/digital-procurement/public-procurement-data-space-ppds\\_en](https://single-market-economy.ec.europa.eu/single-market/public-procurement/digital-procurement/public-procurement-data-space-ppds_en)

<sup>14</sup> <https://single-market-scoreboard.ec.europa.eu/business-framework-conditions/public-procurement>

<sup>15</sup> <https://op.europa.eu/en/web/eu-vocabularies/authority-tables>

<sup>16</sup> <http://publications.europa.eu/resource/authority/procurement-procedure-type>

<sup>17</sup> <http://publications.europa.eu/resource/authority/country>

<sup>18</sup> <http://publications.europa.eu/resource/authority/buyer-legal-type>

■ **Listing 3** Excerpt from CODICE, the Public Procurement Spanish Platform.

```
<entry>
  <id>https://contrataciondelestado.es/.../6854467</id>
  <title>Suministro de piezas de recambio [...] EMAYA.</title>
  <updated>2021-01-07T10:47:01.782+01:00</updated>
  <summary type="text">
    Id licitacion: 942P LOTE 2; Organo de Contratacion: Gerencia de EMAYA
  </summary>
  <cac-place-ext:ContractFolderStatus>
    <cac:Party>
      <cbc:WebsiteURI>http://www.emaya.es</cbc:WebsiteURI>
      <cac:PartyIdentification>
        <cbc:ID schemeName="NIF">A07000029</cbc:ID>
      </cac:PartyIdentification>
      <cac:PartyName>
        <cbc:Name>Gerencia de EMAYA, Empresa Municipal [...]</cbc:Name>
      </cac:PartyName>
      <cac:PostalAddress>
        <cbc:CityName>Palma</cbc:CityName>
        <cbc:PostalZone>07010</cbc:PostalZone>
        <cac:AddressLine>
          <cbc:Line>
            Camino de los Reyes 400, Edificio Central de Son Pacs
          </cbc:Line>
        </cac:AddressLine>
      </cac:PostalAddress>
    </cac:Party>
  </cac-place-ext:ContractFolderStatus>
  <!--...and more nested tags...-->
</entry>
```

The input data for this scenario comes from the Spanish public procurement platform CODICE<sup>19</sup>. CODICE supports the entire procurement lifecycle and provides a suite of standard XSD schemas to ensure interoperability across all public administrations. Listing 3 shows a brief XML excerpt of these input files, conforming to the official CODICE XSD definitions. The main challenge in transforming this data into a KG based on the ePO ontology lies in the complex, deeply nested XML tree: entities corresponding to each ontology class and their properties are scattered throughout the document. Consequently, the pipeline must parse each branch independently, leveraging element names and attribute values to instantiate ontology classes correctly. Since CODICE and ePO were developed independently (they cover the same domain but lack the high alignment of Scenario 2), a thorough understanding of both models is crucial to establish accurate mappings. From a technical point of view, this scenario incorporates all the challenges already presented in Scenario 2 and adds an extra layer of complexity when defining join conditions for object-property triples. In some cases, these joins must span different levels of the XML tree, linking elements on separate branches, which requires robust mapping logic to correlate related entities accurately.

In summary, Scenario 3 tests the full spectrum of challenges: from multi-level XML parsing and independent schema-ontology alignment to nested joins and controlled-vocabulary linking – within a real-world public procurement setting. By combining the rich, modular ePO ontology with CODICE's complex XML structures and national variations, this scenario pushes automated KG pipelines to demonstrate both domain expertise and flexible mapping strategies under realistic conditions.

<sup>19</sup><https://contrataciondelestado.es/wps/portal/codice>



■ **Table 6** Expected output table from the execution of each experiment.

| Column               | Description                                                  |
|----------------------|--------------------------------------------------------------|
| Data Reference       | Each data reference from the input data source files.        |
| Ontology Property    | Name of the ontology property related with the column.       |
| Entity Class         | Class of the subject considered.                             |
| Related Entity Class | Class of the corresponding object.                           |
| Subject Generation   | Template to generate the subject URI.                        |
| Join                 | Equivalence condition in a join.                             |
| Datatype             | Object’s datatype.                                           |
| Language Annotations | Object’s language tag.                                       |
| Function Name        | Name of the function needed.                                 |
| Function Output      | Transformation applied to data reference using the function. |

■ **Table 7** Example of the expected output from Shape and ShapePoint classes from the LinkedGTFS ontology and the shapes.txt file from GTFS spec. All ontology references starting from `:` mean that are using the base ontology IRI, while all subjects should start with `ex:shape/`. Functions are not shown due to the lack of space.

| Data Ref.    | Property          | Class       | Rel. Class  | Subject (ex:shape/)           | Join Con.              | Datatype    |
|--------------|-------------------|-------------|-------------|-------------------------------|------------------------|-------------|
| shape_id     | :id               | :Shape      |             | {shape_id}                    |                        | xsd:string  |
| shape_pt_lat | :latitude         | :ShapePoint |             | {shape_id}-<br>{shape_pt_seq} |                        | geo:lat     |
| shape_pt_lon | :longitude        | :ShapePoint |             | {shape_id}-<br>{shape_pt_seq} |                        | geo:lon     |
| shape_pt_seq | :pointSequence    | :ShapePoint |             | {shape_id}-<br>{shape_pt_seq} |                        | xsd:integer |
| shape_dist   | :distanceTraveled | :Shape      |             | {shape_id}                    |                        | xsd:float   |
| shape_id     | :shapePoint       | :Shape      | :ShapePoint | {shape_id}                    | shape_id =<br>shape_id |             |

### 3.3 Metrics and Expected Output

Reliable, quantitative metrics are essential for evaluating how well automated systems handle each of the identified tasks involved in KG construction. Defining a fair metric is complex: semantic equivalence can be expressed in many ways, and simple string matching misses valid paraphrases or alternate URIs. It is also known that LLMs struggle with syntax problems when generating RML or SPARQL-Anything rules [22, 30], so BLINKG does not focus on evaluating whether systems can produce syntactically valid mapping languages. Instead, for each scenario we ask the model to produce a structured, tabular output that explicitly lists, for example, the selected classes, properties, and data references for each column or field. A detailed description of the expected output is shown in Table 6 and Table 7 shows illustrative example of this expected output format extracted from the Scenario 2. Thus, for evaluation, we frame each identified task (Ontology Class Identification, Subject URI Generation, Ontology Property Identification, etc.) as a binary outcome: the model’s output is either correct or incorrect. By breaking the pipeline into these subtasks, BLINKG can measure Precision, Recall, and F-score on each task independently.

We quantify performance on each KG-construction task using precision, recall and F-score. Precision measures the fraction of an approach’s correct outputs, recall is the fraction of expert-validated items the model recovers, and F-score balances the two as their harmonic mean. Applying these metrics to each binary task outcome yields a clear, repeatable assessment of model strengths and weaknesses.

Since relying on a single execution of a prompt may lead to unstable or biased results, we propose to run the same prompt multiple times under identical conditions. We then compute a macro-average of all metrics over these runs, where  $n$  denotes the number of evaluations of the same prompt:

$$\begin{aligned} \text{Macro-Precision} &= \frac{1}{n} \sum_{i=1}^n \text{Precision}_i, & \text{Macro-Recall} &= \frac{1}{n} \sum_{i=1}^n \text{Recall}_i \\ \text{Macro-F-score} &= \frac{1}{n} \sum_{i=1}^n \text{F-score}_i \end{aligned}$$

Although these metrics provide good support for quantifying the quality of the results, relying on a manual evaluation is not scalable, and strict string-equality checks fall short: an automatic approach might express the same output with a paraphrase, synonym, or alternate URI that could not match the reference ground truth. Similar to what is done in other domains such as NLP, we enhance precision, recall, and F-score by inserting a preliminary similarity check between each output and the ground truth. We compute three complementary measures:

- Levenshtein distance for a straightforward string-level similarity score.
- Cosine similarity over a language model embeddings (such as SBERT) of the raw output, capturing semantic closeness beyond exact text matches.
- Cosine similarity on ontology-driven verbalizations, where we replace the raw output with the canonical label or description of the target class/property (retrieved via lookup) and computing the embeddings on that text.

Because the identified tasks range from simple class identification to complex joins, we compute all three similarity scores (Levenshtein, raw-embedding cosine and verbalization-embedding cosine) for each candidate mapping and take the highest value. We then compare that top score against a predefined threshold: if it exceeds the threshold, the annotation is marked correct; otherwise, it's deemed incorrect. In detail, the proposed metric would be:

$$\text{correct annotation} = \begin{cases} 1, & \max\{s_{\text{lev}}, s_{\text{raw}}, s_{\text{verbal}}\} \geq \tau, \\ 0, & \max\{s_{\text{lev}}, s_{\text{raw}}, s_{\text{verbal}}\} < \tau \end{cases}$$

Where  $s_{\text{lev}}$  is the normalized Levenshtein similarity,  $s_{\text{raw}}$  is the cosine similarity with raw-input embeddings,  $s_{\text{verbal}}$  is the cosine similarity of verbalized-ontology embeddings, and  $\tau$  is the chosen threshold.

### 3.4 Sustainability and Community

BLINKG is designed as a living, community-driven benchmark rather than a one-off snapshot. All resources (ontologies, datasets, gold standards, evaluation library, and examples) are released in an open repository<sup>20</sup>, so that scenarios and baselines can evolve over time while preserving reproducibility of past results. The benchmark is proposed as method-agnostic: although in the paper we are focused on testing LLM-based approaches, the same scenarios and evaluation pipeline can be used to assess rule-based systems, hybrid pipelines, or human-in-the-loop workflows. We provide detailed documentation, example configurations, to lower the barrier for its adoption.

To foster community engagement, we explicitly invite contributions of new scenarios, baselines, and results through GitHub issues, pull requests, and discussions. We provide dedicated issue templates and guidelines for submitting additional resources, so that contributors can extend

<sup>20</sup><https://github.com/citiususc/blinkg>

BLINKG in a structured and consistent way (e.g., by covering new domains, data formats, levels of schema distance, or mapping subtasks). We also plan to align future iterations of BLINKG with the Knowledge Graph Construction Challenge<sup>21</sup>, where the benchmark can be used in new tracks (e.g., automatic KG Construction) and enriched with more demanding configurations, including scenarios based on synthetic ontologies and input data. This tight integration between an open benchmark, a public repository, and a recurring community event is intended to ensure the long-term sustainability, relevance, and extensibility of BLINKG.

## 4 Evaluation

In this section, we outline the BLINKG evaluation in three steps. First, we introduce the selected LLMs. Next, we detail our evaluation procedure and methodology. Finally, we present and discuss the results.

### 4.1 Selected LLMs

Several promising LLMs have been selected from the state-of-the-art, combining both open-access and fee-based options, as well as reasoning models. This selection was motivated by the use of these models in previous works [16, 22, 23], which show medium-to-high performance, aiming for a balanced representation of both proprietary and open-source LLMs. In all our experiments, we set the LLM temperature to 0.3, as preliminary runs with different values indicated that this configuration provided the best trade-off between stability and diversity in the generated mappings.

**DeepSeek-R1.** DeepSeek-R1 [13], developed by DeepSeek AI in 2025, is a reasoning foundational model trained exclusively through large-scale reinforcement learning, eliminating the need for supervised fine-tuning. This approach allows it to develop strong reasoning and problem-solving skills by learning directly from interactions, optimizing for adaptability and long-term outcomes. Its architecture excels at complex tasks such as logical inference and decision-making.

**Gemini 2.5 Pro.** Gemini 2.5 Pro [48], developed by Google in 2025, is a multimodal foundational model that excels in language understanding, reasoning, and complex task resolution. It integrates text, images, audio, and video data for deeper contextual insights, enabling accurate inferences and efficient problem-solving. Optimized for scalability and adaptability, it leverages self-supervised learning and domain-specific fine-tuning.

**GPT-4 Omni.** GPT-4o [33] is an autoregressive foundational model developed by OpenAI in 2024. It is a multimodal model, capable of processing and generating text, images, and audio within a unified architecture. This model excels in complex problem-solving and contextual understanding. Additionally, it achieves high proficiency in code generation, producing accurate and efficient code across multiple programming languages.

**OpenAI o3.** OpenAI o3 [35] is a reasoning foundational model developed by OpenAI in 2025, as an evolution of OpenAI o1 [34], which was trained with large-scale reinforcement learning to reason using a chain of thought. As a result of this targeted training, GPT-o3 demonstrates advanced reasoning abilities across a wide range of cognitive tasks, including complex problem-solving and logical inference.

---

<sup>21</sup><http://w3id.org/kg-construct/workshop/#challenge>

■ **Table 8** Statistics for the ontology and input data used in each scenario, together with the expected number of task instances in the gold standard.

|              |            | Scenario1 |       |       |       |       |       |       |       | Scenario 2 | Scenario 3       |
|--------------|------------|-----------|-------|-------|-------|-------|-------|-------|-------|------------|------------------|
|              |            | 1A        | 1B    | 1C    | 1D    | 1E    | 1F    | 1G    | 1H    | GTFS       | PPDS             |
| Task         | Class      | 1         | 1     | 2     | 1     | 2     | 2     | 2     | 1     | 12         | 8                |
|              | SubjectGen | 1         | 1     | 2     | 1     | 2     | 2     | 2     | 1     | 10         | 13               |
|              | Prop.      | 1         | 2     | 3     | 4     | 5     | 7     | 6     | 1     | 42         | 16               |
|              | DataRef    | 1         | 2     | 2     | 4     | 5     | 7     | 6     | 3     | 44         | 7                |
|              | Rel.Entity |           |       | 1     |       | 1     | 1     | 1     |       | 10         | 16               |
|              | Joins      |           |       | 1     |       | 1     | 1     | 1     |       | 9          | 16               |
|              | Lang.Tag   |           |       |       |       |       |       | 1     | 2     |            |                  |
|              | Datatype   |           |       |       | 2     |       |       | 3     |       | 14         | 3                |
|              | Functions  |           |       |       |       |       |       |       |       | 10         | 3                |
|              | Total      | 1         | 2     | 3     | 4     | 5     | 7     | 6     | 3     | 72         | 26               |
| OWL Ontology |            | AdHoc     | AdHoc | AdHoc | AdHoc | AdHoc | AdHoc | AdHoc | AdHoc | LinkedGTFS | Subset of ePO    |
| SKOS         |            |           |       |       |       |       |       |       |       | 10         | 3                |
| Input Data   |            | 1 CSV     | 1 CSV | 1 CSV | 1 CSV | 2 CSV | 3 CSV | 2 CSV | 2 CSV | 13 CSV     | CODICE XML Entry |

**LLaMa 3.3 70B Instruct.** Llama 3.3 [18] is an instruction-tuned generative model in 70B released by Meta in 2024. It is an autoregressive language model that uses an optimized transformer architecture. The tuned versions use supervised fine-tuning and reinforcement learning with human feedback to align with human preferences for helpfulness and safety. LLaMa 3.3 70B Instruct is an instruct fine-tuned version of LLaMa 3.3 70B.

**Mixtral 8x22B Instruct.** Mixtral 8x22B [31] is a foundational model developed by Mistral AI. It is a sparse Mixture-of-Experts (SMoE) model that uses only 39B active parameters out of 141B, offering unparalleled cost efficiency for its size. It delivers good performance in complex natural language processing tasks, such as text comprehension, content generation, and advanced reasoning, rivaling larger models while reducing computational costs. Its ability to handle extensive contexts and adapt to various domains makes it a versatile tool. Mixtral 8x22B Instruct is an instruct fine-tuned version of Mixtral 8x22B.

## 4.2 Methodology and Setup

This section describes how BLINKG is used to evaluate the performance of the different LLMs presented in the previous section. It is important to note that our goal is not to compare different prompting strategies or output formats. The primary objective of BLINK is to provide a set of resources and a baseline evaluating general-purpose solutions. The intention is to establish a starting point for a new line of research focused on developing more effective and specialized approaches. The proposed procedure is designed to be generalizable across all benchmark scenarios.

We provide an overview of the benchmark size and configuration in Table 8, which reports, for each scenario, the number of instances per evaluation task, the ontology in use, and the characteristics of the input data. As we detailed in the benchmark description, Scenario 1 consists of eight progressively more complex cases, where we use CSV files as input data; we also tested the same experiments switching to JSON and XML, but we did not find significant variances. Scenario 2 relies on a complete version of the LinkedGTFS ontology and aggregates a total of 72 gold-standard mapping instances across the identified tasks. We use 13 GTFS-CSV files as input, provided as GTFS-1-CSV by GTFS-Madrid-Bench [9]. In order to stay within the LLMs' context-window limits, we keep the full schema (i.e., column names) but only a small, representative sample of 5–10 rows per CSV in this scenario. Scenario 3 is grounded in a CODICE XML entry (see Listing 3) mapped to a subset of ePO, with 26 task instances; here we use the full XML fragment, since data references are expressed as XPath expressions.

The evaluation process begins by constructing a standardized prompt template to ensure consistency across use cases. Each prompt clearly defines the task the LLM is expected to perform and includes a request to output the results in a structured table, with one column per predefined mapping task. This formatting constraint minimizes variability and facilitates direct comparison across models. The prompt also incorporates the data sources associated with the use case. For each source, its filename is included along with either its full content or, if input length becomes a constraint, a representative subset that preserves the original file’s structure and semantics. The ontology and SKOS taxonomies, defined in OWL and RDF respectively, are appended at the end of the prompt to ensure all necessary context is available to the model. This setup is designed to isolate model behavior from prompt formulation artifacts and provide traceable, interpretable outputs. We also evaluate different prompting strategies (zero-shot, one-shot, and few-shot) in the most complex setting (Scenario 3), and providing examples for the various tasks involved. The same prompt is run 3 times.

#### Prompt template

*I have these/this input file(s) [that follow this schema [CODICE/GTFS]] and I want to build a knowledge graph with all possible mappings between them using this ontology. Could you make a table with the data references and a links to the corresponding ontology properties? Link all the properties that you can with the information that you have. Provide also the class of the entities or both the classes that they relate to, and a way to generate the subject of them. Use the following header: Data Reference, Ontology Property, Class Entity, ...*  
 {INPUT\_FILES}  
 ...  
 {ONTOLOGY+SKOS}  
 ...

To cope with the context-window limitations of the LLMs, we had to split Scenarios 2 and 3 into smaller sub-scenarios. Concretely, we isolated groups of ontology entities and built sub-scenarios around them, thereby reducing the number of classes, properties, and input files that needed to be processed in a single prompt. This allowed us to stay within the available token budget while preserving the structure of the original tasks. After running the models on each sub-scenario, we merged the outputs to compute the overall metrics for the full scenario. In practice, Scenario 2 was decomposed into six sub-scenarios, whereas Scenario 3 was split into two.

Before conducting the evaluation of the actual tasks, it is necessary to align each row in the model output with its corresponding row in the gold standard. To achieve this, we apply a two-step matching strategy. The primary approach is to use the ontology property column, which typically contains unique values per row. We compute similarity using the metrics described in Section 3.3 (i.e. Levenshtein distance and embedding-based similarity) to identify the closest match. If this initial matching fails (due to duplicate values or low similarity) we extend the matching criteria by incorporating two additional columns: ontology entity class and data reference. These columns were selected based on an analysis of multiple test cases from Scenario 1, where they consistently exhibited the highest similarity scores and helped disambiguate otherwise uncertain matches. The way of matching rows of tables is inspired by the baselines presented in Pugnaroni et al. [40].

For each of the proposed scenarios, we conducted three different evaluation procedures. These are designed not only to measure the performance of the LLMs on the mapping tasks, but also to assess the effectiveness of the evaluation metrics introduced earlier:

- **Expert evaluation:** Multiple experts in knowledge graph construction manually validate each LLM output to determine whether the identified mappings are correct with respect to the gold standard.
- **Raw evaluation:** The performance of each LLM is assessed using the proposed metrics directly on the raw outputs, without any modification or filtering.
- **Post-processed evaluation:** A set of heuristics is applied to clean the raw outputs. These included typical cleaning steps such as removing extra whitespace, discarding incoherent fragments, and eliminating incorrect columns from unrelated tasks. The goal is to improve the structural consistency of the output and provide a more realistic input to the evaluation metrics.

### 4.3 Results

For each scenario, we report the results in two stages. First, we present the similarity scores to analyze in depth how each LLM performs across the different mapping tasks. Then, we include three separate figures showing the F-score values, one for each evaluation setting (expert-based, raw, and post-processed). Additionally, each scenario is accompanied by a table in Appendix A, detailing the precision, recall, and F-score for every task, LLM, and evaluation type.

To select an appropriate similarity threshold for our proposed metric, we test several values and chose the one that produced results most closely aligned with the expert evaluation. We measured this alignment using Mean Absolute Error (MAE), which quantifies the average absolute difference between the automatically computed metric values and those obtained from expert judgments. In our case, the optimal threshold (minimizing MAE) was found to be 0.8.

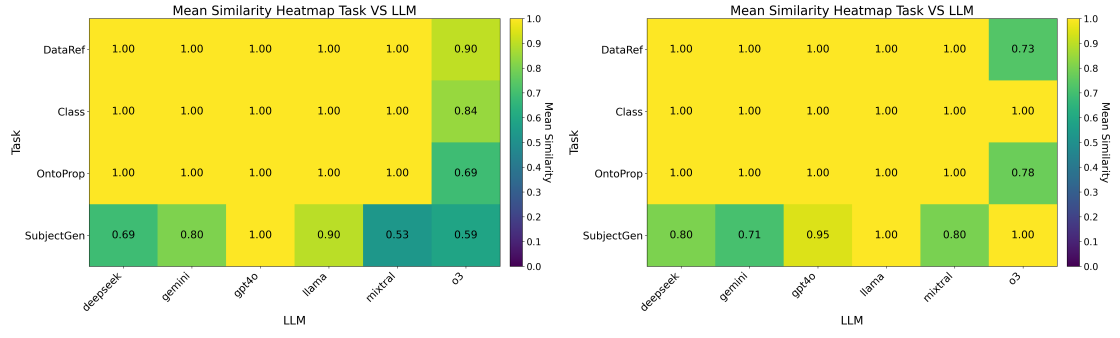
#### Scenario 1

Figure 4 summarizes the average similarity scores across all atomic test cases in Scenario 1, grouped by task and LLM. We only provide the similarity over the post-processed outputs as there are no significant differences between them and the raw ones (see Figure 5). The results show that Entity Class identification consistently yields the highest similarity, with most models achieving values above 0.95, and several reaching perfect alignment with the gold standard. This suggests that LLMs are particularly effective at inferring the class of an entity from tabular data. Ontology Property identification also performs reliably across models, though with slightly more variability than entity classification. Scores remain high overall, indicating that models are able to associate columns with ontology properties in a consistent manner. Related Entity Class shows performance similar to the class detection, though slightly more sensitive to model differences.

Datatype prediction is generally strong for most models ( $\geq 0.89$ ), with the exception of o3, which shows a lower similarity score (0.72), possibly due to inconsistencies in literal typing or formatting in the outputs. In contrast, Join Condition generation remains the most challenging task, with similarity scores ranging from 0.37 (Llama) to 0.69 (Gemini 2.5). This reflects the difficulty LLMs face when inferring logical conditions to connect entities across data sources. Language Annotation exhibits substantial variance between models. Deepseek R1 and Gemini achieve relatively high scores ( $> 0.83$ ), but models like Mixtral and Llama perform below 0.53. This inconsistency suggests that detecting or generating correct language tags is not yet robustly handled by current LLMs. Finally, Data Reference matching shows medium-to-high performance across models. Gemini leads with an average score of 0.88, while GPT4o trails behind at 0.74.

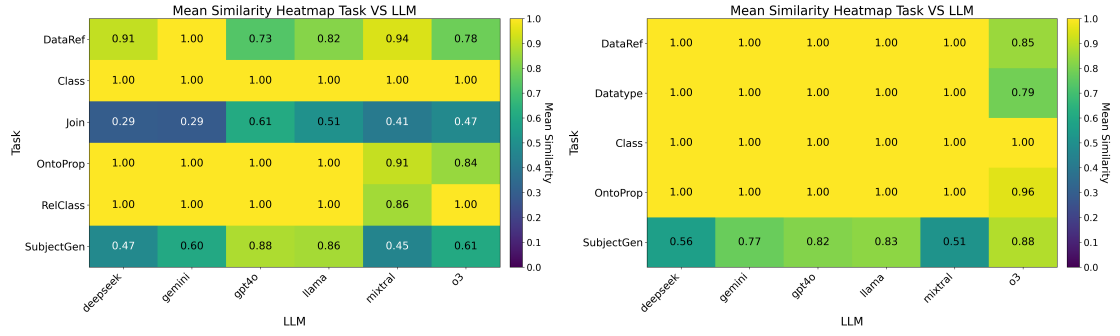
As shown in Figure 5, the obtained F-score in the raw evaluation tends to underestimate model performance, particularly in tasks that are sensitive to formatting or structural correctness, such as Join Condition and Language Annotations. In environments as simple as those proposed in Scenario 1, the differences between raw and post-processed outputs are negligible. This indicates

## 9:22 BLINKG: A Benchmark for LLM-Integrated KG Generation



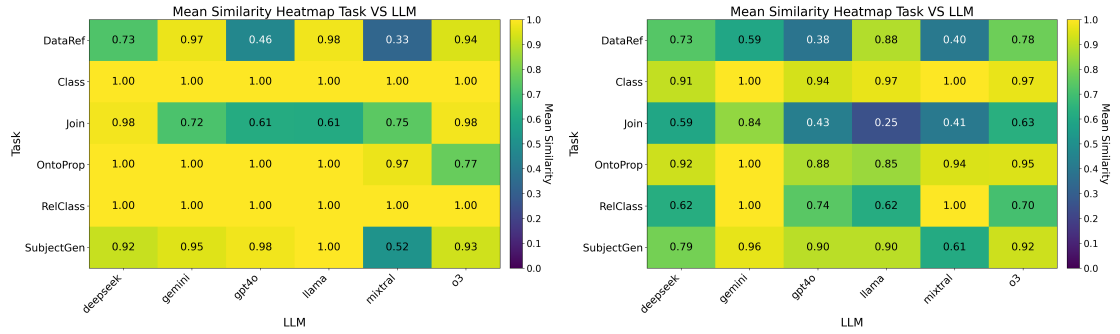
(a) Similarity score for 1A.

(b) Similarity score for 1B.



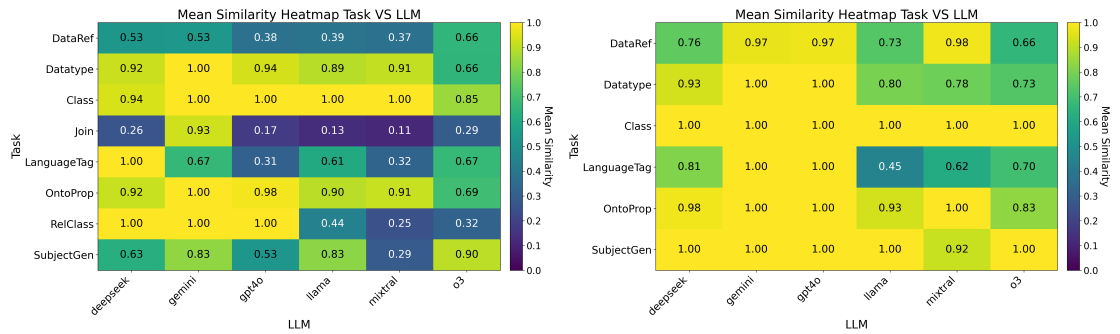
(c) Similarity score for 1C.

(d) Similarity score for 1D.



(e) Similarity score for 1E.

(f) Similarity score for 1F.

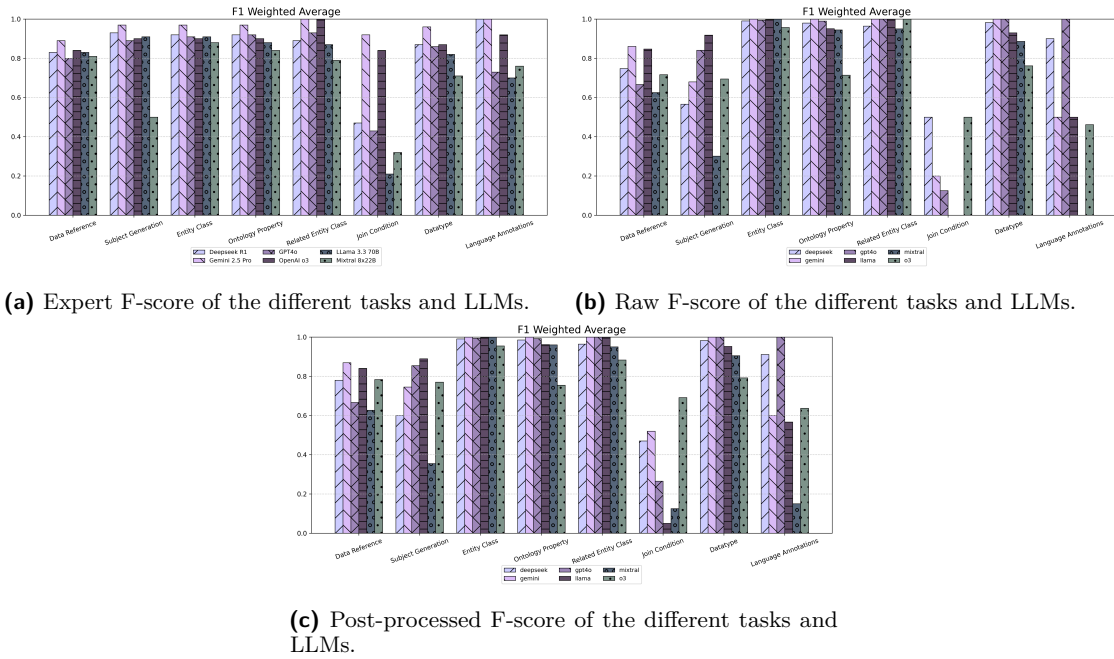


(g) Similarity score for 1G.

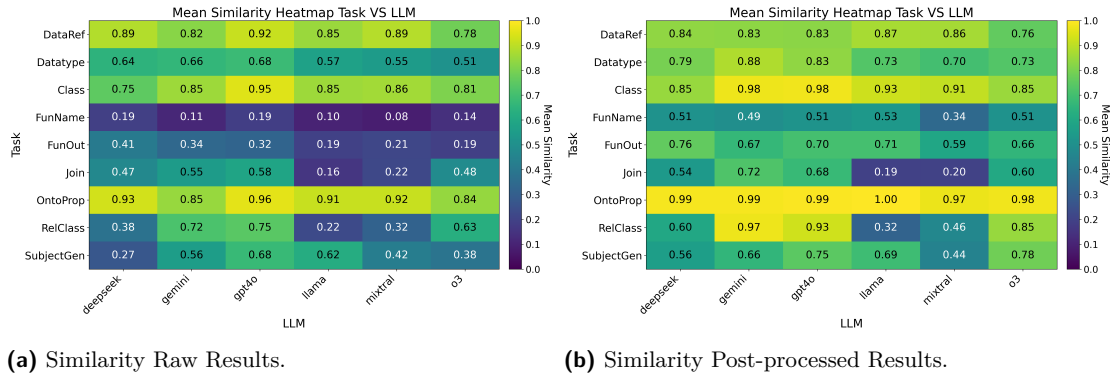
(h) Similarity score for 1H.

**Figure 4** Similarity scores across the eight configurations with respect to the gold standard in the Scenario 1.





**Figure 5** Comparison of the F-score between the three different evaluations in the Scenario 1.

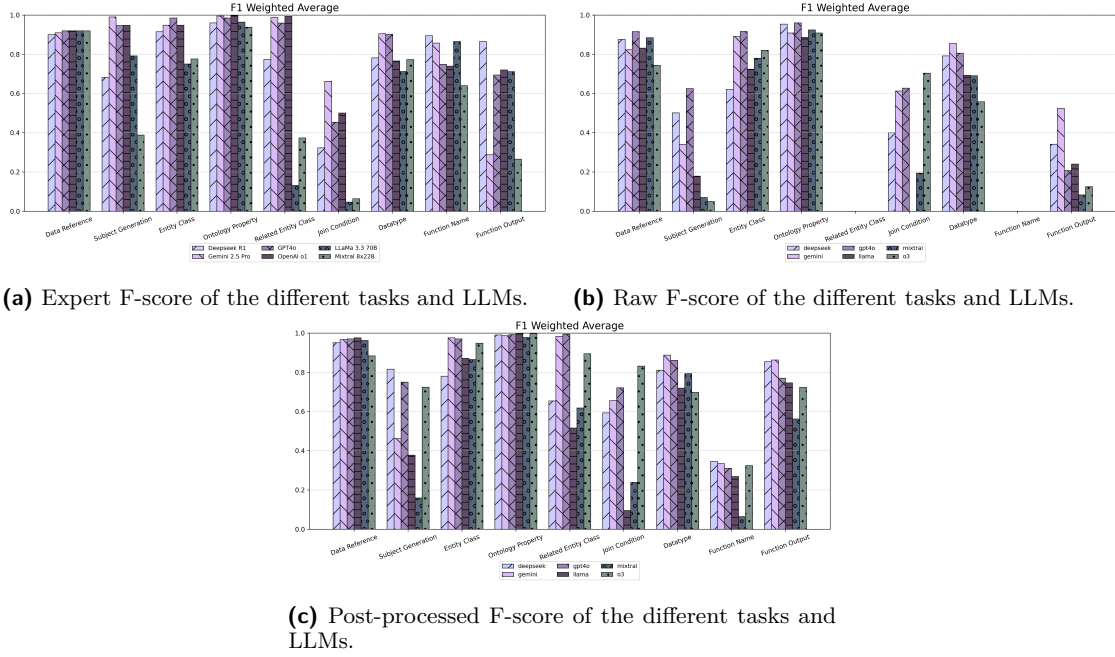


**Figure 6** Comparison of similarity metrics between raw and post-processed configurations with respect to the gold standard in Scenario 2.

that LLMs are capable of closely adhering to the prompt instructions, producing structured outputs with minimal syntactic or formatting noise. As a result, postprocessing has little impact, since most of the relevant information is already correctly and cleanly expressed in the raw outputs.

## Scenario 2

Scenario 2 was designed to test LLMs in tasks that go beyond simple entity or property alignment, requiring them to handle structured data transformations and schema-level logic. As shown in the similarity results (Figure 6), this scenario exposes greater variability across LLMs and highlights significant gaps between raw and post-processed outputs. *Entity Class* and *Data Reference* remain the most reliably handled tasks, with post-processed similarity scores exceeding 0.90 in several models – most notably GPT4o (0.91 in *Data Reference*) and Deepseek (0.95 in *Entity Class*). These tasks are typically guided by strong lexical cues in the input and are less affected by verbosity or syntactic inconsistencies.



**Figure 7** Comparison of the F-score between the three different evaluations in the Scenario 2.

Other concept-matching tasks, such as *Ontology Property* and *Related Entity Class*, also achieve strong performance. For example, Deepseek improves from 0.74 (raw) to 0.80 (post-processed) in *Ontology Property*, while Gemini reaches 0.86 in *Related Entity Class*. This suggests that LLMs can effectively leverage contextual signals to identify relevant ontology terms, as long as surface-level alignment exists and the output is properly cleaned.

In contrast, tasks requiring structural reasoning show much weaker results. *Join Condition*, in particular, remains challenging for all models: even after post-processing, no model exceeds 0.60. The difficulty in inferring relational logic between entities mirrors observations from Scenario 1 and illustrates the persistent gap between language generation capabilities and symbolic reasoning.

Function-related tasks perform worst. Most models score below 0.20 in *Function Name* and below 0.41 in *Function Output*, even after cleaning. These tasks require abstraction, domain-specific function knowledge, and precise formatting – areas where current LLMs often hallucinate, misinterpret, or produce vague and irrelevant output. Similarly, *Subject Generation* remains sensitive to output formatting, but shows moderate improvement with post-processing (e.g., GPT4o rises from 0.64 to 0.71), indicating that some errors are more syntactic than semantic.

F-score scores (Figure 7) align with these observations. Manual annotations consistently yield high results across well-structured tasks, with several models exceeding 0.94 in *Entity Class* and *Subject Generation*. However, raw evaluations often underestimate LLM performance due to noisy output structures. For instance, in *Join Condition*, some raw scores drop by over 0.30 compared to the manual gold standard. Post-processing significantly narrows this gap in most tasks, particularly for *Ontology Property* and *Function Output*, where models like GPT4o and Deepseek improve by 20+ points after cleaning. Nevertheless, difficult tasks like *Function Name* and *Join Condition* still show marked discrepancies, highlighting the limitations of current LLMs and automatic scoring.

Overall, Scenario 2 confirms that while LLMs are increasingly capable in aligning entities and interpreting schema-level concepts when lexical clues are present, they continue to struggle with structural logic and semantic transformation. Post-processing is essential to extract structured meaning from raw outputs, but human-in-the-loop evaluation remains critical to ensure correctness in more abstract or structurally complex mapping tasks.

### Scenario 3

Scenario 3 was designed to challenge LLMs with schema-distant inputs, where the structure and terminology of the source data differ significantly from the target ontology. As shown in the similarity results (Figure 8), this setting reveals critical limitations in the models’ ability to generalize and abstract. Across all tasks, we observe a sharp drop in similarity scores compared to Scenarios 1 and 2, with no task surpassing a mean of 0.53 – even after postprocessing.

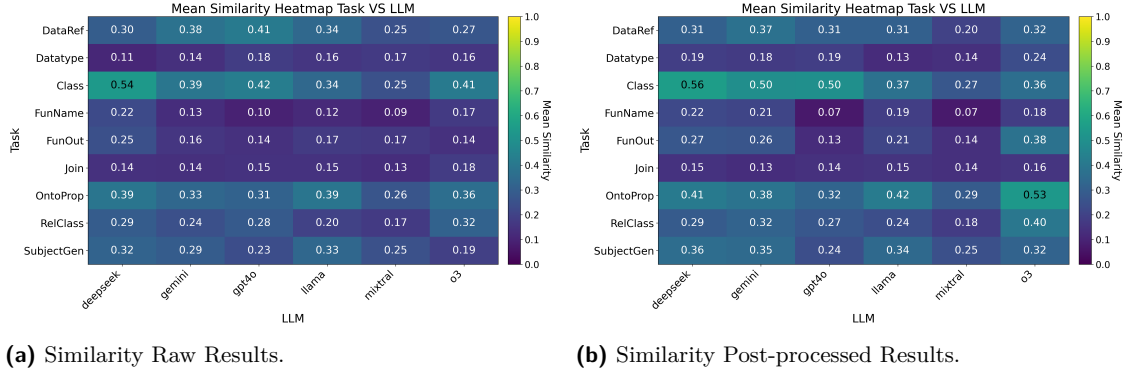
Among all tasks, *Entity Class* remains the best-performing one, with Deepseek achieving 0.53 and GPT4o 0.42. Other concept alignment tasks, such as *Related Entity Class* and *Ontology Property*, yield moderate results (0.35–0.47), but fall significantly short of earlier scenarios. For example, Gemini reaches 0.47 in *Ontology Property*, yet fails to generalize in structurally dependent tasks. This suggests that while some lexical cues may still be leveraged, LLMs struggle to ground predictions in ontology-aware semantics without alignment.

Performance deteriorates further in structurally complex tasks. *Join Condition*, *Datatype*, and *Subject Generation* typically remain below 0.40, even for strong models like GPT4o. Function-related tasks (*Function Name* and *Function Output*) are especially problematic, with all models scoring below 0.25. The models often produce verbose, unrelated operations, failing to extract functional meaning from schema-distant representations.

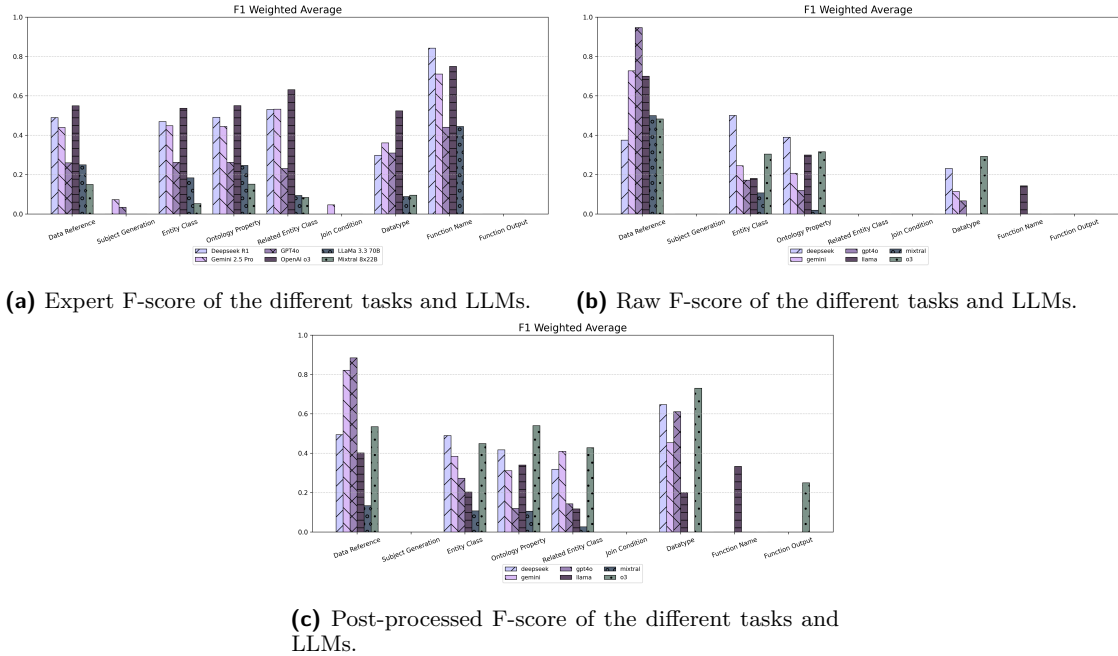
F-score evaluations (Figure 9) reinforce this picture. Manual annotations yield higher scores, especially in concept-level tasks, but even the best-performing models rarely exceed 0.55. Raw evaluations, however, show a marked underestimation of performance due to unstructured outputs, inconsistencies, and hallucinated content. For instance, *Subject Generation* frequently scores near zero in the raw setting, despite partially correct structures. *Join Condition* and *Function Output* also suffer heavily, with most raw scores below 0.10.

Post-processing yields moderate improvements. For example, GPT4o’s score in *Datatype* increases from 0.14 (raw) to 0.31, and similar trends are observed in *Function Name* and *Subject Generation*. This confirms that while relevant information is often present in LLM outputs, superficial noise prevents accurate scoring, and lightweight cleaning helps restore structure. Nonetheless, the gap with manual evaluation remains considerable, especially in tasks requiring symbolic alignment or reasoning.

In Scenario 3, we also analyse the effect of different prompting strategies for DeepSeek. We select DeepSeek as it reports the best performance in the most scenarios. Table 12 shows that, while there are noticeable differences across strategies for tasks such as Data Reference identification, Ontology Property and Entity Class recognition, or Subject Generation, all configurations obtain an F-score of 0 for join-related and function-related tasks. These results highlight that, at least for DeepSeek, changing the prompt (zero-shot, one-shot, or few-shot with examples) helps to a certain extent on “local” tasks that only require selecting classes, properties, or data references, but it does not suffice to trigger more complex reasoning steps such as specifying joins or data transformation functions. This suggests that prompt engineering helps to obtain better results but alone is not enough to build robust mappings for the more compositional aspects of KG Construction.



**Figure 8** Comparison of similarity metrics between raw and post-processed configurations with respect to the gold standard in Scenario 3.



**Figure 9** Comparison of the F-score between the three different evaluations in the Scenario 3.

In summary, Scenario 3 confirms that schema-distant mappings remain largely out of reach for current general-purpose LLMs. While postprocessing helps mitigate output noise, it does not bridge the semantic gap. These findings underscore the need for hybrid approaches – combining LLMs with ontology-aware reasoning components, intermediate representations, or human-in-the-loop validation – to achieve reliable results in such challenging contexts.

## 5 Discussion and Lessons Learned

The BLINKG benchmark reveals both the potential and the limitations of current LLMs when applied to knowledge graph construction through ontology-based mapping generation. Although LLMs demonstrate encouraging results in controlled scenarios, several critical issues emerge when tasks become more complex or less directly aligned with the training distribution of the models.

■ **Table 9** F-score per task for different prompting strategies with DeepSeek in Scenario 3. Join and function-related tasks are not reported, as all strategies obtain an F-score of 0 on these dimensions.

| Method   | Data Ref. | Ont. Prop. | Ent. Class | Rel. Ent. | Subj. Gen. | Datatype |
|----------|-----------|------------|------------|-----------|------------|----------|
| 0-shot   | 0.3750    | 0.3900     | 0.5009     | 0.3852    | 0.0000     | 0.2308   |
| 1-shot   | 0.875     | 0.4167     | 0.3553     | 0.2361    | 0.4444     | 0.3333   |
| few-shot | 0.8636    | 0.3239     | 0.3900     | 0.2222    | 0.7101     | 0.0000   |

Below, we summarize the main findings and observations:

- LLMs perform well in simple scenarios, especially when the input schema and ontology share lexical or structural similarities. However, they tend to hallucinate entities or relationships and produce incorrect outputs as the complexity of the mappings increases, particularly in schema-distant cases.
- Core identification tasks (class, property, related entity) are handled reasonably well, showing low error rates and good semantic alignment with manual annotations. These tasks benefit from contextual cues and general language modeling capabilities.
- More technically grounded tasks, such as joins, datatype assignment, or language tagging, remain challenging, with higher error rates and inconsistencies. These tasks often require strict syntax, explicit constraints, or domain knowledge that LLMs do not reliably internalize.
- The openness of the prompt and input representation has a significant impact on the outcome. Allowing too much flexibility often leads to overly verbose outputs, fabricated mappings, or unintended generalizations. More constrained and structured prompting will tend to yield better results.
- Controlled vocabulary linking is an area where LLMs show unexpected strength. Despite the complexity of these tasks for human annotators, due to the need for specialized knowledge, LLMs are often able to identify the correct concept within a taxonomy when enough context is provided.
- Evaluation metrics are critical for understanding model behavior. General-purpose metrics are useful but insufficient to capture performance nuances across tasks. We observe that task-specific evaluation measures are likely needed to fairly assess LLM performance, especially for cases involving structure, reasoning, or semantic correctness.
- A key limitation of current LLM-based benchmarks, including BLINKG, is the potential impact of memorization and data leakage, since some of the ontologies and datasets we use (i.e., GTFS, ePO) are available on the Web and may partially overlap with the models’ training corpora. Although our results already reveal clear weaknesses in the models’ ability to construct semantic mappings, we acknowledge that a more rigorous disentanglement of genuine reasoning from memorised artefacts is needed [52]. In future iterations of BLINKG should evolve towards the systematic use of synthetic ontologies and input data generated through controlled procedures (i.e. ontology and schema generators), so that the benchmark can better isolate the underlying capabilities of LLMs in KG construction.
- General-purpose LLMs are unlikely to produce valid solutions in highly complex semantic environments. In these cases, hybrid approaches that combine LLM capabilities with symbolic reasoning, such as OWL-based inference or constraint validation, are essential. Techniques like OWL2RML<sup>22</sup> may offer a middle ground between full automation and expert-driven mappings.

<sup>22</sup><https://github.com/citiususc/owl2yarrml>

- Mappings should be characterized not only by task type or schema distance, but also by structural features that may impact their generation, such as star-shaped mappings [25]. Capturing these properties can help isolate factors that affect LLM performance and guide more fine-grained benchmarking.
- LLMs may be valuable for generating initial mapping drafts, but the involvement of a human expert remains essential. A human-in-the-loop approach is crucial for validating mappings, correcting hallucinations, and ensuring that the resulting knowledge graphs are semantically and structurally sound.
- Conducting reproducible and in-depth research with proprietary LLMs poses significant challenges. In our case, some of the models initially used in the evaluation, such as OpenAI’s o1 [34] and Google’s Gemini 2.0 Pro, were removed or restricted on their respective platforms during the course of the evaluation. This forced us to adapt the benchmark execution and repeat the entire evaluation process with the latest models, highlighting the fragility of relying on closed, commercial systems for scientific experimentation.

These observations confirm that, while LLMs represent a promising tool for supporting semantic mapping tasks, their use must be carefully framed within controlled and assisted workflows. BLINKG helps identify where LLMs succeed or fail, and also reveals the importance of hybrid strategies, structured prompting, and task-specific evaluation. As the field advances, we expect that more refined methodologies (combining statistical models with symbolic reasoning and expert oversight) will be key to unlocking the full potential of LLMs in knowledge graph construction.

## 6 Conclusion and Future Work

In this paper, we presented BLINKG, a benchmark specifically designed to evaluate the capabilities of Large Language Models (LLMs) in generating semantic mappings between heterogeneous data sources and ontology terms. Unlike previous efforts, which often focus on end-to-end pipelines or syntactic correctness, BLINKG defines a set of explicit, traceable tasks and provides reusable resources including gold standards, metrics, and diverse scenarios to enable systematic, comparable evaluation across a wide range of conditions. Our experimental results demonstrate that current LLMs are already able to produce promising mappings in well-aligned and controlled scenarios. However, we observe significant limitations when models are faced with schema-distant data, implicit semantics, or complex transformations. In these cases, performance drops notably, particularly in tasks such as conditional joins or function generation. These findings reinforce the need for structured benchmarks like BLINKG to better understand the strengths and weaknesses of LLMs in the context of knowledge graph construction.

Despite its contributions, BLINKG also reveals areas for future improvement. Current evaluation procedures still rely partly on manual inspection or simplified metrics, and handling semantic equivalence beyond syntactic matches remains a challenge. Moreover, the benchmark currently focuses on a core set of tasks; more advanced mapping requirements such as multi-source fusion, list generation, or statement reification are intentionally left for future versions. Finally, as new declarative languages and hybrid AI systems emerge, BLINKG can evolve to include additional modules that evaluate reasoning, transformation chaining, or explainability.

---

## References

- 1 Nora Abdelmageed and Sirko Schindler. Jentab: Matching tabular data to knowledge graphs. In *SemTab@ ISWC*, pages 40–49, 2020. URL: <https://ceur-ws.org/Vol-2775/paper4.pdf>.
- 2 Reham Alharbi, Jacopo de Berardinis, Floriana Grasso, Terry R. Payne, and Valentina Tamma. Characteristics and desiderata for competency question benchmarks. In Reham Alharbi, Jac-



- opo de Berardinis, Paul Groth, Albert Meroño-Peñuela, Elena Simperl, and Valentina Tamma, editors, *Proceedings of the Special Session on Harmonising Generative AI and Semantic Web Technologies (HGAIS 2024) co-located with the 23rd International Semantic Web Conference (ISWC 2024)*, Baltimore, Maryland, November 13, 2024, volume 3953 of *CEUR Workshop Proceedings*, Maryland, USA, November 2024. CEUR-WS.org. URL: <https://ceur-ws.org/Vol-3953/369.pdf>.
- 3 Marcelo Arenas, Alexandre Bertails, Eric Prud'hommeaux, Juan Sequeda, et al. A direct mapping of relational data to rdf. Working group recommendation, World Wide Web Consortium (W3C), September 2012. URL: <http://www.w3.org/TR/rdb-direct-mapping/>.
  - 4 Dylan Van Assche, Thomas Delva, Gerald Haesendonck, Pieter Heyvaert, Ben De Meester, and Anastasia Dimou. Declarative RDF graph generation from heterogeneous (semi-)structured data: A systematic literature review. *J. Web Semant.*, 75:100753, 2023. doi:10.1016/j.websem.2022.100753.
  - 5 Christian Bizer. D2R MAP - A database to RDF mapping language. In Irwin King and Tamás Máray, editors, *Proceedings of the Twelfth International World Wide Web Conference - Posters, WWW 2003, Budapest, Hungary, May 20-24, 2003*, volume 2004. Springer Hiroshima, 2003. URL: <http://www2003.org/cdrom/papers/poster/p004/p4-bizer.html>.
  - 6 Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodríguez-Muro, and Guohui Xiao. Ontop: Answering sparql queries over relational databases. *Semantic Web*, 8(3):471–487, 2016. doi:10.3233/SW-160217.
  - 7 David Chaves-Fraga, Oscar Corcho, Francisco Yedro, Roberto Moreno, Juan Olías, and Alejandro De La Azuela. Systematic construction of knowledge graphs for research-performing organizations. *Information*, 13(12):562, 2022. doi:10.3390/info13120562.
  - 8 David Chaves-Fraga, Kemele M. Endris, Enrique Iglesias, Óscar Corcho, and Maria-Esther Vidal. What are the parameters that affect the construction of a knowledge graph? In Hervé Panetto, Christophe Debruyne, Martin Hepp, Dave Lewis, Claudio Agostino Ardagna, and Robert Meersman, editors, *On the Move to Meaningful Internet Systems: OTM 2019 Conferences - Confederated International Conferences: CoopIS, ODBASE, C&TC 2019, Rhodes, Greece, October 21-25, 2019*, *Proceedings*, volume 11877 of *Lecture Notes in Computer Science*, pages 695–713. Springer, Springer, 2019. doi:10.1007/978-3-030-33246-4\_43.
  - 9 David Chaves-Fraga, Freddy Priyatna, Andrea Cimmino, Jhon Toledo, Edna Ruckhaus, and Óscar Corcho. Gtfs-madrid-bench: A benchmark for virtual knowledge graph access in the transport domain. *J. Web Semant.*, 65:100596, 2020. doi:10.1016/j.websem.2020.100596.
  - 10 Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF Mapping Language. Working group recommendation, World Wide Web Consortium (W3C), September 2012. URL: <http://www.w3.org/TR/r2rml/>.
  - 11 Ioannis Dasoulas, Duo Yang, Xuemin Duan, and Anastasia Dimou. Torchictab: Semantic table annotation with wikidata and language models. In *CEUR Workshop Proceedings*, pages 21–37. CEUR Workshop Proceedings, 2023. URL: <https://ceur-ws.org/Vol-3557/paper2.pdf>.
  - 12 Luciano Frontino de Medeiros, Freddy Priyatna, and Óscar Corcho. MIRROR: automatic R2RML mapping generation from relational databases. In Philipp Cimiano, Flavius Frasincar, Geert-Jan Houben, and Daniel Schwabe, editors, *Engineering the Web in the Big Data Era - 15th International Conference, ICWE 2015, Rotterdam, The Netherlands, June 23-26, 2015*, *Proceedings*, volume 9114 of *Lecture Notes in Computer Science*, pages 326–343. Springer, Springer, 2015. doi:10.1007/978-3-319-19890-3\_21.
  - 13 DeepSeek-AI. Deepseek-R1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. doi:10.48550/arXiv.2501.12948.
  - 14 Anastasia Dimou and David Chaves-Fraga. Declarative description of knowledge graphs construction automation: Status & challenges. In *Proceedings of the 3rd International Workshop on Knowledge Graph Construction (KGCW 2022) co-located with 19th Extended Semantic Web Conference (ESWC 2022)*, volume 3141, 2022. URL: <https://ceur-ws.org/Vol-3141/paper5.pdf>.
  - 15 Nadeen Fathallah, Arunav Das, Stefano De Giorgis, Andrea Poltronieri, Peter Haase, and Liubov Kovriguina. Neon-GPT: a large language model-powered pipeline for ontology learning. In *European Semantic Web Conference*, pages 36–50. Springer, 2024. doi:10.1007/978-3-031-78952-6\_4.
  - 16 Michael Freund, Rene Dorsch, Sebastian Schmid, and Andreas Harth. Mapping by example: Towards an RML mapping reverse engineering pipeline. In David Chaves-Fraga, Ioannis Dasoulas, Christophe Debruyne, Anastasia Dimou, Umutcan Serles, and Dylan Van Assche, editors, *Proceedings of the 6th International Workshop on Knowledge Graph Construction co-located with 22nd Extended Semantic Web Conference (ESWC 2025)*, Portorož, Slovenia, June 1, 2025, volume 3999 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2025. URL: <https://ceur-ws.org/Vol-3999/paper5.pdf>.
  - 17 Daniel Garijo, María Poveda-Villalón, Elvira Amador-Domínguez, Ziyuan Wang, Raúl García-Castro, and Óscar Corcho. Llms for ontology engineering: A landscape of tasks and benchmarking challenges. In Reham Alharbi, Jacopo de Berardinis, Paul Groth, Albert Meroño-Peñuela, Elena Simperl, and Valentina Tamma, editors, *Proceedings of the Special Session on Harmonising Generative AI and Semantic Web Technologies (HGAIS 2024) co-located with the 23rd International Semantic Web Conference (ISWC 2024)*, Baltimore, Maryland, November 13, 2024, volume 3953 of *CEUR Workshop Proceedings*, Maryland, USA, November 2024. CEUR-WS.org. URL: <https://ceur-ws.org/Vol-3953/364.pdf>.



- 18 Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models, 2024. doi:10.48550/arXiv.2407.21783.
- 19 Guntur Budi Herwanto, Stefani Tsaneva, and Marta Sabou. Ontology corpora for llm-based knowledge engineering research. In Reham Alharbi, Jacopo de Berardinis, Paul Groth, Albert Meroño-Peñuela, Elena Simperl, and Valentina Tamma, editors, *Proceedings of the Special Session on Harmonising Generative AI and Semantic Web Technologies (HGAIS 2024) co-located with the 23rd International Semantic Web Conference (ISWC 2024)*, Baltimore, Maryland, November 13, 2024, volume 3953 of *CEUR Workshop Proceedings*, Maryland, USA, November 2024. CEUR-WS.org. URL: <https://ceur-ws.org/Vol-3953/359.pdf>.
- 20 Pieter Heyvaert, David Chaves-Fraga, Freddy Priyatna, Óscar Corcho, Erik Mannens, Ruben Verborgh, and Anastasia Dimou. Conformance test cases for the RDF mapping language (RML). In Boris Villazón-Terrazas and Yusniel Hidalgo-Delgado, editors, *Knowledge Graphs and Semantic Web - First Iberoamerican Conference, KGSWC 2019, Villa Clara, Cuba, June 23-30, 2019, Proceedings*, volume 1029 of *Communications in Computer and Information Science*, pages 162–173. Springer, 2019. doi:10.1007/978-3-030-21395-4\_12.
- 21 Pieter Heyvaert, Ben De Meester, Anastasia Dimou, and Ruben Verborgh. Declarative rules for linked data generation at your fingertips! In *The Semantic Web: ESWC 2018 Satellite Events: ESWC 2018 Satellite Events, Heraklion, Crete, Greece, June 3-7, 2018, Revised Selected Papers 15*, pages 213–217. Springer, 2018. doi:10.1007/978-3-319-98192-5\_40.
- 22 Marvin Hofer, Johannes Frey, and Erhard Rahm. Towards self-configuring knowledge graph construction pipelines using llms - a case study with rml. In *Proceedings of the 5th International Workshop on Knowledge Graph Construction (KGCW 2024) co-located with 21st Extended Semantic Web Conference (ESWC 2024)*, volume 3718, 2024. URL: <https://ceur-ws.org/Vol-3718/paper6.pdf>.
- 23 Lea Hölting, Sven Zentgraf, Philipp Hagedorn, and Markus König. Utilizing large language models for semantic enrichment of infrastructure condition data: a comparative study of gpt and llama models. *AI in Civil Engineering*, 4(1):14, 2025. doi:10.1007/s43503-025-00055-9.
- 24 Viet-Phi Huynh, Jixiong Liu, Yoan Chabot, Frédéric Deuzé, Thomas Labbé, Pierre Monnin, and Raphaël Troncy. DAGOBAB: table and graph contexts for efficient semantic annotation of tabular data. In Ernesto Jiménez-Ruiz, Vasilis Efthymiou, Jiaoyan Chen, Vincenzo Cutrona, Oktie Hassanzadeh, Juan Sequeda, Kavitha Srinivas, Nora Abdelmageed, Madelon Hulsebos, Daniela Oliveira, and Catia Pesquita, editors, *Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching co-located with the 20th International Semantic Web Conference (ISWC 2021)*, Virtual conference, October 27, 2021, volume 3103 of *CEUR Workshop Proceedings*, pages 19–31. CEUR-WS.org, 2021. URL: <https://ceur-ws.org/Vol-3103/paper2.pdf>.
- 25 Enrique Iglesias, Maria-Esther Vidal, Diego Colarana, and David Chaves-Fraga. Empowering the sdm-rdfizer tool for scaling up to complex knowledge graph creation pipelines. *Semantic Web*, 16(2):SW-243580, 2025. doi:10.3233/SW-243580.
- 26 Ana Iglesias-Molina, Andrea Cimmino, Edna Ruckhaus, David Chaves-Fraga, Raúl García-Castro, and Oscar Corcho. An ontological approach for representing declarative mapping languages. *Semantic Web*, 15(1):191–221, 2024. doi:10.3233/SW-223224.
- 27 Ana Iglesias-Molina, Dylan Van Assche, Julián Arenas-Guerrero, Ben De Meester, Christophe Debruyne, Samaneh Jozashoori, Pano Maria, Franck Michel, David Chaves-Fraga, and Anastasia Dimou. The RML ontology: a community-driven modular redesign after a decade of experience in mapping heterogeneous data to RDF. In *International Semantic Web Conference*, pages 152–175. Springer, 2023. doi:10.1007/978-3-031-47243-5\_9.
- 28 Ernesto Jiménez-Ruiz, Oktie Hassanzadeh, Vasilis Efthymiou, Jiaoyan Chen, and Kavitha Srinivas. Semtab 2019: Resources to benchmark tabular data to knowledge graph matching systems. In *The Semantic Web: 17th International Conference, ESWC 2020, Heraklion, Crete, Greece, May 31–June 4, 2020, Proceedings 17*, pages 514–530. Springer, 2020. doi:10.1007/978-3-030-49461-2\_30.
- 29 Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ian Horrocks, Christoph Pinkel, Martin G Skjæveland, Evgenij Thorstensen, and Jose Mora. Bootox: Practical mapping of rdbs to owl 2. In *International Semantic Web Conference*, pages 113–132. Springer, 2015. doi:10.1007/978-3-319-25010-6\_7.
- 30 Jan Maushagen, Sara Sepehri, Audrey Sanctorem, Tamara Vanhaecke, Olga De Troyer, and Christophe Debruyne. Populating CSV files from unstructured text with llms for KG generation with RML. In Daniel Garijo, Anna Lisa Gentile, Anelia Kurteva, Andrea Mannocci, Francesco Osborne, and Sahar Vahdati, editors, *Joint Proceedings of Posters, Demos, Workshops, and Tutorials of the 20th International Conference on Semantic Systems co-located with 20th International Conference on Semantic Systems (SEMANTiCS 2024)*, Amsterdam, The Netherlands, September 17-19, 2024, volume 3759 of *CEUR Workshop Proceedings*. CEUR-WS, CEUR-WS.org, 2024. URL: <https://ceur-ws.org/Vol-3759/paper6.pdf>.
- 31 AI Mistral. Mixtral-8x22b: Cheaper, better, faster, stronger, 2024. URL: <https://mistral.ai/news/mixtral-8x22b>.
- 32 Phuc Nguyen, Natthawut Kertkeidkachorn, Ryutaro Ichise, and Hideaki Takeda. Mtab4d: Semantic annotation of tabular data with dbpedia. *Semantic Web*, 15(6):2613–2637, 2024. doi:10.3233/SW-223098.
- 33 OpenAI. GPT-4o System Card, 2024. doi:10.48550/arXiv.2410.21276.

- 34 OpenAI. OpenAI o1 System Card, 2024. doi: 10.48550/arXiv.2412.16720.
- 35 OpenAI. Openai o3, 2025. URL: <https://platform.openai.com/docs/models/o3>.
- 36 Maria Pano, Anastasia Dimou, Ben De Meester, Ana Iglesias-Molina, David Chaves-Fraga, and Dylan Van Assche. RML-core: Ontology, specification, shapes and test cases, May 2023. Released on 2023-05-10. doi:10.5281/zenodo.7919848.
- 37 Aleksandr Perevalov, Aleksandr Gashkov, Maria Eltsova, and Andreas Both. Understanding SPARQL Queries: Are We Already There? Multilingual Natural Language Generation Based on SPARQL Queries and Large Language Models. In *International Semantic Web Conference*, pages 173–191. Springer, 2024. doi:10.1007/978-3-031-77850-6\_10.
- 38 Christoph Pinkel, Carsten Binnig, Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Wolfgang May, Andriy Nikolov, Ana Sasa Bastinos, Martin G Skjæveland, Alessandro Solimando, Mohsen Taheriyani, et al. Rodi: Benchmarking relational-to-ontology mapping generation quality. *Semantic Web*, 9(1):25–52, 2017. doi:10.3233/SW-170268.
- 39 Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. In *Journal on data semantics X*, pages 133–173. Springer, 2008. doi:10.1007/978-3-540-77688-8\_5.
- 40 Francesco Pugnali, Luca Zecchini, Matteo Paganelli, Matteo Lissandrini, Felix Naumann, and Giovanni Simonini. Table Overlap Estimation through Graph Embeddings. *Proceedings of the ACM on Management of Data*, 3(3):1–25, 2025. doi:10.1145/3725365.
- 41 Filip Radulovic, María Poveda-Villalón, Daniel Vila-Suero, Víctor Rodríguez-Doncel, Raúl García-Castro, and Asunción Gómez-Pérez. Guidelines for linked data generation and publication: An example in building energy consumption. *Automation in Construction*, 57:178–187, 2015. doi: 10.1016/j.autcon.2015.04.002.
- 42 Alex Randles and Declan O’Sullivan. R2 [RML]-ChatGPT Framework. In *Proceedings of the 5th International Workshop on Knowledge Graph Construction (KGCW 2024) co-located with 21st Extended Semantic Web Conference (ESWC 2024)*, volume 3718, 2024. URL: <https://ceur-ws.org/Vol-3718/paper5.pdf>.
- 43 Youssra Rebboud, Pasquale Lisena, Lionel Tailhardat, and Raphaël Troncy. Benchmarking llm-based ontology conceptualization: A proposal. In Reham Alharbi, Jacopo de Berardinis, Paul Groth, Albert Meroño-Peñuela, Elena Simperl, and Valentina Tamma, editors, *Proceedings of the Special Session on Harmonising Generative AI and Semantic Web Technologies (HGAIS 2024) co-located with the 23rd International Semantic Web Conference (ISWC 2024)*, Baltimore, Maryland, November 13, 2024, volume 3953 of *CEUR Workshop Proceedings*, Maryland, USA, November 2024. CEUR-WS.org. URL: <https://ceur-ws.org/Vol-3953/365.pdf>.
- 44 Youssra Rebboud, Lionel Tailhardat, Pasquale Lisena, and Raphael Troncy. Can LLMs Generate Competency Questions? In *European Semantic Web Conference*, pages 71–80. Springer, 2024. doi:10.1007/978-3-031-78952-6\_7.
- 45 Julián Andrés Rojas, Marina Aguado, Polymnia Vasilopoulou, Ivo Velitchkov, Dylan Van Assche, Pieter Colpaert, and Ruben Verborgh. Leveraging semantic technologies for digital interoperability in the european railway domain. In *International Semantic Web Conference*, pages 648–664. Springer, 2021. doi:10.1007/978-3-030-88361-4\_38.
- 46 Wilma Johanna Schmidt, Irlán Grangel-González, Tobias Huschle, Lena Wagner, Evgeny Kharlamov, and Adrian Paschke. Llm-supported mapping generation for semantic manufacturing treasure hunting. In Edward Curry, Maribel Acosta, María Poveda-Villalón, Marieke van Erp, Adegboyega K. Ojo, Katja Hose, Cogan Shimizu, and Pasquale Lisena, editors, *The Semantic Web - 22nd European Semantic Web Conference, ESWC 2025, Portoroz, Slovenia, June 1-5, 2025, Proceedings, Part II*, volume 15719 of *Lecture Notes in Computer Science*, pages 84–101. Springer, Springer, 2025. doi: 10.1007/978-3-031-94578-6\_5.
- 47 Álvaro Sicilia and German Nemirovski. AutoMap4OBDA: Automated generation of R2RML mappings for OBDA. In *European Knowledge Acquisition Workshop*, pages 577–592. Springer, 2016. doi:10.1007/978-3-319-49004-5\_37.
- 48 Gemini Team. Gemini: A family of highly capable multimodal models, 2024. doi:10.48550/arXiv.2312.11805.
- 49 Stefani Tsaneva, Guntur Budi Herwanto, and Marta Sabou. Benchmarking ontology validation capabilities of llms. In Reham Alharbi, Jacopo de Berardinis, Paul Groth, Albert Meroño-Peñuela, Elena Simperl, and Valentina Tamma, editors, *Proceedings of the Special Session on Harmonising Generative AI and Semantic Web Technologies (HGAIS 2024) co-located with the 23rd International Semantic Web Conference (ISWC 2024)*, Baltimore, Maryland, November 13, 2024, volume 3953 of *CEUR Workshop Proceedings*, Maryland, USA, November 2024. CEUR-WS.org. URL: <https://ceur-ws.org/Vol-3953/358.pdf>.
- 50 Mikel Val-Calvo, Mikel Egana Aranguren, Juan Mulero-Hernandez, Gines Almagro-Hernandez, Prashant Deshmukh, José Antonio Bernabé-Díaz, Paola Espinoza-Arias, José Luis Sánchez-Fernández, Juergen Mueller, and Jesualdo Tomas Fernandez-Breis. Ontogenix: Leveraging large language models for enhanced ontology engineering from datasets. *Information Processing & Management*, 62(3):104042, 2025. doi:10.1016/j.ipm.2024.104042.
- 51 Guohui Xiao, Diego Calvanese, Roman Kontchakov, Domenico Lembo, Antonella Poggi, Riccardo Rosati, and Michael Zakharyashev. Ontology-based data access: A survey. In *Proc. 27th International Joint Conference on Artificial Intelligence*, pages 5511–5519. International Joint Conferences on Artificial Intelligence, 2018. doi:10.24963/ijcai.2018/777.
- 52 Zhangyue Yin, Qiushi Sun, Qipeng Guo, Jiawen Wu, Xipeng Qiu, and Xuanjing Huang. Do large language models know what they don’t

- know? In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8653–8665, Toronto, Canada, July 2023. Association for Computational Linguistics. doi:10.18653/v1/2023.findings-acl.551.
- 53 Bohui Zhang, Valentina Anita Carriero, Katrin Schreiberhuber, Stefani Tsaneva, Lucía Sánchez González, Jongmo Kim, and Jacopo de Berardinis. OntoChat: a framework for conversational ontology engineering using language models. In *European Semantic Web Conference*, pages 102–121. Springer, 2024. doi:10.1007/978-3-031-78952-6\_10.

## A Detailed evaluations of the different scenarios

The Mean Absolute Error of F-score between expert evaluation and Post-processed in the Scenario 1 are Deepseek: 0.106, Gemini: 0.166, GPT4o: 0.133, o3: 0.169, Llama: 0.123 and Mixtral: 0.236.

**Table 10** Evaluation metrics (Precision, Recall, F1-score) for Scenario 1 across all models and evaluation types.

|                             | deepseek |      |      | gemini |      |      | gpt4o  |      |      | o3     |      |      | llama  |      |      | mixtral |      |      |
|-----------------------------|----------|------|------|--------|------|------|--------|------|------|--------|------|------|--------|------|------|---------|------|------|
|                             | Manual   | Raw  | Post | Manual | Raw  | Post | Manual | Raw  | Post | Manual | Raw  | Post | Manual | Raw  | Post | Manual  | Raw  | Post |
| <i>Data Reference</i>       |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.84     | 0.88 | 0.88 | 0.92   | 1.00 | 1.00 | 0.81   | 0.75 | 0.75 | 0.76   | 1.00 | 0.88 | 0.84   | 0.88 | 0.63 | 0.80    | 0.63 | 1.00 |
| R                           | 0.82     | 0.69 | 0.69 | 0.87   | 0.82 | 0.83 | 0.80   | 0.65 | 0.65 | 0.94   | 0.58 | 0.59 | 0.82   | 0.83 | 0.83 | 0.82    | 0.63 | 0.63 |
| F1                          | 0.83     | 0.77 | 0.77 | 0.89   | 0.86 | 0.91 | 0.80   | 0.67 | 0.70 | 0.84   | 0.72 | 0.71 | 0.83   | 0.85 | 0.71 | 0.81    | 0.63 | 0.77 |
| <i>Ontology Property</i>    |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.94     | 1.00 | 1.00 | 1.00   | 1.00 | 1.00 | 0.94   | 1.00 | 1.00 | 0.82   | 1.00 | 1.00 | 0.90   | 1.00 | 1.00 | 0.81    | 1.00 | 1.00 |
| R                           | 0.91     | 0.97 | 0.96 | 0.95   | 1.00 | 1.00 | 0.90   | 1.00 | 0.98 | 1.00   | 0.62 | 0.58 | 0.86   | 0.88 | 0.91 | 0.87    | 0.82 | 0.91 |
| F1                          | 0.92     | 0.98 | 0.98 | 0.97   | 1.00 | 1.00 | 0.92   | 1.00 | 0.99 | 0.90   | 0.76 | 0.74 | 0.88   | 0.93 | 0.95 | 0.84    | 0.89 | 0.95 |
| <i>Entity Class</i>         |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.94     | 1.00 | 1.00 | 1.00   | 1.00 | 1.00 | 0.92   | 1.00 | 1.00 | 0.82   | 1.00 | 1.00 | 0.93   | 1.00 | 1.00 | 0.86    | 1.00 | 1.00 |
| R                           | 0.91     | 0.98 | 0.98 | 0.95   | 1.00 | 1.00 | 0.90   | 0.99 | 0.99 | 1.00   | 0.93 | 0.92 | 0.89   | 0.99 | 0.99 | 0.90    | 1.00 | 1.00 |
| F1                          | 0.92     | 0.99 | 0.99 | 0.97   | 1.00 | 1.00 | 0.91   | 0.99 | 0.99 | 0.90   | 0.96 | 0.96 | 0.91   | 1.00 | 1.00 | 0.88    | 1.00 | 1.00 |
| <i>Related Entity Class</i> |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.80     | 1.00 | 1.00 | 1.00   | 1.00 | 1.00 | 0.87   | 1.00 | 1.00 | 1.00   | 1.00 | 1.00 | 0.78   | 1.00 | 1.00 | 0.87    | 1.00 | 1.00 |
| R                           | 1.00     | 0.94 | 0.94 | 1.00   | 1.00 | 1.00 | 1.00   | 1.00 | 1.00 | 1.00   | 1.00 | 0.79 | 1.00   | 1.00 | 1.00 | 0.75    | 0.92 | 0.92 |
| F1                          | 0.89     | 0.96 | 0.97 | 1.00   | 1.00 | 1.00 | 0.93   | 1.00 | 1.00 | 1.00   | 1.00 | 0.88 | 0.87   | 1.00 | 1.00 | 0.79    | 0.95 | 0.96 |
| <i>Subject Generation</i>   |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.96     | 0.75 | 0.75 | 1.00   | 1.00 | 1.00 | 0.90   | 1.00 | 1.00 | 0.82   | 0.75 | 0.75 | 0.93   | 1.00 | 1.00 | 0.50    | 0.50 | 0.50 |
| R                           | 0.91     | 0.47 | 0.47 | 0.94   | 0.58 | 0.58 | 0.88   | 0.77 | 0.77 | 1.00   | 0.65 | 0.68 | 0.89   | 0.87 | 0.87 | 0.49    | 0.25 | 0.25 |
| F1                          | 0.93     | 0.57 | 0.58 | 0.97   | 0.68 | 0.73 | 0.89   | 0.84 | 0.87 | 0.90   | 0.69 | 0.72 | 0.91   | 0.92 | 0.93 | 0.50    | 0.30 | 0.34 |
| <i>Join Condition</i>       |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.40     | 0.50 | 0.50 | 0.92   | 0.25 | 0.50 | 0.32   | 0.25 | 0.25 | 0.79   | 0.50 | 0.75 | 0.15   | 0.00 | 0.00 | 0.33    | 0.00 | 0.00 |
| R                           | 0.58     | 0.50 | 0.50 | 0.92   | 0.17 | 0.42 | 0.67   | 0.08 | 0.08 | 1.00   | 0.50 | 0.63 | 0.33   | 0.00 | 0.00 | 0.33    | 0.00 | 0.00 |
| F1                          | 0.47     | 0.50 | 0.50 | 0.92   | 0.20 | 0.45 | 0.43   | 0.13 | 0.12 | 0.84   | 0.50 | 0.68 | 0.21   | 0.00 | 0.00 | 0.32    | 0.00 | 0.00 |
| <i>Datatype</i>             |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.92     | 1.00 | 1.00 | 1.00   | 1.00 | 1.00 | 0.89   | 1.00 | 1.00 | 0.77   | 1.00 | 1.00 | 0.82   | 1.00 | 1.00 | 0.71    | 1.00 | 1.00 |
| R                           | 0.83     | 0.96 | 0.97 | 0.92   | 1.00 | 1.00 | 0.83   | 0.98 | 1.00 | 1.00   | 0.59 | 0.65 | 0.83   | 0.91 | 0.92 | 0.72    | 0.91 | 0.85 |
| F1                          | 0.87     | 0.98 | 0.98 | 0.96   | 1.00 | 1.00 | 0.86   | 0.99 | 1.00 | 0.87   | 0.71 | 0.79 | 0.82   | 0.95 | 0.96 | 0.71    | 0.94 | 0.92 |
| <i>Language Annotations</i> |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 1.00     | 1.00 | 1.00 | 1.00   | 0.50 | 0.50 | 0.81   | 1.00 | 1.00 | 0.87   | 0.50 | 0.50 | 0.58   | 0.50 | 0.50 | 0.83    | 0.00 | 0.00 |
| R                           | 1.00     | 0.83 | 0.83 | 1.00   | 0.50 | 0.50 | 0.78   | 1.00 | 1.00 | 1.00   | 0.43 | 0.43 | 1.00   | 0.50 | 0.50 | 0.78    | 0.00 | 0.00 |
| F1                          | 1.00     | 0.90 | 0.91 | 1.00   | 0.50 | 0.50 | 0.73   | 1.00 | 1.00 | 0.92   | 0.46 | 0.46 | 0.70   | 0.50 | 0.50 | 0.76    | 0.00 | 0.00 |

## 9:34 BLINKG: A Benchmark for LLM-Integrated KG Generation

The Mean Absolute Error of F-score between expert evaluation and Post-processed in the Scenario 2 are Deepseek: 0.16, Gemini: 0.22, GPT4o: 0.13, o3: 0.21, Llama: 0.21 and Mixtral: 0.22.

■ **Table 11** Evaluation metrics (Precision, Recall, F1-score) for Scenario 2 across all models and evaluation types.

|                             | deepseek |      |      | gemini |      |      | gpt4o  |      |      | o3     |      |      | llama  |      |      | mixtral |      |      |
|-----------------------------|----------|------|------|--------|------|------|--------|------|------|--------|------|------|--------|------|------|---------|------|------|
|                             | Manual   | Raw  | Post | Manual | Raw  | Post | Manual | Raw  | Post | Manual | Raw  | Post | Manual | Raw  | Post | Manual  | Raw  | Post |
| <i>Data Reference</i>       |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.85     | 1.00 | 1.00 | 0.84   | 1.00 | 1.00 | 0.85   | 1.00 | 1.00 | 0.84   | 1.00 | 1.00 | 0.88   | 1.00 | 1.00 | 0.90    | 1.00 | 1.00 |
| R                           | 0.96     | 0.80 | 0.91 | 1.00   | 0.72 | 0.95 | 1.00   | 0.86 | 0.95 | 0.94   | 0.63 | 0.79 | 0.96   | 0.76 | 0.96 | 0.94    | 0.82 | 0.94 |
| F1                          | 0.90     | 0.88 | 0.95 | 0.91   | 0.82 | 0.97 | 0.92   | 0.92 | 0.97 | 0.92   | 0.74 | 0.86 | 0.92   | 0.83 | 0.98 | 0.92    | 0.88 | 0.96 |
| <i>Ontology Property</i>    |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.98     | 1.00 | 1.00 | 0.99   | 1.00 | 1.00 | 0.99   | 1.00 | 1.00 | 1.00   | 1.00 | 1.00 | 1.00   | 1.00 | 1.00 | 0.98    | 1.00 | 1.00 |
| R                           | 0.94     | 0.92 | 0.98 | 1.00   | 0.84 | 0.98 | 0.99   | 0.93 | 0.99 | 1.00   | 0.84 | 1.00 | 0.93   | 0.83 | 1.00 | 0.90    | 0.87 | 0.95 |
| F1                          | 0.96     | 0.95 | 0.99 | 1.00   | 0.91 | 0.99 | 0.99   | 0.96 | 0.99 | 1.00   | 0.91 | 1.00 | 0.96   | 0.89 | 1.00 | 0.94    | 0.92 | 0.98 |
| <i>Entity Class</i>         |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.93     | 0.83 | 1.00 | 0.95   | 1.00 | 1.00 | 0.99   | 1.00 | 1.00 | 0.95   | 1.00 | 1.00 | 0.78   | 1.00 | 1.00 | 0.82    | 1.00 | 1.00 |
| R                           | 0.90     | 0.57 | 0.65 | 0.95   | 0.82 | 0.95 | 0.99   | 0.86 | 0.94 | 0.95   | 0.73 | 0.90 | 0.73   | 0.66 | 0.79 | 0.74    | 0.71 | 0.77 |
| F1                          | 0.91     | 0.62 | 0.73 | 0.95   | 0.89 | 0.97 | 0.99   | 0.92 | 0.96 | 0.95   | 0.82 | 0.93 | 0.75   | 0.72 | 0.83 | 0.78    | 0.78 | 0.82 |
| <i>Related Entity Class</i> |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.84     | 0.67 | 0.83 | 0.98   | 0.83 | 1.00 | 1.00   | 0.83 | 1.00 | 1.00   | 0.33 | 0.50 | 0.12   | 0.50 | 0.67 | 0.41    | 0.83 | 1.00 |
| R                           | 0.73     | 0.38 | 0.54 | 1.00   | 0.81 | 0.97 | 0.92   | 0.79 | 0.98 | 0.99   | 0.32 | 0.50 | 0.14   | 0.38 | 0.59 | 0.35    | 0.66 | 0.82 |
| F1                          | 0.77     | 0.47 | 0.63 | 0.99   | 0.82 | 0.98 | 0.96   | 0.81 | 0.99 | 0.99   | 0.32 | 0.50 | 0.13   | 0.43 | 0.62 | 0.37    | 0.72 | 0.88 |
| <i>Subject Generation</i>   |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.95     | 1.00 | 1.00 | 0.99   | 0.83 | 0.83 | 0.97   | 1.00 | 1.00 | 0.95   | 0.17 | 0.83 | 0.82   | 0.33 | 0.33 | 0.45    | 0.17 | 0.17 |
| R                           | 0.53     | 0.37 | 0.68 | 1.00   | 0.23 | 0.30 | 0.93   | 0.51 | 0.63 | 0.95   | 0.03 | 0.59 | 0.76   | 0.13 | 0.22 | 0.34    | 0.04 | 0.06 |
| F1                          | 0.68     | 0.50 | 0.75 | 0.99   | 0.34 | 0.42 | 0.95   | 0.62 | 0.71 | 0.95   | 0.05 | 0.67 | 0.79   | 0.18 | 0.27 | 0.39    | 0.07 | 0.08 |
| <i>Join Condition</i>       |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.23     | 0.50 | 0.67 | 0.61   | 0.67 | 0.67 | 0.36   | 0.83 | 0.83 | 0.38   | 0.83 | 1.00 | 0.03   | 0.00 | 0.00 | 0.05    | 0.33 | 0.33 |
| R                           | 0.53     | 0.33 | 0.50 | 0.73   | 0.57 | 0.59 | 0.70   | 0.53 | 0.56 | 0.80   | 0.64 | 0.79 | 0.13   | 0.00 | 0.00 | 0.10    | 0.14 | 0.14 |
| F1                          | 0.32     | 0.40 | 0.55 | 0.66   | 0.61 | 0.62 | 0.45   | 0.63 | 0.65 | 0.50   | 0.70 | 0.85 | 0.05   | 0.00 | 0.00 | 0.06    | 0.19 | 0.19 |
| <i>Datatype</i>             |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.79     | 1.00 | 1.00 | 0.90   | 1.00 | 1.00 | 0.89   | 1.00 | 1.00 | 0.76   | 1.00 | 1.00 | 0.74   | 0.83 | 0.83 | 0.84    | 1.00 | 1.00 |
| R                           | 0.77     | 0.67 | 0.70 | 0.90   | 0.77 | 0.82 | 0.91   | 0.74 | 0.79 | 0.77   | 0.40 | 0.52 | 0.69   | 0.63 | 0.64 | 0.72    | 0.59 | 0.70 |
| F1                          | 0.78     | 0.79 | 0.81 | 0.90   | 0.86 | 0.89 | 0.90   | 0.81 | 0.86 | 0.77   | 0.56 | 0.68 | 0.71   | 0.69 | 0.72 | 0.77    | 0.69 | 0.80 |
| <i>Function Name</i>        |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 1.00     | 0.00 | 0.20 | 0.98   | 0.00 | 0.20 | 0.86   | 0.00 | 0.20 | 0.71   | 0.00 | 0.20 | 1.00   | 0.00 | 0.20 | 0.80    | 0.00 | 0.00 |
| R                           | 0.81     | 0.00 | 0.20 | 0.76   | 0.00 | 0.20 | 0.67   | 0.00 | 0.20 | 0.78   | 0.00 | 0.20 | 0.76   | 0.00 | 0.20 | 0.56    | 0.00 | 0.00 |
| F1                          | 0.89     | 0.00 | 0.20 | 0.86   | 0.00 | 0.20 | 0.75   | 0.00 | 0.20 | 0.74   | 0.00 | 0.20 | 0.86   | 0.00 | 0.20 | 0.64    | 0.00 | 0.00 |
| <i>Function Output</i>      |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.93     | 0.67 | 1.00 | 0.33   | 0.83 | 1.00 | 0.75   | 0.33 | 1.00 | 0.75   | 0.33 | 1.00 | 0.76   | 0.33 | 0.80 | 0.30    | 0.17 | 1.00 |
| R                           | 0.81     | 0.27 | 0.79 | 0.25   | 0.43 | 0.80 | 0.65   | 0.18 | 0.63 | 0.70   | 0.08 | 0.58 | 0.67   | 0.19 | 0.68 | 0.24    | 0.06 | 0.35 |
| F1                          | 0.86     | 0.34 | 0.86 | 0.29   | 0.52 | 0.86 | 0.69   | 0.21 | 0.73 | 0.72   | 0.13 | 0.67 | 0.71   | 0.24 | 0.73 | 0.26    | 0.08 | 0.49 |

The Mean Absolute Error of F-score between expert evaluation and Post-processed in the Scenario 3 are Deepseek: 0.42, Gemini: 0.38, GPT4o: 0.35, o3: 0.32, LLaMa: 0.35, Mixtral: 0.34

■ **Table 12** Evaluation metrics (Precision, Recall, F1-score) for Scenario 3 across all models and evaluation types.

|                             | deepseek |      |      | gemini |      |      | gpt4o  |      |      | o3     |      |      | llama  |      |      | mixtral |      |      |
|-----------------------------|----------|------|------|--------|------|------|--------|------|------|--------|------|------|--------|------|------|---------|------|------|
|                             | Manual   | Raw  | Post | Manual | Raw  | Post | Manual | Raw  | Post | Manual | Raw  | Post | Manual | Raw  | Post | Manual  | Raw  | Post |
| <i>Data Reference</i>       |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.05     | 1.00 | 1.00 | 0.07   | 1.00 | 1.00 | 0.08   | 1.00 | 1.00 | 0.02   | 1.00 | 1.00 | 0.02   | 1.00 | 1.00 | 0.00    | 0.50 | 0.50 |
| R                           | 0.05     | 0.24 | 0.33 | 0.10   | 0.64 | 0.70 | 0.06   | 0.84 | 0.79 | 0.03   | 0.24 | 0.40 | 0.03   | 0.31 | 0.29 | 0.00    | 0.17 | 0.08 |
| F1                          | 0.05     | 0.39 | 0.49 | 0.09   | 0.78 | 0.82 | 0.07   | 0.91 | 0.88 | 0.02   | 0.38 | 0.54 | 0.02   | 0.42 | 0.40 | 0.00    | 0.25 | 0.13 |
| <i>Ontology Property</i>    |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.42     | 0.50 | 1.00 | 0.39   | 0.50 | 1.00 | 0.34   | 0.50 | 1.00 | 0.53   | 1.00 | 1.00 | 0.21   | 0.00 | 0.50 | 0.15    | 0.00 | 0.00 |
| R                           | 0.59     | 0.15 | 0.51 | 0.53   | 0.06 | 0.31 | 0.22   | 0.04 | 0.56 | 0.58   | 0.17 | 0.60 | 0.31   | 0.00 | 0.13 | 0.15    | 0.00 | 0.00 |
| F1                          | 0.49     | 0.23 | 0.65 | 0.44   | 0.11 | 0.45 | 0.26   | 0.07 | 0.61 | 0.53   | 0.29 | 0.73 | 0.25   | 0.00 | 0.20 | 0.15    | 0.00 | 0.00 |
| <i>Entity Class</i>         |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.40     | 1.00 | 1.00 | 0.39   | 1.00 | 1.00 | 0.34   | 1.00 | 1.00 | 0.51   | 1.00 | 1.00 | 0.16   | 1.00 | 1.00 | 0.05    | 1.00 | 1.00 |
| R                           | 0.56     | 0.33 | 0.32 | 0.54   | 0.14 | 0.25 | 0.22   | 0.09 | 0.16 | 0.56   | 0.18 | 0.30 | 0.23   | 0.10 | 0.11 | 0.05    | 0.06 | 0.06 |
| F1                          | 0.47     | 0.50 | 0.49 | 0.45   | 0.25 | 0.38 | 0.26   | 0.17 | 0.27 | 0.54   | 0.30 | 0.45 | 0.18   | 0.18 | 0.20 | 0.05    | 0.11 | 0.11 |
| <i>Related Entity Class</i> |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.52     | 1.00 | 1.00 | 0.51   | 1.00 | 1.00 | 0.31   | 0.50 | 0.50 | 0.60   | 1.00 | 1.00 | 0.08   | 0.50 | 0.50 | 0.07    | 0.00 | 0.50 |
| R                           | 0.54     | 0.24 | 0.19 | 0.56   | 0.16 | 0.27 | 0.19   | 0.11 | 0.08 | 0.67   | 0.19 | 0.30 | 0.13   | 0.05 | 0.07 | 0.10    | 0.00 | 0.01 |
| F1                          | 0.53     | 0.39 | 0.32 | 0.53   | 0.27 | 0.41 | 0.23   | 0.18 | 0.14 | 0.63   | 0.31 | 0.43 | 0.09   | 0.09 | 0.12 | 0.08    | 0.00 | 0.03 |
| <i>Subject Generation</i>   |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.00     | 0.00 | 0.00 | 0.07   | 0.00 | 0.00 | 0.05   | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00    | 0.00 | 0.00 |
| R                           | 0.00     | 0.00 | 0.00 | 0.08   | 0.00 | 0.00 | 0.03   | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00    | 0.00 | 0.00 |
| F1                          | 0.00     | 0.00 | 0.00 | 0.07   | 0.00 | 0.00 | 0.03   | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00    | 0.00 | 0.00 |
| <i>Join Condition</i>       |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.00     | 0.00 | 0.00 | 0.04   | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00    | 0.00 | 0.00 |
| R                           | 0.00     | 0.00 | 0.00 | 0.07   | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00    | 0.00 | 0.00 |
| F1                          | 0.00     | 0.00 | 0.00 | 0.05   | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00    | 0.00 | 0.00 |
| <i>Datatype</i>             |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.22     | 1.00 | 1.00 | 0.25   | 1.00 | 1.00 | 0.89   | 0.50 | 0.50 | 0.44   | 1.00 | 1.00 | 0.09   | 1.00 | 1.00 | 0.10    | 0.50 | 0.50 |
| R                           | 0.48     | 0.25 | 0.27 | 0.67   | 0.12 | 0.18 | 0.91   | 0.07 | 0.07 | 0.67   | 0.19 | 0.38 | 0.10   | 0.18 | 0.20 | 0.10    | 0.01 | 0.06 |
| F1                          | 0.30     | 0.39 | 0.42 | 0.36   | 0.21 | 0.31 | 0.90   | 0.12 | 0.12 | 0.52   | 0.32 | 0.54 | 0.09   | 0.30 | 0.34 | 0.10    | 0.02 | 0.11 |
| <i>Function Name</i>        |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.78     | 0.00 | 0.00 | 0.62   | 0.00 | 0.00 | 0.47   | 0.00 | 0.00 | 0.75   | 0.00 | 0.00 | 0.67   | 0.50 | 0.50 | 0.00    | 0.00 | 0.00 |
| R                           | 0.92     | 0.00 | 0.00 | 0.83   | 0.00 | 0.00 | 0.42   | 0.00 | 0.00 | 0.75   | 0.00 | 0.00 | 0.42   | 0.08 | 0.25 | 0.00    | 0.00 | 0.00 |
| F1                          | 0.84     | 0.00 | 0.00 | 0.71   | 0.00 | 0.00 | 0.44   | 0.00 | 0.00 | 0.75   | 0.00 | 0.00 | 0.44   | 0.14 | 0.33 | 0.00    | 0.00 | 0.00 |
| <i>Function Output</i>      |          |      |      |        |      |      |        |      |      |        |      |      |        |      |      |         |      |      |
| P                           | 0.00     | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00   | 0.00 | 0.50 | 0.00   | 0.00 | 0.00 | 0.00    | 0.00 | 0.00 |
| R                           | 0.00     | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00   | 0.00 | 0.17 | 0.00   | 0.00 | 0.00 | 0.00    | 0.00 | 0.00 |
| F1                          | 0.00     | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00   | 0.00 | 0.00 | 0.00   | 0.00 | 0.25 | 0.00   | 0.00 | 0.00 | 0.00    | 0.00 | 0.00 |

