WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

EFMD EQUIS ACCREDITED

Formal and Empirical Methods for Maintaining Evolving Knowledge Graphs

Nicolas Ferranti, M.Sc.

Student ID: 12147991

Institute for Data, Process and Knowledge Management
Department of Information Systems and Operations Management
Vienna University of Economics and Business

Supervisor:

Univ.-Prof. Dr. Axel Polleres

Cosupervisor:

Univ.-Prof. Dr. Jairo Francisco de Souza

Internal reviewer:

Univ.-Prof. Dr. Marta Sabou

External reviewer:

Univ.-Prof. Dr. Jose Emilio Labra Gayo

This cumulative dissertation is submitted for the degree of
PhD in Economic and Social Sciences

August 2026

Acknowledgements

This work would not have been possible without the contributions of many people to whom I am deeply grateful.

First, I would like to thank my supervisor, Prof. Dr. Axel Polleres, for his continuous guidance over the years and for always pushing me to think outside the box. I am very thankful for all the sessions we had together, from drafting research topics to chasing paper deadlines. I have learned many valuable lessons that I will carry with me for the rest of my life.

I would also like to thank my co-supervisor, Prof. Dr. Jairo Francisco de Souza, for accepting this challenge and for all the meaningful discussions we had along the way.

Moreover, I would like to express my gratitude to the researchers and colleagues who collaborated directly on the research presented in this thesis: Prof. Dr. Shqiponja Ahmetaj and Dayane Guimarães.

Prof. Dr. Stefan Sobernig and Prof. Dr. Andreas Mild, many thanks for giving me the chance to pursue complementary research challenges and for the opportunity to learn from your extensive experience.

This journey would not have been the same without the supportive environment created by many professionals and colleagues at WU, to whom I am deeply grateful. Many thanks to Martin Beno, Hannah Schuster, Amr Azzam, Stefan Bachhofner, Shahrom Hosseini, Stefani Tsaneva, Majlinda Llugiqi, Venugopal Ramadasu, Jorão Junior, Kenneth Qua, Danii Dobriy, Miguel Vazquez, Alexander Prock, Katrin Ehrenmüller, Felicia Schmidt, Arief Muharram, Hanna Kern, Gregor Käfer, Ikhsan Muhammad, Dr. Kate Revoredo, Dr. Sabrina Kirrane, Alexandra Hager, Roman Franz, Doris Wyk, and Katharina Disselbacher-Kollmann.

Above all, I thank my family. To my father, who always taught me to do the right thing, no matter the consequences. To my mother, who made me understand the value of pursuing my studies. To my brother, who always had my back and helped me realize that not everything has to be so complicated. Finally, to Luma Palmer, who offered me peace and companionship throughout these years.

Abstract

Knowledge graphs have become a central technology for representing, integrating, and reusing structured knowledge across domains. Open and collaboratively maintained knowledge graphs such as Wikidata are particularly valuable because they evolve continuously through contributions from editors, bots, and tools. This openness, however, also complicates their maintenance. In particular, **(i)** Wikidata property constraints are defined through a community-specific model whose validation semantics are partly formalized, **(ii)** approaches to constraint repair have largely focused on changes to data while treating constraints as fixed validation criteria, **(iii)** studies of contributor behavior do not directly address who should inspect a particular constraint violation, and **(iv)** static snapshots and current-state query interfaces do not expose the fine-grained histories of individual statements and constraints. These gaps make it difficult to understand and support maintenance as a process in which data, schema-related structures, constraints, and contributors evolve together.

This dissertation investigates these gaps through four complementary contributions. To address **(i)**, it provides a declarative account of Wikidata property constraints using standard Semantic Web technologies. It analyzes the expressivity of SHACL and provides SPARQL formalizations for all 32 property constraint types considered in the study, allowing their validation semantics to be declaratively expressed and evaluated over Wikidata data. To address **(ii)**, the dissertation extends the analysis of constraint repair beyond changes to individual data statements. It develops a taxonomy covering both instance-level and terminological-level repairs and empirically investigates how changes to data, constraint definitions, and schema-related structures can resolve violations. To address **(iii)**, it investigates how historical repair activity can be used to identify contributors with relevant expertise. By combining repair histories with embedding-based topical clustering, it derives indicators of contributor specialization and evaluates strategies for recommending users for detected violations. To address **(iv)**, it introduces machinery for reconstructing Wikidata statement lifecycles from historical dumps. This transforms page-oriented revision histories into statement-level temporal artifacts containing creation, deletion, and editor provenance, enabling analyses that require historically grounded views of statements, schema-related structures, and constraints.

Together, these contributions provide formal, empirical, and computational methods for maintaining evolving knowledge graphs beyond the detection of isolated inconsistencies in static snapshots. The thesis contributes an operational formalization of community-defined constraints, a repair model that accounts for changes to both data and constraint definitions, an expertise-aware approach to collaborative repair, and reusable machinery for fine-grained historical analysis. By connecting constraint semantics, repair, contributor expertise, and statement evolution, this dissertation provides a unified perspective on knowledge graph maintenance as an evolving process and reusable artifacts for research and tooling around validation, repair, recommendation, and temporal analysis.

Contents

Preface **5**

1 Formalizing and Validating Wikidata's Property Constraints using SHACL and SPARQL **17**

2 Formalizing Repairs for Wikidata Constraint Violations: A Taxonomy and Empirical Analysis **66**

3 Who Can Fix This? User Recommendation for Knowledge Graph Repair via Embedding-Based Clustering **86**

4 Tracing Evolving Knowledge in Wikidata through Statement Lifecycles **91**

5 Conclusions and the Way Ahead **109**
5.1 Main Findings and Answers to the Research Questions 109
5.1.1 RQ1: Declarative Formalization and Validation of Property Constraints 110
5.1.2 RQ2: Repairing Constraint Violations 111
5.1.3 RQ3: Recommending Contributors for Repair 112
5.1.4 RQ4: Statement Lifecycles and Temporal Analysis 114
5.2 Limitations and Future Work 115
5.2.1 Formalization and Validation of Property Constraints 115
5.2.2 Snapshot-based Repair Analysis 116
5.2.3 Dataset Scope for User Recommendation 116
5.2.4 Statement Lifecycle Analysis and Applications 117
5.3 Final Remarks 118

Preface

Knowledge graphs [16] have become a fundamental technology for representing structured knowledge on the Web. They use a graph-based model to describe entities, their attributes, and the relationships between them. Entities may refer to people, places, objects, events, or concepts, while relationships connect these entities and make their associations explicit.

This graph-based model is commonly implemented using Semantic Web standards. The Resource Description Framework (RDF) [24] provides a standard data model for publishing and exchanging knowledge graphs, while SPARQL [12] provides a standard language for querying them. In addition, RDF Schema [3] and the Web Ontology Language (OWL) [10] can be used to describe schema-level knowledge and logical semantics. Constraint languages such as SHACL [18] and ShEx [23] complement these standards by expressing constraints about the structure and validity of the data.

Together, these standards make knowledge graphs useful for building and maintaining collaborative infrastructures to manage structured knowledge. They support the integration of heterogeneous information across domains and facilitate the exchange of data between systems and organizations. In open settings, they also support the collaborative construction of structured knowledge by distributed communities. As a result, large-scale knowledge graphs are used in applications such as search engines, recommender systems, digital assistants, and scientific data integration [17].

However, knowledge graphs are rarely produced once and then only consumed as static datasets. In many settings, they are constructed, revised, and reused over time by multiple actors, including domain experts, knowledge engineers, community users, and automated agents. This is necessary because knowledge itself is not static. New information becomes available, existing information may become outdated or obsolete, errors have to be corrected, and modeling practices are refined as communities learn how knowledge should be represented. In this sense, knowledge graphs increasingly function as infrastructures for the collaborative management of evolving knowledge [22].

Figure 1 illustrates knowledge graph evolution along two axes. The horizontal axis distinguishes the role of time. On the left, time is part of the data itself. This corresponds to valid time, where temporal information describes when facts hold true in the world. On the right, time is metadata about the graph, this corresponds to transaction time, where temporal information describes when statements are created, changed, or deleted in the knowledge graph itself. The vertical axis distinguishes the granularity at which change is observed. At one end, evolution is captured through discrete graph snapshots or dumps. At the other end, evolution is captured through

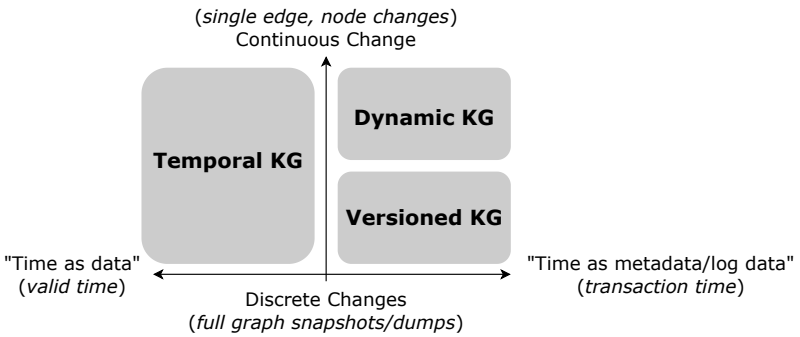


Figure 1: Evolving knowledge graphs dimensions: the horizontal axis distinguishes whether time is represented as part of the data, as valid time, or as metadata about changes to the graph, as transaction time. The vertical axis distinguishes discrete graph snapshots from fine-grained continuous changes. These dimensions give rise to temporal, versioned, and dynamic views of knowledge graph evolution. Reproduced from [22].

continuous, fine-grained changes at the level of individual nodes, edges, or statements. Combining these axes gives rise to complementary perspectives in investigating evolving knowledge graphs: temporal knowledge graphs, versioned knowledge graphs, and dynamic knowledge graphs. These perspectives are not mutually exclusive, but they emphasize different aspects of how knowledge and graph data change over time [22]. Temporal KGs represent the temporal validity of data as part of the graph. Versioned KGs represent static snapshots of their materialized state at specific points in time. Dynamic KGs provide access to fine-grained atomic changes, such as individual node or edge additions and deletions.

Open knowledge graphs illustrate these perspectives in different ways. DBpedia [19] represents an important early model of an open knowledge graph. It extracts structured RDF data from Wikipedia, especially from infoboxes, and transforms collaboratively written encyclopedic content into a queryable knowledge graph. In relation to Figure 1, DBpedia is most naturally associated with the versioned knowledge graph perspective, since its evolution is observable through periodically released graph snapshots [14], even though the underlying Wikipedia content changes continuously. Collaboration in DBpedia is therefore largely indirect. Much of the instance-level data originates from Wikipedia, while the construction of the knowledge graph depends on extraction frameworks, mappings, and ontology maintenance [15].

A different open knowledge graph configuration is illustrated by Wikidata, which we focus on in this thesis as a main object of study. Instead of deriving structured data mainly from another collaborative resource, Wikidata exposes the knowledge graph itself as the object of direct collaborative editing. Launched by the Wikimedia Foundation in 2012 [4], Wikidata provides a central structured knowledge base that can be reused across Wikimedia projects and beyond. Its community can create, revise, discuss, and curate entities, statements, references, and schema-like structures such as properties, classes, and constraints directly in the graph.

Wikidata is a prominent useful case for studying evolving knowledge graphs because its evolution can be observed along all dimensions represented in Figure 1. With respect to the role of time, Wikidata combines temporal information stored as part of the data with temporal information stored as metadata about graph changes. Validity time can be represented within statements, for example through qualifiers that indicate when a statement holds in the world, while transaction time can be recovered from revision histories that record when entities, statements, and related structures are created, changed, or deleted. With respect to the granularity of observation, Wikidata can be analyzed through static dumps and snapshots [7], but its revision histories also make it possible to reconstruct more fine-grained, continuous views of graph evolution [27]. This makes Wikidata especially valuable for longitudinal analyses of evolving knowledge, although reconstructing and processing such fine-grained histories also introduces substantial operational challenges [27].

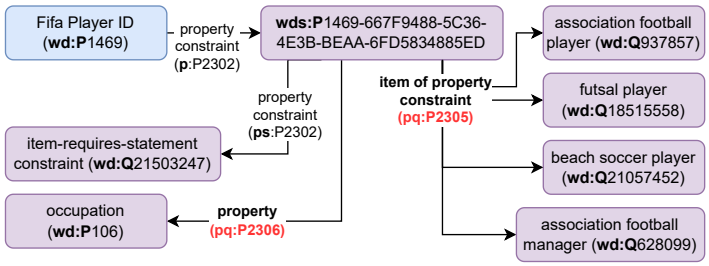
Because Wikidata can be observed across these dimensions, it provides a suitable setting for examining the challenges identified by Polleres et al. [22] for evolving knowledge graphs. At the structural level, both data and schema may change over time, so entities, statements, classes, properties, and constraints cannot be interpreted only with respect to a single static snapshot. At the dynamic level, different parts of a graph may evolve at different speeds: some domains, properties, or communities are highly active, while others remain comparatively stable. Timeliness becomes important when the graph is expected to reflect recent changes in the world. Non-monotonicity is another central issue, because evolution may involve corrections, deletions, and reinterpretations rather than only additions. In collaborative knowledge graphs, these issues are further shaped by the behavior of users, experts, communities, and automated agents, whose activities influence how information is added, inspected, discussed, and repaired. Finally, semantic drift may affect both the meaning of concepts and the modeling practices used to represent them, complicating long-term interpretation, validation, and reuse.

This thesis focuses on a subset of these challenges, centering on the evolution, maintenance, and convergence of data quality, particularly in terms of analyzing repairs and the evolution of constraints and schema within Wikidata. In this context, maintenance refers to the processes through which potential quality problems are identified, interpreted, and addressed as the graph evolves. Wikidata property constraints play a central role in these processes by expressing community-defined expectations about how properties should be used. They do not strictly prevent statements from being added, but instead identify statements that may require inspection.

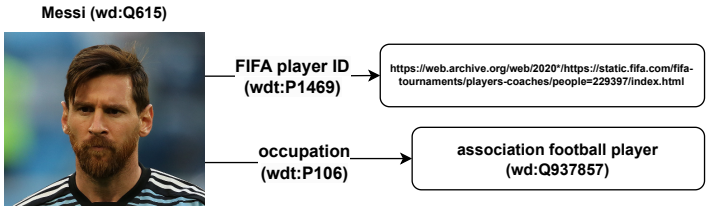
Consider, as an example, an item-requires-statement constraint instance associated with the property FIFA player ID, presented in Figure 2. This constraint specifies that an item carrying a FIFA player ID should also have at least one occupation whose value is one of the following: association football player, futsal player, beach soccer player, or association football manager. In the snapshot represented in the figure, Lionel Messi satisfies this expectation because he has at least one occupation specified by the constraint. By contrast, in the snapshot represented in the figure, Thiago Neves has a FIFA player ID but does not have an occupation matching any of the values specified by the

constraint and is therefore flagged as a constraint violation.

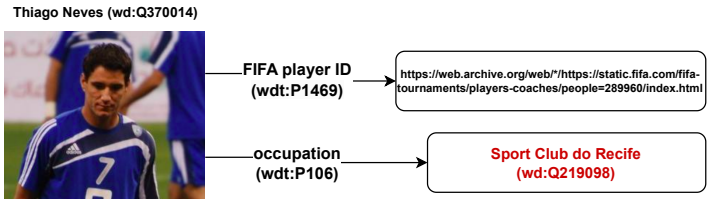
This violation does not establish by itself which part of the graph is incorrect. It might be addressed by adding, removing, or modifying a data statement, but it might also reveal that the values expected by the constraint need to be revised. Determining the appropriate response additionally depends on contributors with relevant knowledge and on understanding how both the statements and the constraint definition developed over time. This example therefore connects four complementary dimensions of Wikidata maintenance: the formal interpretation of constraints, the characterization of possible repairs, the identification of knowledgeable contributors, and the reconstruction of statement and constraint evolution.



(a) Representation of an item-requires-statement constraint instance. Qualifiers used to parameterize the constraint are highlighted in red; the remaining colors indicate the roles of entities in the constraint model [5].



(b) Data graph complying with the constraint.



(c) Data graph not complying with the constraint.

Figure 2: Example of a Wikidata property constraint and data graphs with different behaviors. Reproduced from [5].

From this broader perspective, four thesis-specific challenges are identified.

First, **(C1)** Wikidata property constraints are represented through a community-developed model in which constraint-specific qualifiers parameterize their meaning, while parts of their validation behavior are described in natural-language documentation. At the time of the study presented in Chapter 1, Wikidata comprised 32 property

constraint types instantiated through around 72,000 constraint instances [5]. Previous work provides complementary foundations: MARS offers a logical representation of Wikidata constraints [20], while WShEx proposes a language for describing and validating Wikibase entities [9]. Neither work, however, provides an executable account covering all 32 property constraint types analyzed in Chapter 1, including their constraint-specific qualifiers, Wikidata-specific RDF representation, and documented validation behavior. The challenge is therefore to formalize these constraints unambiguously and reproduce their validation using standard, declarative Semantic Web technologies.

Second, **(C2)** detecting a constraint violation does not by itself determine how the violation should be repaired. A detected violation can reflect an incorrect statement, missing contextual information, or a constraint definition that no longer reflects the intended modeling practice. At Wikidata's scale, distinguishing between these possible sources of violations becomes a substantial maintenance challenge: validation of the 2023 snapshot identified 16.7 million item-requires-statement violations, 34.3 million type violations, and 95 million single-value violations [7]. Existing approaches to SHACL repair and edit-history-based correction primarily consider changes to data while treating the applicable constraints as fixed [1, 26]. The challenge is therefore to characterize repairs that modify data statements, constraint definitions, or both.

Third, **(C3)** in a collaborative setting, such as Wikidata, repair requires identifying contributors with relevant prior expertise. Previous work has characterized contributor roles and collaborative modeling practices in Wikidata [21], but does not directly address the assignment of individual constraint violations to suitable contributors. The challenge is therefore to derive evidence of topical expertise from historical repair activity and use it to recommend potential repairers.

Fourth, **(C4)** static snapshots [7] and current-state query endpoints [1] support analyses of materialized graph states, but fine-grained maintenance analysis additionally requires the transaction times and editor provenance of individual statements and constraints [22]. Historical query infrastructures provide access to revision-level changes and past Wikidata states [27], but raw edit histories remain organized around page revisions rather than explicit statement lifecycles. The challenge is therefore to reconstruct statement and constraint histories at sufficient granularity to analyze how data, schema-related structures, and maintenance activity evolve together.

These challenges motivate the respective research questions addressed in this thesis:

- **RQ1:** How can Wikidata property constraints be formally represented and validated using standard, declarative, Semantic Web technologies?
- **RQ2:** How can constraint violations in Wikidata be systematically categorized and analyzed in terms of possible repair strategies based on static snapshots of the graph?
- **RQ3:** How can knowledge graph repair processes be supported by recommending users with expertise to resolve detected constraint violations?

[1] https://query.wikidata.org/

- **RQ4:** How can reconstructing Wikidata statement lifecycles enable temporal analyses of knowledge graph evolution and maintenance on a fine-grained level, beyond static snapshots?

Research Method

Information Systems research is commonly characterized by two paradigms: (1) behavioral science and (2) design science [13]. Behavioral science develops and validates theories that explain or predict human and organizational behavior, whereas design science focuses on creating and evaluating artifacts that address identified problems and enhance the capabilities of individuals or organizations.

Design Science Research (DSR) provides a structured methodology for developing innovative artifacts and evaluating their effectiveness in solving relevant problems. Typical artifacts include models, methods, algorithms, and software systems. Design science frameworks generally describe an iterative research process consisting of problem identification, artifact design and construction, evaluation, refinement, and communication of results.

This thesis follows the design science paradigm because its main objective is to design, implement, and evaluate computational artifacts for supporting constraint management and repair in large-scale collaborative knowledge graphs. Some parts of the thesis analyze contributor activity, particularly for recommending users who may be able to repair detected violations. These analyses assume that contributors tend to concentrate their activity around topics in which they have expertise and use such observable editing patterns as a proxy for expertise. They do not, however, investigate the cognitive, psychological, or social mechanisms underlying this behavior, and therefore are not positioned as behavioral science research. Using Wikidata as the primary case study, the thesis addresses challenges related to the formalization of constraint mechanisms, the analysis of constraint violations and repairs, and the support of collaborative repair activities. Following the design science framework, the research process comprises four main steps and creates corresponding artifacts.

First, the relevance of the research problem is established by analyzing data quality challenges in collaborative knowledge graphs. In Wikidata, property constraints are defined by the community but are not formally aligned with existing Semantic Web standards, we propose and establish the missing formalizations. Additionally, the collaborative editing model leads to frequent inconsistencies that require manual repair. This situation reveals a gap between existing constraint validation technologies and the mechanisms used in Wikidata to maintain data quality. We also formalize repairs, as well as machinery to evaluate and analyze such repairs.

Second, artifacts are designed to address these challenges. The dissertation proposes formal representations of Wikidata property constraints using SHACL and SPARQL, a taxonomy of repair strategies for constraint violations, a recommendation method for identifying community users capable of resolving detected inconsistencies, and a lifecycle extraction pipeline for reconstructing and analyzing Wikidata statement histories.

Third, the proposed artifacts are evaluated using empirical and computational

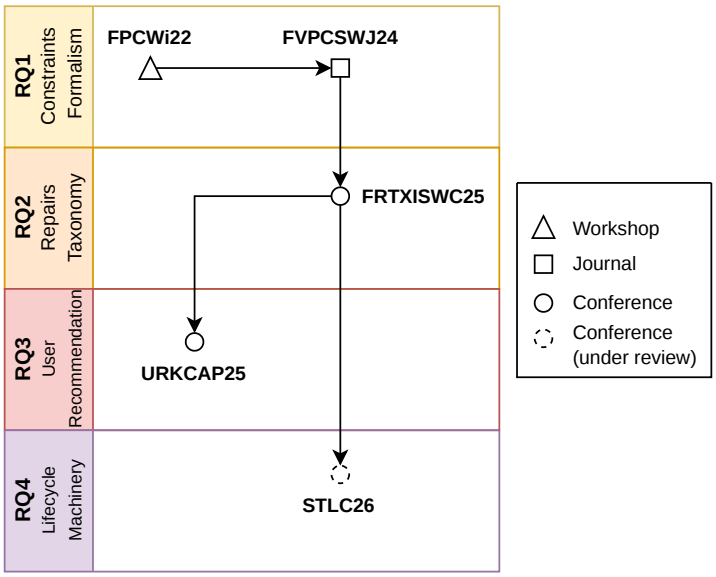


Figure 3: Publication Overview.

methods. The constraint representations are assessed by applying them to Wikidata data and comparing the results with existing violation reports. The repair taxonomy is evaluated through empirical analyses of constraint violations and observed repair patterns. The user recommendation method is evaluated using embedding-based techniques and clustering approaches to assess its ability to identify suitable contributors for resolving violations. The lifecycle machinery is evaluated by reconstructing truthy statement lifecycles from Wikidata's historical dumps and deriving analysis-ready artifacts for property-centered, class-context, and constraint-maintenance analyses.

Finally, insights from the evaluation are used to refine the proposed artifacts and improve their applicability to large-scale collaborative knowledge graphs. This iterative process enables the identification of limitations and shall invite future work towards the improvement of the developed methods.

Selected Papers

The research results of this cumulative thesis are communicated through peer-reviewed publications and one manuscript currently under review that contribute to the body of knowledge in knowledge graph management and Semantic Web research. Figure 3 illustrates the genesis of the papers that are part of this thesis, as well as their contributions to the respective research questions (RQ1-RQ4) and the publication year as a suffix. The type of publication is symbolized via the geometry and the outline of the shape, which is described in the legend: a triangle represents a workshop paper; a square represents a journal article; a circle with a solid outline, a conference paper; and a circle with the densely dashed outline, a conference paper currently under review.

FPCWi22 [8]: *Nicolas Ferranti, Axel Polleres, Jairo Francisco de Souza, and Shqiponja Ahmetaj. Formalizing property constraints in Wikidata. In Proceedings of the 3rd Wikidata Workshop 2022 co-located with the 21st International Semantic Web Conference (ISWC2022), Virtual Event, Hanghzou, China, October 2022, volume 3262 of CEUR Workshop Proceedings. CEUR-WS.org, 2022.* This paper initiates the investigation into the declarative representation of Wikidata property constraints using SHACL shapes and property-agnostic SPARQL queries. All constraint types and the qualifiers used to contextualize constraint instances were systematically mapped. Additionally, reusable SPARQL query patterns are proposed and organized into groups shared across different constraint types.

FVPCSWJ24 [5]: *Nicolas Ferranti, Jairo Francisco De Souza, Shqiponja Ahmetaj, and Axel Polleres. Formalizing and validating Wikidata's property constraints using SHACL and SPARQL. Semantic Web, 15(6):2333-2380, 2024.* In this paper, the discussion and experimental evaluation of the aspects introduced in **FPCWi22** are substantially deepened. We provide a detailed analysis of the nuances of the Wikidata data model, including an extensive discussion of qualifiers and their usage. Furthermore, we examine the expressivity of SHACL-Core and SHACL-SPARQL in representing more challenging Wikidata constraint types. In addition, we analyze the semantics of the SPARQL queries by comparing their results with the database reports system available on Wikidata[2]. The findings demonstrate that SPARQL queries offer a flexible and efficient approach for retrieving batches of constraint violations for further analysis.

FRTXISWC25 [7]: *Nicolas Ferranti, Dayane Guimaraes, Jairo Francisco de Souza, and Axel Polleres. Formalizing repairs for Wikidata constraint violations: A taxonomy and empirical analysis. In The Semantic Web - ISWC 2025 - 24th International Semantic Web Conference, Nara, Japan, November 2-6, 2025, Proceedings, Part I, volume 16140 of Lecture Notes in Computer Science, pages 369-387. Springer, 2025.* Building on our previous work on retrieving batches of constraint violations, this paper investigates the theory of constraint repair. Through existing formalisms, such as MAPL, different types of repairs are characterized and organized into a taxonomy. The paper formally defines logical patterns of inconsistency and, based on these patterns, identifies possible repair strategies that involve either modifying the data or adjusting the constraint parameters. To evaluate the prevalence of these repair patterns, we analyze sets of violations retrieved through the SPARQL queries over time. The empirical results indicate that constraint repairs can have a broader impact on the knowledge graph compared to individual data repairs. Despite this significance, constraint repairs remain neglected in the Wikidata repair literature [26, 28].

URKCAP25 [6]: *Nicolas Ferranti, Dayane Guimaraes, Jairo de Souza, and Axel Polleres. Who can fix this? user recommendation for knowledge graph repair via embedding-based clustering. In Proceedings of the 13th Knowledge Capture Conference*

[2] `https://www.wikidata.org/wiki/Wikidata:Database_reports/Constraint_violations`

2025, K-CAP 2025, Dayton, Ohio, USA, December 10-12, 2025, pages 170-173. ACM, 2025. Due to the large scale and dynamic nature of knowledge graphs, identifying users with expertise to analyze and potentially resolve constraint violations is a crucial step in the knowledge graph lifecycle, particularly in community-driven knowledge graphs such as Wikidata. This paper leverages historical repair activity to investigate whether users can be effectively recommended based on their topical expertise. As an initial step, the study focuses on violations associated with a single constraint type and employs embedding-based clustering to group repairs into coherent topical clusters. The analysis shows that for the constraint type considered in the experiments, approximately 76% of users who performed repairs in the past tend to operate within a single topic cluster, suggesting a strong specialization effect. Furthermore, we evaluate various recommendation strategies and find that ranking users by repair frequency within a cluster outperforms recommendations based on cosine similarity within the same cluster.

STLC26 : *Nicolas Ferranti, Axel Polleres. Tracing Evolving Knowledge in Wikidata through Statement Lifecycles. Under review.* This paper complements prior work [7] by addressing a limitation of snapshot-based analyses: the difficulty of studying how Wikidata statements, schema elements, and constraint-related structures evolve over time. It introduces a pipeline for reconstructing Wikidata statement lifecycles from the full historical dump, including creation and deletion provenance, and derives analysis-ready artifacts for studying knowledge graph maintenance over time. The paper demonstrates the usefulness of this machinery through analyses of property-centered, class-context, and constraint-maintenance activity. It further shows how statement lifecycle analysis can support studies that are not possible using static dumps or standard query endpoints alone, including historical graph slicing, temporally grounded link prediction evaluation, and time-aware selection of schema information for repair models.

Related Papers

In addition to the publications that directly address the research questions of this thesis, related works conducted in our research group with external collaborators have provided important conceptual foundations and inspiration for the research presented in this dissertation. These papers address complementary aspects of Wikidata analysis and knowledge graph evolution, which influenced the design of the methods and artifacts developed in the main publications.

ALWiESWC22 [11]: *Armin Haller, Axel Polleres, Daniil Dobriy, Nicolas Ferranti, and Sergio J. Rodriguez Mendez (2022, May). An analysis of links in Wikidata. In European Semantic Web Conference (pp. 21-38). Cham: Springer International Publishing.* This paper presents a large-scale analysis of links between entities in Wikidata, examining how properties connect items across the knowledge graph and analyzing their structural characteristics. The study investigates the distribution and usage patterns of properties, revealing how different types of links contribute to the connectivity and

organization of Wikidata. The identification of recurring property usage patterns and structural regularities helped inspire the later work on formalizing Wikidata property constraints presented in **FPCWi22** and **FVPCSWJ24**.

HKEvTGDK23 [22]: *Axel Polleres, Romana Pernisch, Angela Bonifati, Daniele Dell'Aglio, Daniil Dobriy, Stefania Dumbrava, Lorena Etcheverry, Nicolas Ferranti, Katja Hose, Ernesto Jimenez-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, December 2023.* This survey investigates the evolution of open and collaboratively maintained knowledge graphs such as Wikidata and DBpedia. The paper analyzes different dimensions of knowledge graph evolution, including structural changes, temporal aspects of data, and collaborative editing processes. This dynamic environment motivates the need for systematic approaches to detect constraint violations and manage repair processes, as investigated in **FRTXISWC25**. Additionally, the analysis of collaborative editing patterns and user roles helped motivate the user recommendation approach proposed in **URKCAP25**. Most directly, this survey also motivates **STLC26** by identifying knowledge graph evolution as a central challenge for open collaborative knowledge graphs and by emphasizing the need for methods that go beyond static snapshots toward temporal analysis of evolving facts, schema, constraints, and community activity.

Research Impact

The research presented in this thesis contributes both theoretical and practical advances toward improving the quality and maintenance of large-scale collaborative knowledge graphs, with a particular focus on Wikidata. The work addresses the research questions introduced earlier by developing formal methods to represent constraints, analyze violation patterns, and support collaborative repair processes. Beyond the individual publications included in this thesis, the results have also contributed to community tools, educational activities, and events aimed at strengthening collaboration between the Semantic Web research community and the Wikidata ecosystem.

With respect to **RQ1**, this thesis provides a formal representation of Wikidata property constraints using standard, Semantic Web technologies such as SHACL and SPARQL. Paper **FVPCSWJ24** [5] demonstrates how the semantics of Wikidata's community-defined constraints can be expressed declaratively and evaluated using standard query mechanisms. In particular, the SPARQL formalizations developed in this research allow violations for almost all current property constraint types to be retrieved directly from Wikidata data. This formalization helps clarify ambiguities in the current constraint documentation and provides reproducible methods for constraint validation that can be reused in tools and future research.

Addressing **RQ2**, the thesis investigates how constraint violations arise and how they can be systematically repaired. Building on the formal retrieval of violation data, paper **FRTXISWC25** [7] proposes a taxonomy of repair strategies that categorizes different types of inconsistencies and possible repair actions. The empirical analysis in

this research shows that repairs can involve either modifying individual data statements or adjusting constraint definitions. The results highlight that constraint-level repairs can have broader effects on the knowledge graph than individual edits, suggesting that understanding repair dynamics is essential for maintaining data quality in collaborative knowledge graphs.

RQ3 focuses on supporting collaborative repair processes by identifying users who are likely able to resolve specific constraint violations. To this end, the thesis proposes a user recommendation approach based on embedding-based clustering of historical repair activities, detailed in Chapter 3 and in **URKCAP25** [6]. The empirical analysis indicates that community members tend to specialize in particular topical areas of the knowledge graph, allowing violations to be associated with members who have previously performed similar repairs. The evaluation shows that ranking users by repair activity within topical clusters can yield effective recommendations for potential repairers.

RQ4 extends the thesis by addressing the temporal dimension of Wikidata maintenance. The thesis introduces statement lifecycle machinery that reconstructs the creation, modification, and deletion of statements and links these lifecycles to editor provenance. From these lifecycles, **STLC26** derives artifacts for analyzing property usage, class-context evolution, and constraint-maintenance activity over time. This enables empirical analyses that are not possible from static dumps alone and provides a foundation for historically grounded repair analysis, temporal graph slicing, and time-aware evaluation of knowledge graph repair methods.

Beyond the conceptual and empirical contributions, this thesis also produces reusable research artifacts. The work on property constraints provides declarative SPARQL formalizations for all 32 constraint types considered in Chapter 1, together with corresponding SHACL representations where expressible and the `wd2shacl` prototype for automatically generating SHACL shapes from Wikidata constraint instances. These resources also provide real-world workloads for evaluating SHACL validators and SPARQL engines on Wikidata constraint validation [3]. The repair study is accompanied by open-source extraction and analysis code, SPARQL queries, and datasets of historical violations and repairs between the 2019 and 2023 snapshots [4]. The user-recommendation study provides an extended repair dataset linking historical repairs to their contributors, together with the implementation and experimental resources for the embedding, clustering, and recommendation pipeline [5]. The statement-lifecycle work provides an implemented extraction pipeline that transforms Wikidata page-history dumps into statement-level lifecycle artifacts, including truthy statements, class and hierarchy histories, constraint lifecycles, and an entity-to-file index for downstream historical analyses[6]. Together, these outputs make the methods developed in the thesis reproducible and provide reusable resources for further work on knowledge graph validation, repair, user recommendation, and temporal analysis.

[3] `https://github.com/nicolasferranti/wikidata-constraints-formalization`

[4] `https://github.com/nicolasferranti/wikidata-repairs`

[5] `https://github.com/nicolasferranti/wd_user_recommendation`

[6] `https://anonymous.4open.science/r/wikidata_statements_lifecycle_machine-77B0/`

Community Contributions

Beyond the scientific contributions reported in the included publications, the research conducted during this PhD has also contributed to the broader dissemination and development of knowledge within the Semantic Web and Wikidata communities. The methods and insights developed in this work have been presented at tutorials and community events for both researchers and practitioners working with Wikidata and knowledge graph technologies. In particular, a tutorial related to Wikidata constraint analysis and repair was presented at the International Semantic Web Conference (ISWC 2024) in Baltimore[7].

In addition, this research contributed to organizing community events focused on Wikidata and knowledge graph quality. This includes the organization of the 5th Wikidata Workshop co-located with the International Semantic Web Conference (ISWC 2025) in Nara[8]. Further dissemination activities are planned for 2026, including Wikidata-related courses at the European Summer School in Artificial Intelligence (ESSAI 2026)[9] and the organization of the 6th Wikidata Workshop, which has been accepted as a co-located event of the International Semantic Web Conference (ISWC 2026) in Bari[10]. Together, these activities aim to foster collaboration between researchers and practitioners and to support the continued development of tools and methodologies for maintaining high-quality collaborative knowledge graphs.

[7] https://ww101.ai.wu.ac.at/

[8] https://wikidataworkshop.github.io/2025/

[9] https://essai2026.eu/courses.php

[10] https://wikidataworkshop.github.io/2026/

Chapter 1

Formalizing and Validating Wikidata's Property Constraints using SHACL and SPARQL

Published in The Semantic Web Journal: Interoperability, Usability, Applicability. 2024;15(6):2333-2380. doi:10.3233/SW-243611

Semantic Web 15 (2024) 2333–2380
DOI 10.3233/SW-243611
IOS Press

Formalizing and validating Wikidata's property constraints using SHACL and SPARQL

Nicolas Ferranti [a,*], Jairo Francisco De Souza [b], Shqiponja Ahmetaj [c] and Axel Polleres [a,d]

[a] *Department of Information Systems and Operations Management, Vienna University of Economics and Business, Austria*
E-mails: nicolas.ferranti@wu.ac.at, axel.polleres@wu.ac.at
[b] *Department of Computer Science, Federal University of Juiz de Fora, Brazil*
E-mail: jairo.souza@ufjf.edu.br
[c] *Vienna University of Technology, Austria*
E-mail: shqiponja.ahmetaj@tuwien.ac.at
[d] *Complexity Science Hub Vienna, Austria*

Editors: Lucie-Aimée Kaffee, Applied Policy Researcher at Hugging Face, Berlin, Germany; Simon Razniewski, Institute for Artificial Intelligence, Dresden, Germany; Pavlos Vougiouklis, Huawei Technologies, Edinburgh, United Kingdom
Solicited reviews: Jose Emilio Labra Gayo, Web Semantics Oviedo (WESO) research group, University of Oviedo, Spain; One anonymous reviewer

Abstract. In this paper, we delve into the crucial role of constraints in maintaining data integrity in knowledge graphs with a specific focus on Wikidata, one of the most extensive collaboratively maintained open data knowledge graphs on the Web. The World Wide Web Consortium (W3C) recommends the Shapes Constraint Language (SHACL) as the constraint language for validating Knowledge Graphs, which comes in two different levels of expressivity, SHACL-Core, as well as SHACL-SPARQL. Despite the availability of SHACL, Wikidata currently represents its property constraints through its own RDF data model, which relies on Wikidata's specific reification mechanism based on authoritative namespaces, and – partially ambiguous – natural language definitions. In the present paper, we investigate whether and how the semantics of Wikidata property constraints, can be formalized using SHACL-Core, SHACL-SPARQL, as well as directly as SPARQL queries. While the expressivity of SHACL-Core turns out to be insufficient for expressing all Wikidata property constraint types, we present SPARQL queries to identify violations for all 32 current Wikidata constraint types. We compare the semantics of this unambiguous SPARQL formalization with Wikidata's violation reporting system and discuss limitations in terms of evaluation via Wikidata's public SPARQL query endpoint, due to its current scalability. Our study, on the one hand, sheds light on the unique characteristics of constraints defined by the Wikidata community, in order to improve the quality and accuracy of data in this collaborative knowledge graph. On the other hand, as a "byproduct", our formalization extends existing benchmarks for both SHACL and SPARQL with a challenging, large-scale real-world use case.

Keywords: Wikidata, data quality, knowledge graphs, constraints, Shapes Constraint Language, SPARQL

*Corresponding author. E-mail: nicolas.ferranti@wu.ac.at.

1570-0844 © 2024 – The authors. Published by IOS Press. This is an Open Access article distributed under the terms of the Creative Commons Attribution License (CC BY 4.0).

1. Introduction

A Knowledge Graph (KG) uses a graph-based model to represent real-world entities, their attributes, and relationships [40]. Entities are anything that can be uniquely identified and described, such as people, places, things, or concepts, but also the relationships between those. The "graph" metaphor stems from the idea of depicting statements representing relationships between entities as directed graph edges. A wide range of information can be represented using KGs, including encyclopedic knowledge, scientific data, corporate data, and, – along with meta-information attached to statements – also contextual information, such as who *provenance*, *preference* amongst statements, or temporal context (e.g. *when* a statement was added, so-called transaction time, or was valid, called *validity time*, cf. e.g. [30,61]). The Semantic Web initiative within the World Wide Web Consortium (W3C) has established a set of essential standards, readily available to manage and process KGs:

- the *Resource Description Framework (RDF)* [54] to publish and interchange KGs;
- the *SPARQL Protocol and RDF Query Language* [32] to query KGs;
- *RDF Schema (RDFS)* [15] and the *Web Ontology Language (OWL)* [37] to define and describe the schema of KGs in RDF itself;
- the *Shapes Constraint Language (SHACL)* [29,41] to validate KGs.

The goal of said standards is to enable interoperability, but also the ability to unambiguously describe the (allowed) schema and semantics of knowledge graphs, which in turn is a crucial aspect in order to maintain KG quality, as more and more KGs are published in a decentralized, collaboratively created fashion across the Web.

Since its creation by the Wikimedia Foundation in 2012, Wikidata has become one of the largest such KGs, publicly available on the Web, with more than 100M items[1] and 14B triples.[2] One of the main factors responsible for this growth is Wikidata's user community, with more than 24k active users (humans and bots). At the beginning of Wikidata, the large user community was primarily motivated by enriching Wikipedia with structured data, as Wikipedia pages increasingly incorporate content from Wikidata [24]; yet, in the meantime, Wikidata has gained importance and usage far beyond and independent of Wikipedia.

In terms of supporting the above-mentioned Semantic Web standards, the Wikidata KG is available in standard RDF format and can be queried via a public SPARQL endpoint. Yet, Wikidata does neither adhere to OWL/RDFS, nor SHACL: while other knowledge graphs often have predefined formal ontologies or schemas defined in RDFS and OWL, Wikidata takes a different approach, with its community focusing on the development of the data layer (A-box) and the terminology layer (T-Box) evolving alongside with it. This means that Wikidata does not have a single, pre-defined formal ontology [50] adhering to RDFS/OWL's well-defined semantics. In fact, while some Wikidata properties, such as *subclassOf* (P279) or *subproperty of* (P1647), loosely correspond to constructs of the OWL and RDFS vocabularies [31], Wikidata does not make any formal ontological commitment on these properties in terms of OWL's Description Logics based semantics, and the respective properties are rather freely used and usable by the community. Rather, in order to reinforce consistent usage of the community-developed terminology, separate Wikidata projects have emerged to specify *constraints*, which serve as a means to identify errors in the data layer wrt. vocabulary usage. However, none of these projects deploys the current W3C recommendation for validating RDF graphs against constraints, namely, SHACL.

In the current paper, we focus on the largest and most widely supported amongst these constraints approaches in Wikidata, namely the *Wikidata Property Constraints Project*:[3] in this project, Wikidata has developed its own "representation model" to describe constraints on properties, both on property values in statements, but also contextual meta-data aspects on the usage of such properties. We estimate that 99% of Wikidata properties are affected by at least one property constraint, while further projects that define constraints on the class level only cover around 0.2% of the classes (for details on those other approaches and constraint projects cf. Section 7).

When it comes to how property constraints should be interpreted/checked, there is a description in natural language for each constraint type available on a respective help page, for instance, the *single-value constraint*

[1] https://www.wikidata.org/wiki/Wikidata:Statistics, as from January 2023.
[2] https://short.wu.ac.at/7t66, last accessed 13 February 2023.
[3] https://www.wikidata.org/wiki/Help:Property_constraints_portal

(Q19474404).[4]) As opposed to W3C's standard, SHACL, which relies on standardized validators to identify inconsistencies, Wikidata calculates its own violation reports, the so-called *Wikidata Database reports* with an ad-hoc extension of Wikibase (Wikidata's underlying software framework [55]). Violations per constraint type are published as parts of these reports on separate HTML pages.[5] Yet, the approach behind the generation of these reports is not published, and there is a maximum limit of violations displayed for each constraint type in the respective property pages. For a community-based KG with billions of triples, efforts like the Property Constraints project represent a key resource for creating tools to assist in the analysis and refinement of inconsistent data on Wikidata. However, we believe that the development of such tools is hampered by the way this data is currently collected and made available: since the only official description of how to check property constraints is in natural language, and their verification is not entirely transparent, the semantics of property constraints may be subject to ambiguous interpretations.

In the present paper, we explore the use of both SHACL and SPARQL as tools for formalizing Wikidata's property constraints; the use of these standardized tools should provide more accurate, open, and efficient means of identifying and addressing inconsistencies in Wikidata and resolving potential ambiguities. To this end, our main contributions are as follows:

- We provide a gentle and comprehensive introduction to Wikidata's specific, namespace-based RDF reification model with many illustrative examples, that show how Wikidata's wide range of different property constraints are represented using this model.
- We study to what extent the expressiveness of the SHACL-Core language is sufficient to express Wikidata property constraints and come to the conclusion that the SHACL-Core language is not expressive enough to represent all Wikidata property constraints: Among the 32 investigated property constraint types, SHACL-Core lacks components to express two of them. In addition, we argue that another four constraint types are not reasonably, or only partially expressible.
- For the Wikidata property constraints expressible in SHACL-Core, we present a tool to automatically translate such constraints; the tool can benefit also other Wikibase KGs that import Wikidata property constraints.
- We show how the non-SHACL-Core-expressible remaining constraints can be formalized in full SHACL (using the SHACL-SPARQL extension), and argue for an, in our opinion more effective, formalization in SPARQL alone.
- We consequently unambiguously formalize all 32 Wikidata property constraint types as SPARQL queries which provide a declarative means to express constraints, directly operationalizable via Wikidata's SPARQL endpoint.[6] SPARQL queries offer the possibility of checking the violations in real-time on the Wikidata SPARQL endpoint, as well as the flexibility to collect useful information to analyze the usage of a constraint such as status, reasons for deprecation, and exceptions.
- We present a comparison of our SPARQL approach to the current Wikidata violation reports, demonstrating the feasibility of using SPARQL to actually check constraints: particularly, we highlight potential ambiguities and reasons for deviations in violations found with our approach compared to the Wikidata violation reports; we believe that our approach as such helps clarifying such ambiguities in a reproducible manner.
- We note that, due to the known scalability limits of Wikidata's SPARQL endpoint, we still run into timeouts in checking some of the most violated constraints; yet we argue that our work, can be understood as providing challenging benchmarks for both (i) SHACL validators and (ii) SPARQL engines, based on the real-world use case of Wikidata; as such we extend and go beyond recent related benchmarks.[7]

[4] https://www.wikidata.org/wiki/Help:Property_constraints_portal/Single_value

[5] https://www.wikidata.org/wiki/Wikidata:Database_reports/Constraint_violations/Summary

[6] Notably, as it turns out, some constraint types can only be partially evaluated online due to incomplete RDF representation of Wikidata's own RDF data model on Wikidata's SPARQL query endpoint.

[7] For instance, our constraint checking queries are not restricted to "truthy" statements, as opposed to the recent WDbench [6] SPARQL benchmark.

The remainder of this paper is structured as follows. Section 2 presents an exhaustive, tutorial-style introduction to Wikidata's property constraints, diving into Wikidata's RDF meta-modeling, and explaining how property constraints are represented within this model.

Section 3 discusses how to represent the semantics of Wikidata property constraints in SHACL-Core. We also present *wd2shacl*, a tool that automatically generates expressible SHACL constraints from Wikidata property constraints, and as such could be viewed as a "benchmark generator" for SHACL motivated by a real use case.

Yet, as not all Wikidata property constraints are expressible in SHACL-Core, in Section 4 we instead present a complete mapping of *all* Wikidata property constraints to SPARQL: we argue that SPARQL can be used for operationalizing all property constraints' verification continuously, directly on the Wikidata SPARQL endpoint.

As a demonstration of feasibility, we present a detailed analysis and experiments, comparing violations found by our approach with the officially reported constraint violations by Wikidata itself in Sections 5 and 6.

Finally, after discussing related works on constraint formalization and quality analysis for Wikidata and other KGs in Section 7, we conclude in Section 8 with pointers to future research directions.

2. Background

As mentioned already in the introduction, standard ontological inference as a means to detect inconsistencies is not directly applicable to the approach taken by Wikidata. Firstly, Wikidata's data model may arguably be described as extending RDF's plain, triple-based model, by various meta-modeling features for adding references and other contextual qualifiers to statements. Indeed, Wikidata's data model is mapped to RDF via a specific reification mechanism. Secondly, there is neither a strict distinction between the data and terminology layers nor does Wikidata's terminology rely on OWL/RDFS [31]. Rather, the terminology layer evolves in the background as editors add/update new facts, potentially introducing new properties and classes in a community-based approach. Additionally, proprietary, community-driven, ad-hoc processes have been set up within Wikidata to define constraints on the terminology used. In particular, the *Property Constraints* project,[8] which we will focus on in this paper, aims at defining restrictions applied to the usage of Wikidata properties.

In order to provide the required background, in the following subsections we introduce the RDF data representation adopted by Wikidata (Section 2.1) with several examples, followed by illustrating details of how Wikidata's property constraints are represented within this data model (Section 2.2), in particular focusing on qualifiers used as "parameters" for constraint definitions (Section 2.3). Finally, we discuss both (i) challenges in understanding the exact meaning of these property constraints (the semantics of which are largely described in natural language only), as well as (ii) potential issues in verifying them on Wikidata's RDF representation (Section 2.4).

2.1. Data modeling in Wikidata

In this section, we describe how Wikidata's data model – and specifically property constraints – are modeled in RDF and can be queried with SPARQL.[9]

To this end, let us start with the bare basics of RDF and SPARQL and then gradually dive into the specifics of Wikidata. When talking about RDF, as usual, we will refer to *RDF graphs* and their subgraphs as sets of *triples*

(*subject*, *predicate*, *object*)

where *subjects* and *predicates* are typically URIs, whereas objects can be either plain, typed, or language-tagged literal values. These triples can be viewed as directed edges in a graph, such that we may consider a simple graph

[8] https://www.wikidata.org/wiki/Wikidata:WikiProject_property_constraints

[9] For details, we refer the reader to https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format, focusing on the part of Wikidata's RDF representation relevant to and affected by constraints.

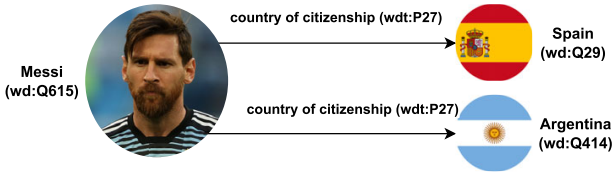


Fig. 1. Simple RDF (sub-)graph example from Wikidata, showing two statements/triple, indicating that Lionel Messi (URL: `wd:Q615`) has two citizenships (property URL: `wdt:P27`): Spain (`wd:Q29`) and Argentina (`wd:Q414`).

Table 1

The most important namespaces used in Wikidata's RDF representation; we omit some additional standard namespaces such as *dct*: (Dublin core terms), *schema*: (Schema.org), *rdf*: (RDF), *owl*: (OWL), etc

Prefix	Namespace	Purpose
wd:	http://www.wikidata.org/entity/	used to identify Wikidata concepts (entities or properties).
wdt:	http://www.wikidata.org/prop/direct/	used for direct property relationships (claims) between (*wd*:) concepts.
p:	http://www.wikidata.org/prop/	relates a (*wd*:) concept to a (*wds*:) statement node further describing the direct (*wdt*:) claim.
wds:	http://www.wikidata.org/entity/statement/	used for statement nodes, i.e., intermediate nodes that link claims to statement qualifiers.
pq:	http://www.wikidata.org/prop/qualifier/	used for "qualifying", adding meta-data to (*wds*:) statements.
ps:	http://www.wikidata.org/prop/statement/	used to "re-connect" (*wds*:) statement nodes to the original claim's object value.
psv:	http://www.wikidata.org/prop/statement/value/	used for statement nodes referring to quantity value nodes in the (*wdv*:) namespace.
wdv:	http://www.wikidata.org/value/	used for value nodes referring to quantity values.
prov:	http://www.w3.org/ns/prov#	used in Wikidata exclusively (cf. https://w.wiki/7KX7) for the property *prov:derivedFrom* from the PROV ontology [43] to link claims to references.
pr:	http://www.w3.org/ns/prov#	used in for properties defined by the Wikidata community to further describe references.
wikibase:	http://wikiba.se/ontology#	reserved namespace to denote special properties in the Wikibase ontology, which refers to some special concepts from https://wikiba.se, Wikidata's underlying software framework.
ontolex:	http://www.w3.org/ns/lemon/ontolex#	Wikidata re-uses specific properties from the Ontolex-Lemon [19] vocabulary to describe lexemes.

as in Fig. 1, stating that Lionel Messi has two citizenships, as a set of two triples

$$G_{simple} = \{(\texttt{wd:Q615}, \texttt{wdt:P27}, \texttt{wd:Q29}), (\texttt{wd:Q615}, \texttt{wdt:P27}, \texttt{wd:Q414})\}$$

Here, URIs are represented as namespace-prefixed identifiers, e.g. *wd:Q615* which should be understood as a shortcut for a full URI, e.g. ⟨http://www.wikidata.org/entity/Q615⟩, according to the namespace prefixes defined in Table 1. Additionally – particularly in SHACL examples later on – we will also refer to RDF graphs using Turtle [10] syntax in the remainder of this paper. As an illustration, G_{simple} can be written in Turtle concisely as follows:

```
wd:Q615 wdt:P27 wd:Q29,
                wd:Q414 .
```

For queries over RDF data, we will use SPARQL [32]: indeed the RDF representation of Wikidata can be fully queried by means of Wikidata's SPARQL query service. Wikidata's query service.,[10] which allows to formulate

[10] available at https://query.wikidata.org.

queries over RDF triples in terms of graph patterns, such as for instance:

```
SELECT ?Citizenship WHERE { wd:Q615 wdt:P27 ?Citizenship }
```

Here, the triple pattern `(wd:Q615, wdt:P27,?Citizenship)` would match both countries, Argentina and Spain on the graph G_{simple}.[11]

Other methods, apart from Wikidata's query service to access RDF from Wikidata include complete RDF dumps[12] or retrieving RDF triples per entity via Wikidata's Web API directly.[13]

Statement nodes: Wikidata's internal data model heavily relies on meta-modeling, i.e., claims such as the ones represented by the triples in G_{simple} can be annotated yet again with contextual meta-information, provenance and temporal information, etc. In order to map this meta-information to "flat" RDF triples, Wikidata's RDF representation relies on the consistent usage of specific, authoritative [31,38]) namespaces,[14] the most important of which are listed in Table 1. We note that Wikidata's meta-modeling in terms of specific namespaces can be seen as a custom *reification mechanism*: here, we mean "custom" in the sense of not following any "standard" reification scheme, such as the RDF reification vocabulary [34, Appendix D], named graphs [18], singleton reification [46], RDF-* [33], etc., for a detailed discussion, we refer the interested reader to [36].

In this reification model, Wikidata uses URIs that represent hashes for "anonymous" reified *statement nodes* and *quantity value nodes* as illustrated in our examples in the following: for instance, Fig. 2 shows a more complete graphical illustration of Wikidata's RDF model including meta-information about Messi, adding such statement nodes, which yields the following set of triples G:

$$
\begin{aligned}
G = \{&(\texttt{wd:Q615}, \texttt{wdt:P27}, \texttt{wd:Q29}),\\
&(\texttt{wd:Q615}, \texttt{p:P27}, \texttt{wds:Q615-469B5D33-8EA7-4CAB- 8A71-75AE59EAFD85}),\\
&(\texttt{wds:Q615-469B5D33-8EA7-4CAB-8A71-75AE59EAFD85}, \texttt{ps:P27}, \texttt{wd:Q29}),\\
&(\texttt{wds:Q615-469B5D33-8EA7-4CAB-8A71-75AE59EAFD85}, \texttt{pq:580}, \texttt{"2005 - 01 - 01"}),\\
&(\texttt{wds:Q615-469B5D33-8EA7-4CAB-8A71-75AE59EAFD85},
\end{aligned}
$$

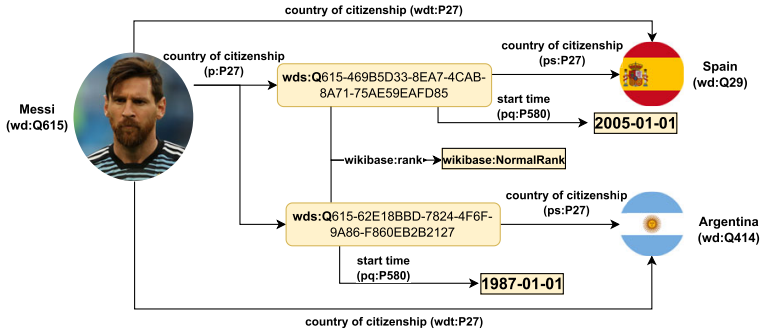


Fig. 2. Subgraph example from Wikidata. Direct claims can be stated (using *wdt*), while metadata is added through qualifiers (using *pq*). Statement ranks use the property *Wikibase:rank*.

[11] Note that, interestingly, the query from above also returns `wd:Q38` (Italy) on Wikidata's query service, cf. https://w.wiki/8Vqt: we will occasionally refer to such shortcut URLs to Wikidata's query service in the following for illustrating notable queries.

[12] cf. https://www.wikidata.org/wiki/Wikidata:Database_download#RDF_dumps.

[13] For instance, supporting content negotiation, the result of running `curl -L -H "Accept:text/turtle"` https://www.wikidata.org/entity/Q615 retrieves all RDF data for Messi in Turtle syntax.

[14] In a nutshell, a dataset speaks "authoritatively" about a URI, or likewise a namespace prefix, if it is published/accessible on the same pay-level-domain. I.e., for instance, Wikidata is authoritative for all URIs (and, resp., namespaces) which start with https://www.wikidata.org.

`wikibase:rank, wikibase:NormalRank),`

`...}`

As a side note, let us emphasize that Wikidata's use of such "hashed" statements nodes is a deliberate choice to avoid the use of *blank nodes* [39].[15]

Entities: Items and properties, and lexemes In principle, any concept in Wikidata is an *entity*, such as real-world entities, but also properties, classes, or specific property constraint types; entities can be directly referred to as subjects or objects in *statements* via the namespace *wd*:. Entities are further subdivided into *Properties*, and *Items* which – rather than by prefix – can be distinguished by their numeric identifiers: **Q**-identifiers are used for Items, such as **Q**615, denoting the Item/Entity "Lionel Messi", whereas **P**-identifiers are used for properties, such as **P**27 for the relation "country of citizenship".

Besides Items and Properties, since a large part of Wikidata is also specialized in linguistics knowledge and multilinguality, another special kind of entities, so-called *Lexemes*, i.e., words with their senses and forms in particular different languages, are identified by separate **L**-identifiers, cf. Example 5 below.

Claims and statements: Claims made about entities are represented as "flat" RDF triples and use Properties in the predicate position with the namespace *wdt*:, such as the triples in G_{simple} above. Note that, just like in the claim "Messi's country of citizenship is Argentina", which is represented by the RDF triple (`wd:Q615, wdt:P27, wd:Q414`), statements about Properties also use these different namespaces: that is, the *wd*: namespace is never used in a predicate position, whereas the *wdt*: namespace is never used in subject or object position. As an illustration, the claim that property *country of citizenship* (P27) is a *subproperty of* (P1647) *country* (P17) is denoted by the triple

```
(wd:P27, wdt:P1647, wd:P17)
```

Such direct claims can be further described and annotated with meta-information. That is, for each claim, a separate, *wds*:-prefixed *statement node* is created in Wikidata's RDF graph, which permits to refer to the claim itself in meta-statements, such as for instance declaring since when Messi has the Argentinean citizenship. These statement nodes are connected to the claim's subject entity via the claim's property using prefix *p*: instead of *wdt*:; additional meta-information about statement nodes uses specific so-called *qualifier properties*, denoted by the prefix *pq*:, and the statement node itself is connected back to the claim's object via the claim's property using prefix *ps*:.

Example 1. Fig. 2 presents a subgraph of Wikidata that illustrates this RDF representation, containing two claims about *Lionel Messi* (Q615), concerning his two nationalities and their different respective *start time* (P580) as qualifiers of the respective claims' statement nodes. We will explain the additional *wikibase:rank* triples also visible in the next figure.

Statement ranks and truthy statements: Note that not all statements in Wikidata are represented as directly queryable *wdt*: claims, the reason being that statements may be marked as normal (*wikibase:NormalRank*), preferred (*wikibase:PreferredRank*), or deprecated (*wikibase:DeprecatedRank*), via the special "statement-rank" (*wikibase:rank*) property. While in Fig. 2, both claims have rank *wikibase:NormalRank*, i.e. denoting equally valid – so-called "truthy" claims, let us provide another illustrating example to show a different setting.

Example 2. Fig. 3 shows two claims about the capital of the US, one of which (the current capital) has a *wikibase:PreferredRank*, while the other has *wikibase:NormalRank*: when you compare both figures, note that in presence of a *wikibase:PreferredRank*ed statement, only this preferred statement has a direct, i.e. "truthy" *wdt*: triple. Intuitively, in this example, this makes sense, as it allows us to write simple SPARQL queries to Wikidata's query service, asking for the *capital* (P36) of the *USA* (Q30) just with:

```
SELECT ?Capital WHERE { wd:Q30 wdt:P36 ?Capital }
```

[15] cf. https://www.mail-archive.com/wikidata-tech@lists.wikimedia.org/msg01511.html.

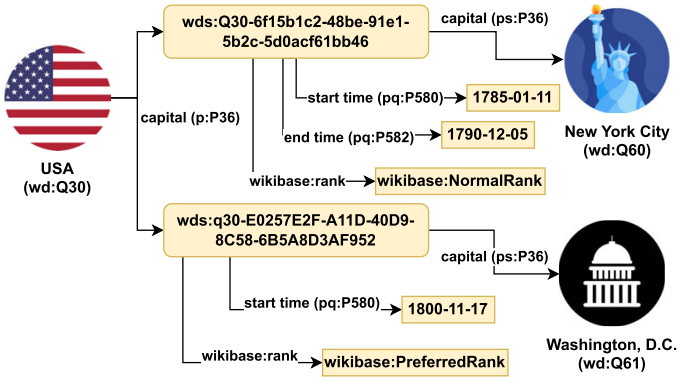


Fig. 3. A subgraph showing claims about two *capitals* (P36) of the *USA* (Q30) and their start and end times. According to a *single-value* constraint on P36, multiple values are allowed as long as they have different values for *start time* (P580) (and other so-called "separators", not shown here).

which will only return the current capital, whereas if we wanted to query for *all* past and present capitals, we would need a more complex query using a path expression:

```
SELECT ?Capital WHERE { wd:Q30 p:P36/ps:P36 ?Capital }
```

Note that similarly, *wikibase:DeprecatedRank*ed statements (which we do not illustrate at this point in a separate example, but which we will get back to later) do not have a *wdt*: claim in Wikidata's RDF representation. Following the intuition that only preferred, non-deprecated, normal ranked statements without an "overriding" preferred statement are represented as *wdt*: triples, these wdt:statements are also called "truthy" statements in Wikidata terminology.

Quantity values: quantity values such as numbers, dates, etc. are represented in Wikidata – yet again different to RDFs typed literals – using a similar, specific reification mechanism, referring to properties using the separate namespace *psv*: to refer to quantity values, which can have an amount and unit, referred to by special *wikibase*: properties.

References: references can be given for any claims, where all such references are referred to with the *prov:derivedFrom* property. Similar to quantity values, reference nodes are represented by hash values in a separate namespace (*wdref*:), which can be further described with property-value pairs, using reference properties – identified by yet another separate namespace (*pr*:).

Example 3. The upper half of Fig. 4 illustrates the modeling of quantity values in Wikidata, in this case about Lionel Messi's height. The lower part of the figure illustrates another heavily used feature of Wikidata, namely references: the property "reference URL" (*pr:P854*) is used to provide a reference source for the quantity claim, in the form of a URL that reported Lionel Messi's height.

Labels and descriptions: strings used to name or describe entities, i.e., Items, Properties, or also Lexemes, in different languages, are denoted in Wikidata's RDF dump by language tagged literals, using the reserved properties *rdfs:label* and *schema:description*, respectively.

Example 4. Figure 5 illustrates labels and descriptions, where English, Spanish, and Arabic labels and descriptions for Item Q615 (*Messi*) are shown.

Special statements about lexemes: linguistic knowledge about lexemes plays a key role in Wikidata, and uses a yet again dedicated representation; as a final example, we present a subgraph about the lexeme "football", involving its senses (*ontolex:sense*) and lexical forms (*ontolex:lexicalForm*).

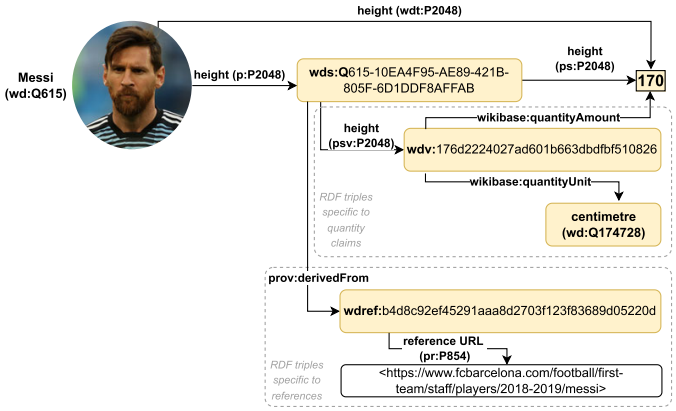


Fig. 4. A subgraph containing a *quantity claim* about the *height* (P2048) of Messi, and a *reference* for this claim. According to an *allowed-units* constraint on P2048, *centimetre* (Q174728) is an allowed unit for this property.

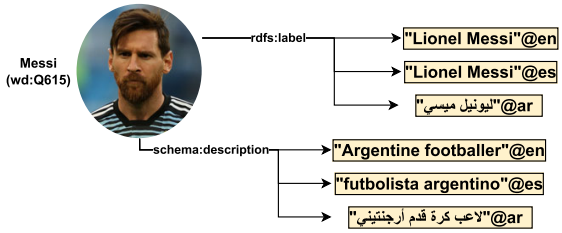


Fig. 5. A subgraph illustrating additional RDF triples for representing (multi-lingual) labels and descriptions of Wikidata entities, leveraging RDF's language-tagged literals.

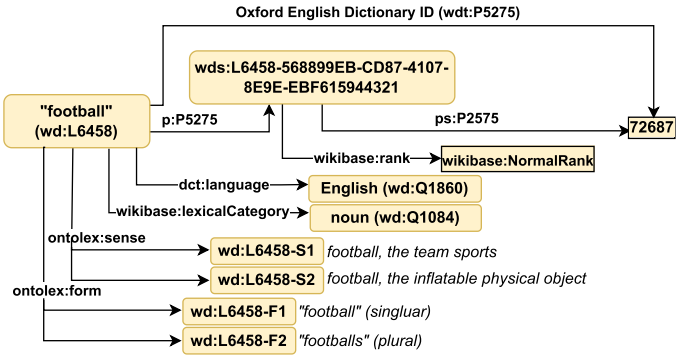


Fig. 6. A subgraph about the English noun "football", including normal claims, but also Wikidata-specific additional vocabulary to talk about languages.

Example 5. As illustrated in Fig. 6, the lexeme *football* (L6458), is an English noun, with different senses – such the team sports (L6458-S1) and the physical object (L6458-S2) – and forms – such as its singular (L6458-F1) and plural (L6458-F2) forms.

As we can see in the example, lexemes can be involved in normal (*wdt*:) statements as discussed before, such as carrying external identifiers in particular dictionaries, but also involve lexeme-specific statements for identifying the

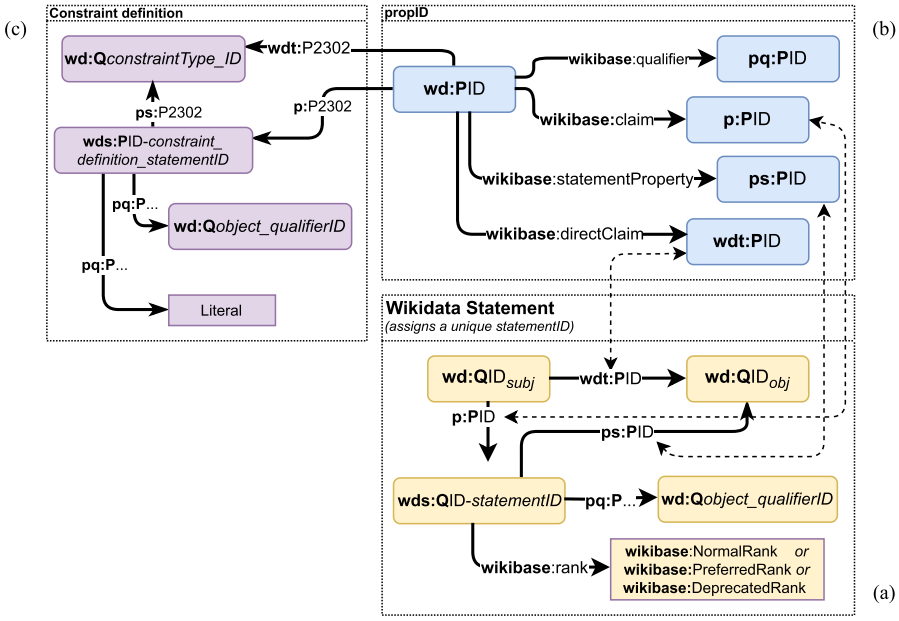


Fig. 7. The Wikidata meta-model in RDF and its namespaces usage illustrated for (a) statements and claims, (b) properties, and (c) property constraint definitions. Dashed lines represent equivalent entities. Figures 2 and 3 illustrate concrete instantiations of the "Wikidata Statement" block (a), while Fig. 8(c) illustrates the block "property constraint definition" block (c). Abbreviations: QID = entity ID, PID = property ID.

language (*dct:language*), category (*wikibase:lexicalCategory*) and lexicographic forms and senses (prefix *ontolex*:). Note that these extra statements have no statement nodes nor qualifiers, somewhat deviating from the standard Wikidata statement model. Also observe, as opposed to language-tagged literals for labels and descriptions, these special statements about lexemes represent the language explicitly as an item (in our example Q1860 for *English*).

Figure 7(a) summarizes the modeling of regular statements and ranks, including the involved namespaces, in a more abstract manner. As shown in Fig. 7(b), the RDF model contains also triples to "navigate" between the differently prefixed URIs per property ID (PID); we will need to make use of these connections in our modeling of constraints in SHACL and SPARQL later on, but let us first turn to how these constraints themselves are actually represented within RDF model.

2.2. Property constraints modeling by example

Wikidata property constraints make use of the described modeling to represent specific community-defined constraint types, where specific instantiations of a *constraint type* are defined as qualified statements on a particular property that should fulfill this constraint. To date, Wikidata represents **32** property constraint types as *subclasses* (P279) of *property constraint* (Q21502402). Table 2 gives an overview of all the constraint types.

Whereas each constraint type is modeled as an item – for instance, the *item-requires-statement (IRS) constraint* (Q21503247), such constraint types are instantiated and parameterized specifically *per property*. That is, each such instantiation is defined by a *constraint-definition-statement* linked to the respective *constrained property* P via the *property constraint* (P2302) property, as illustrated abstractly in Fig. 7(c).

In terms of parameters, constraint-type specific *property qualifiers* are used on the constraint-definition-statement: the overview in Table 2 list all property qualifiers that can be used per constraint type, as well as the number of different properties that use each constraint type (from February 2023).

For instance, the *item-requires-statement (IRS) constraint type* (Q21503247), is used to specify that each item with the constrained property P should also have another given property P'. The constraint is parameterized through the qualifiers

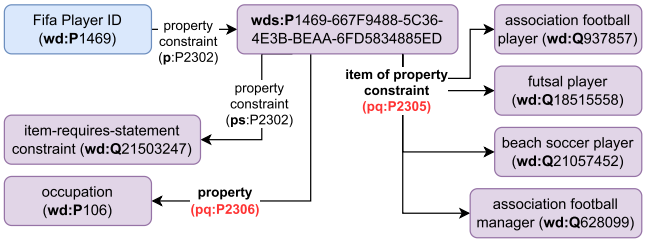
Table 2

Wikidata property constraints types: incl. information about their usage in constraint definitions, information about whether and how we could express them in SHACL-Core and SPARQL, as well as which qualifiers they use (verified on the status at the writing of the paper using a variation of this query: https://w.wiki/7KrH; the short links in the SPARQL column refer to our direct links into our Github repository, available at https://github.com/nicolasferranti/wikidata-constraints-formalization/: besides the SPARQL formalizations you also find all corresponding SHACL shapes (where expressible) there

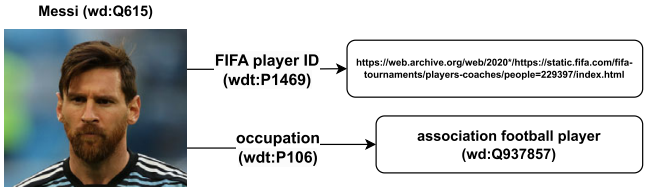
ID	Name	SHACL-Core	PropCount	SPARQL	Qualifiers used in descriptions
Q52004125	allowed entity types	Partially	9657	Partially (https://short.wu.ac.at/pdhs)	P2303;P2305;P2316;P4680;P6607
Q21510851	allowed qualifiers	Not reasonably expressible	819	https://short.wu.ac.at/6tnv	P2241;P2303;P2304;P2306;P2316;P6607
Q21514353	allowed units	Yes	509	https://short.wu.ac.at/ntff	P2303;P2305;P2316;P6607
Q54554025	citation needed	Yes	370	https://short.wu.ac.at/fty7	P2303;P2316;P6607
Q21510852	Commons link	Yes	89	https://short.wu.ac.at/4c5k	P2307;P2316
Q21502838	conflicts-with	Yes	1181	https://short.wu.ac.at/6hsf	P2303;P2304;P2305;P2306;P2316;P6607;P6824;P9729
Q25796498	contemporary	Yes	128	https://short.wu.ac.at/53m9	P2303;P2316;P6607
Q111204896	description in language	Yes	7	https://short.wu.ac.at/z759	P424;P2316
Q21510854	difference-within-range	No	9	https://short.wu.ac.at/cqrs	P2303;P2306;P2312;P2313;P2316;P4680;P6607
Q21502410	distinct-values	Partially	7462	https://short.wu.ac.at/6u73	P2303;P2304;P2316;P4155;P6607
Q21502404	format	Yes	7690	https://short.wu.ac.at/vfwt	P1793;P2241;P2303;P2316;P2916;P4680;P6607
Q52848401	integer	Yes	183	https://short.wu.ac.at/8f39	P2303;P2316
Q21510855	inverse	Yes	125	https://short.wu.ac.at/wsz9	P2241;P2303;P2306;P2316;P4680;P6607
Q21503247	item-requires-statement	Yes	4171	https://short.wu.ac.at/rmba (only req. prop.) https://short.wu.ac.at/72ng (also req. val.)	P2241;P2303;P2304;P2305;P2306;P2316;P4680;P6607
Q108139345	*label in language*	Yes	762	https://short.wu.ac.at/cnfx	P2316;P424
Q55819106	*lexeme requires language*	Yes	172	https://short.wu.ac.at/68dk	P2305;P6607
Q55819078	*lexeme requires lexical category*	Yes	13	https://short.wu.ac.at/mne4	P2305
Q64006792	*lexeme value requires lexical category*	Yes	1	https://short.wu.ac.at/fcke	P2305
Q21510857	*multi-value*	Yes	36	https://short.wu.ac.at/8t2p	P2304;P2316;P6607
Q51723761	no-bounds	Yes	76	https://short.wu.ac.at/f7dw	P2303;P2316
Q52558054	none-of	Yes	115	https://short.wu.ac.at/48jf	P2303;P2304;P2305;P2316;P6104;P6607;P6824;P97
Q21510859	one-of	Yes	222	https://short.wu.ac.at/hejz	P2241;P2303;P2305;P2316;P6607
Q52712340	one-of qualifier value property	Yes	10	https://short.wu.ac.at/nj53	P2305;P2306
Q53869507	property scope	Yes	9912	https://short.wu.ac.at/pzs3 (as main value) https://short.wu.ac.at/wxth (as qualifier) https://short.wu.ac.at/scxh (as reference)	P2303;P2304;P2316;P4680;P5314;P6607
Q21510860	range	Yes	358	https://short.wu.ac.at/tyvx (for values) https://short.wu.ac.at/9ggg (for dates)	P2303;P2310;P2311;P2312;P2313;P2316;P6607
Q21510856	required qualifier	Yes	391	https://short.wu.ac.at/ym6e	P2241;P2303;P2304;P2306;P2316;P4680;P6607
Q52060874	single-best-value	No	207	https://short.wu.ac.at/ad6r	P2303;P2316;P4155;P4680;P6607
Q19474404	single-value	Partially	7254	https://short.wu.ac.at/8ds8	P2241;P2303;P2304;P2316;P4155;P4680;P6607

Table 2
(Continued)

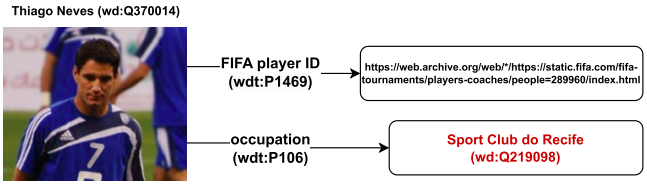
ID	Name	SHACL-Core	PropCount	SPARQL	Qualifiers used in descriptions
Q21510862	symmetric	Yes	48	https://short.wu.ac.at/7hpr	P2303;P2316
Q21503250	subject type	Yes	6860	https://short.wu.ac.at/v7uz (instanceOf) https://short.wu.ac.at/wvzt (subclassOf) https://short.wu.ac.at/eydg (instanceOrSubclassOf)	P2241;P2303;P2304;P2308;P2309;P2316;P4680;P6607
Q21510864	value-requires-statement	Yes	230	https://short.wu.ac.at/4nht (only req. prop.) https://short.wu.ac.at/aadh (also req. val.)	P2241;P2303;P2304;P2305;P2306;P2316;P4680;P6607
Q21510865	value-type	Yes	1077	https://short.wu.ac.at/3akc (instanceOf) https://short.wu.ac.at/xaed (subclassOf) https://short.wu.ac.at/7pv9 (instanceOrSubclass)	P2303;P2304;P2308;P2309;P2316;P6607



(a) Wikidata property constraint representation format exemplified for an instantiation of an *item-requires-statement* constraint; qualifiers to parameterize the constraint are highlighted in red while the color of the other entities are according to their role presented in Figure 7(c)



(b) Data graph complying with the constraint



(c) Data graph not complying with the constraint

Fig. 8. Example of a Wikidata property constraint and data graphs with different behaviors (as of 2022-03-29).

– *property* (P2306), defining the required additional property P', as well as
– *item of property constraint* (P2305), which, if provided, contains permitted values for P'.

Example 6. Fig. 8a illustrates how these qualifiers are concretely instantiated for an IRS constraint the property $P =$ P1469, *FIFA player ID*: this instance of IRS constraint states that if an item has a *FIFA player ID* (P1469), this very

same item should also (i) have an *occupation* (P106), (ii) with one of the following four items: *association football player* (Q937857), *futsal player* (Q18515558), *beach soccer player* (Q21057452), or *association football manager* (Q628099); (i)+(ii) are defined through the resp. qualifiers P2306+P2305 specific to the particular constraint type. That is, firstly, the triple

```
(wd:P1469,p:P2302,wds:P1469-667F9488-5C36-4E3B-BEAA-6FD5834885ED)
```

connects the property *FIFA player ID* (wd:P1469) to its constraint-definition-statement via the property *property constraint* (P2302).

Further, the IRS-specific qualifier *property* (pq:P2306) is bound to *occupation* (wd:P106) through the triple:

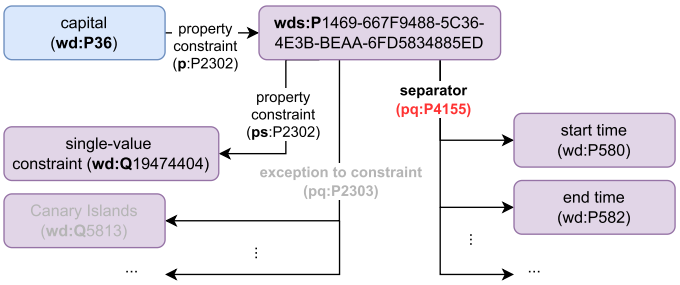
```
(wds:P1469-667F9488-5C36-4E3B-BEAA-6FD5834885ED,pq:P2306,wd:P106)
```

whereas the respective allowed values are defined via four additional triples using the *item of property constraint* (pq:P2305) qualifier, similarly illustrated in Fig. 8a.

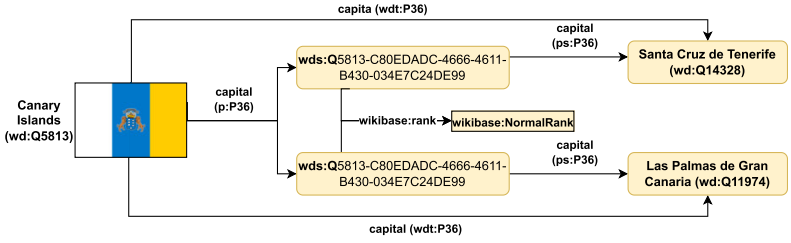
Figures 8b and 8c presents data subgraphs for two different items, *Messi* (Q615) and *Thiago Neves* (Q370014): both have a *FIFA player ID* but only the first one complies with the IRS constraint, having a valid *occupation*, whereas the second one violates it.

As a second example, let us look at another constraint type, the so-called *single-value constraint* (Q19474404), which imposes that property P is only allowed to have one single value unless there are different values for at least one separator, parameterizable by the *separator* (pq:P4155) qualifier.

Example 7. As illustrated in Fig. 9, the property P = P36 (capital) instantiates a *single-value constraint* (Q19474404). As shown in Fig. 9a, several *separator* (pq:P4155) qualifiers can be declared as parameters, only one of which needs to differ, in order to fulfill the constraint, despite non-single values for P. The item *USA* (Q30), shown in Fig. 3, therefore complies with this constraint since the two capitals have different *start times* (pq:580). Figure 9 also illustrates, another "feature" of property constraints modeling in Wikidata, exceptions, which we will turn to next.



(a) Wikidata property constraint representation format exemplified for an instantiation of a *single-value* constraint on the property capital.



(b) Data graph for the Canary Islands – an exception to the constraint

Fig. 9. Another example of a Wikidata property constraint and data graphs (as of 2022-08-20).

Exceptions to constraints Using the dedicated qualifier property *exception to constraint* (P2303), an instantiation of a constraint on a specific property can explicitly mention exceptions. These are items which, for various reasons may be valid, despite violating the constraint.

Example 8. The single-value constraint on P36 in Fig. 9a, lists (amongst others) the *Canary Islands* (wd:Q5813) as an exception of the single-value constraint on *capital* (P36), since it has two co-capitals. Therefore, the data graph in Fig. 9b should, while not complying with the constraint, be considered an "allowed" violation.

2.3. Constraint qualifiers

We hope that the previous subsection has sufficiently illustrated the most relevant aspects of modeling and parameterizing property constraints. Rather than in terms of fully elaborated examples, let us summarize all mentioned and remaining qualifiers used in the context of constraint modeling and parameterization in the following. To this end, Table 2 provides an overview of which qualifiers are used in current descriptions of constraints of different types. For each of the used qualifiers, we will provide a description of how they are used in the context of the different constraint types listed in Table 2, along with specific constrained properties P, also mentioning concrete usage examples. We present these qualifiers in three overall groups:

Core constraint qualifiers (Section 2.3.1), which are essential for modeling the semantics of constraints and for verifying them; i.e., these will be essential for our formalization in SHACL and SPARQL.

Constraint exception qualifiers (Section 2.3.2), which essentially mark concrete items as exceptions to constraints or deactivate whole constraints, that do not need to be verified.

Descriptive constraint qualifiers (Section 2.3.3), which have no semantic relevance for formalizing the (verification of) constraints as such, but serve other, mostly descriptive purposes.

2.3.1. Core constraint qualifiers

- ***item of property constraint* (P2305)**: Lists items expected as values, depending on the constraint type, of either
  - Case 1: P itself, or
  - Case 2: another path P' from the subject of P

  Usage examples: as an example for Case 1, the *one-of constraint* (Q21510859) on the property *sex or gender* (P21) uses qualifier P2305 to declare a list of allowed genders. Example 6 provides an instance of Case 2, where the *item requires statement* (Q21503247) constraint restrict the values of property P' = P1469 defined via qualifier P2306. Example 3 above illustrates another instance of Case 2: *allowed units constraint* (Q21514353) use a P2305 qualifier to restrict the values of the path $P' = p{:}P/psv{:}P/wikibase{:}quantityUnit$ to specific quantity units; for instance, there is an allowed unit constraint on the property *height* (P2048), which allows amongst others the unit *centimetre* (Q174728) – Fig. 4 illustrates this path.
- ***property* (P2306)**: used to define P' as a property used to test for the existence of a path starting at the subject or object of a constrained property P; it is optionally combined with the *Item of property constraint* (P2305), in order to also restrict the *values* of path P'.
  Usage examples: Besides our IRS constraint from Example 6 above, where P2305 is used to set to P' = P1469 as mentioned above, as another example *allowed qualifier* (Q21510851) constraints restrict the usage of certain qualifier properties on the statement nodes of values of P itself using P2306; for instance, an allowed qualifier constraint on property *property constraint* (P2302) itself restricts the usage of qualifiers on constraint-definition-nodes to exactly those listed here. Further, the *conflicts-with* (Q21502838) constraint uses both P2306 and also P2305, to disallow conflicting properties P' (and potentially also their values) that conflict with P statements.
- ***format as a regular expression* (P1793)**: used only by the *format constraint* to express that the value of the constrained property P should be a literal value complying with a predefined regular expression. We note that similar to P2305, nothing prevents this qualifier from also being used in property constraints (in combination with P2306) to restrict values of another path P' in the future, but we have not seen such usage yet.
  Usage example: the property *Spotify user ID* (P11625) uses a *format constraint* (Q21502404) to restrict its value to conform to the specific regular expression "[a-z\d_.-]+", matching sequence of one or more characters that can be lowercase letters, digits, underscores, periods, or hyphens.

– ***language* (P424)**: used by *label in language* (Q108139345) as well as by *description in language* (Q111204896) constraints, this qualifier is used to ensure the existence of either

  * a label (i.e., the path $P' = rdfs{:}label$), or, resp.,
  * a description (i.e., $P' = schema{:}description$)

  for the subject items of constrained property P in a particular language.
  Usage example: the property *Library of Congress authority ID* (P244) uses a *label in language* (Q108139345) constraint to ensure subjects have an English label.
  As an interesting side observation, we note that the similar-in-spirit *lexeme requires language* (Q55819106) constraint rather uses the *item of property constraint* (P2305) qualifier to specify the required language, inline with the different modeling of languages for labels and descriptions vs. lexemes, illustrated in Figs 5 and 6.
– ***separator* (P4155)**: a qualifier property used by constraints that aim to check the uniqueness of the subject or object of a statement for a given constrained property P. When a respective unique statement is expected but multiple are found, a separator can be used to distinguish such conflicting values, which can be understood as a composite key to uniquely identify statements [7] based on the qualifier values only. Therefore, the non-uniqueness of all separators' combinations should be tested to flag a violation.
  Usage examples: as an illustrating example we have already discussed the separator qualifier's use in *single-value* (Q19474404) constraint, such as the instantiation on the property *capital* (P36) from Fig. 9: here, multiple values are allowed as long as they have different combinations of the *start time* (P580) and *end time* (P582) qualifiers.[16] Other constraint types using this qualifier similarly include the *single-best value* (Q52060874) constraint, as well as the *distinct-values* (Q21502410) constraint; as a side note, the latter resembles the constraint reading of an inverse-functional property in OWL, i.e., a value unique for this property over all Wikidata entities.
– ***class* (P2308) and *relation* (P2309)**: Relation and Class qualifiers are used together in *subject type* (Q21503250) and *value-type* (Q21510865) constraints. Here, P2309 represents the expected relationship between the subject (for subject type) or object (for object type) to the set of items described by *Class* (P2308). The possible values for P2309 are:[17] *instance of* (Q21503252), *subclass of* (Q21514624), and *instance or subclass of* (Q30208840).
  Usage example: For instance, property *date of birth* (P569) has a *subject type* (Q21503250) constraint, defining that the subjects of this property should be an *instance of* (Relation) *human* (Q5), *fictional character* (Q95074), or other "classes" of living beings. Here, according to the description of the subject type property constraint,[18] the "*instance of* (Q21503252)" relation should be interpreted as either being a direct *instance of* (P31) or *subclass* (P279) of an instance of. The interested reader will note the resemblance to (the constraint reading of) an *rdfs:domain* statement, whereas the value-type constraint is (analogously) resembling RDFS' *rdfs:range* statements; we will get back to that later.
– **Range checking qualifiers**: *minimum value* (P2313), *maximum value* (P2312), *minimum date* (P2310), and *maximum date* (P2311): these all describe ranges of values or dates and are used within two constraint types, namely *range* (Q21510860) and *difference-within-range* (Q21510854) constraints, which restrict either the value of P, or the difference to the values of another property P', denoted by the above mentioned P2306 qualifier.
  Usage examples: As an example, the property *atomic number* (P1086), representing the number of protons in an atom's nucleus, has a *range constraint* limiting the values between 0 and 155 using the P2313 and P2312 qualifiers. Likewise, a *difference-within-range constraint* can be found on property P = P570 (*date of death*)

[16] We note that the actual definition of the single-value constraint on P36 on Wikidata lists even more separator properties.
[17] I.e., note that P2309 is itself restricted by a *one-of* constraint, which restricts values of a property to single values of an enumeration.
[18] https://www.wikidata.org/wiki/Help:Property_constraints_portal/Subject_class

to be within −1 and 150 years[19] after the above-mentioned (*date of birth*) property $P' = \text{P569}$, referred to with the P2306 qualifier.

– ***property scope* (P5314)**: defines the "scope" where the constraint should be checked. Such scopes are exclusively used in the *property scope constraint* (Q53869507) type and can be identified according to the namespace used by a property in a triple, distinguishing three cases:

  * Case 1: "as main values", i.e. using the namespaces for claims (*p*:, and for truthy statements (*wdt*:)
  * Case 2: "as qualifiers", i.e. using the *pq*: namespace
  * Case 3: "as reference" (i.e., only in reference statements about claims, identifiable via the *pr*: namespace

  Usage example: the property *reference URL* (P854) – which we saw used in Fig. 4 – uses a *property scope constraint*, scoping this predicate to be always be used *as reference*, i.e., exclusively with the *pr*: prefix.

2.3.2. Constraint exception qualifiers

– ***exception to the constraint* (P2303)**: this qualifier is used to mark a set of Wikidata entities as an exception to the constraint, i.e., that should not be tested for violation, despite using predicate P.
  Usage Example: cf. Fig. 9 above; In principle this qualifier is applicable to all constraint types, i.e., all constraints could list such exceptions, as we can see in Table 2 (last column) though, not all qualifiers have exceptions (i.e., we did not find usage of this qualifier in all constraint types).

– ***reason for deprecated rank* (P2241)**: typically in combination with a *wikibase:DeprecatedRank* on the P2302-linked constraint-definition-statement,[20] this qualifier used to express, that a constraint is deprecated and indicates the particular deprecation reason. Possible reasons include *obsolete* (Q107356532) and *constraint provides suggestions for manual input* (Q99460987). Similar to P2303 for noting exceptions, this qualifier is – in principle – applicable to any constraints, and we indeed see it used in Table 2 with many constraint types.
  Usage Examples: Previously, entities having *place of birth* (P19) should also have *sex or gender*, but currently, this restriction is deprecated, with reason Q99460987 ("constraint provides suggestions for manual input"): constraints of this nature should not be checked or enforced, since the constraint should be rather interpreted as a suggestion than enforcing/restricting certain values.

2.3.3. Descriptive constraint qualifiers

– ***group by* (P2304)**: a qualifier used to group violations in specific groups of Wikidata's reports.
  Usage Example: for instance, for the property *population* (P1082) a group by qualifier specifies that violations for the *allowed qualifier* (Q21510851) constraints should be grouped by *country* (P17), as visible in in Wikidata's database reports.[21]

– ***constraint status* (P2316)**: represent the severity degree of a constraint as *mandatory* (Q21502408), *suggested* (Q62026391), or normal (no declared constraint status).[22]
  Usage Example: For example, the *subject type* constraint on property *academic degree* (P512) states that it is *mandatory*, that all the subjects be an *instance of human*, *fictional character*, or *person*. On the contrary, the above-mentioned *subject type* constraint on *date of birth* (P569), does not have an explicit status (and therefore should be considered a "normal" constraint).

– ***constraint clarification* (P6607)** and ***syntax clarification* (P2916)**: both represent text descriptions for constraints. *Constraint Clarification* was originally created to describe the purpose of the constraint for a property. We note that, while we see it used in the context of most constraint *types* (cf. Table 2) – at the time of writing –

[19] *year* (Q577) is again a type of quantityUnit in Wikidata, similar to the "centimetre" unit mentioned in Fig. 4. While we do not detail unit conversion with SHACL and SPARQL, or likewise, interval duration computation between dates in the present work, we refer to preliminary work under submission for the Wikidata 2023 workshop [60] that points in this direction, proposing a resp. query rewriting approach.

[20] Just like any other "regular" claims about Items, constraint-definition-statements about Properties also have a *wikibase:rank*; notably, at the time of writing, we found 5 constraint definitions using qualifier P2241, while actually having a non-deprecated rank, cf. https://w.wiki/7MW9.

[21] https://www.wikidata.org/wiki/Wikidata:Database_reports/Constraint_violations/P1082

[22] Notable, at the time of writing, there were 17 property constraint definitions with a different status, cf. https://w.wiki/7KfY, and indeed the allowed statuses are not restricted by a constraint themselves.

only 865 out of 72,339 constraint definitions in total actually had such a clarification.[23] It is also used to describe suggested repairs in textual form. *Syntax Clarification* provides a textual description of the regex syntax of a value and is to be used in combination with the above-mentioned *format as a regular expression* (P1793) qualifier as documentation.
Usage Examples: For instance, a *none-of constraint* (Q52558054) for *country of citizenship* (P27) lists several "Pokemón regions", such as *Kanto* (Q1657833), that should not be used, where the additional P6607 clarifies that "It's not always clear in Pokémon canon if regions are part of a larger country or are countries in and of themselves."

– ***replacement property* (P6824) and *replacement value* (P9729)**: Both these qualifiers represent suggestions to fix inconsistencies, rather than to test/verify them; while *replacement property* suggests the replacement of constrained property P by another property, *replacement value* recommends using a specific value (a Wikidata concept or literal) instead of the current one.
Usage Example: for instance, the property *country* (P17) uses a *conflicts-with constraint*, in combination with *item of property constraint* (P2306) qualifiers listing values

  * *musical group* (Q215380)
  * *musical ensemble* (Q2088357)
  * *musical duo* (Q9212979)
  * *musical trio* (Q281643)

  stating that entities that are instances of these values[24] should not use *country*, but – using the the mentioned *replacement property* (P6824) qualifier – rather the property *country of origin* (P495).[25]

2.4. Additional challenges in understanding and verifying the semantics of constraints

The above summary of the used qualifier properties to parameterize constraints should have illustrated that the semantics of Wikidata's vocabulary used to describe constraints are not always uniquely determined: indeed, the interpretation of constraint qualifiers depends on (i) in which context (ii) in which particular combination with other qualifiers, they are used (iii) in particular constraint types.

Before we continue in Section 3 with more details on how these qualifier properties are interpretable as parameters in SHACL-Core shapes for verifying different constraint types, let us discuss some additional challenges that potentially complicate these formalizations, and motivate our idea to design bespoke translations per *constraint type*.

The above description of Wikidata property constraints modeling in RDF defines how constraints are represented but not how they should be checked. To understand how to check constraints, a description property (*schema:description*) is provided along with the *described at URL* (P973) property, indicating a link to a page that describes the constraint. The vast majority of Wikidata constraint types (more precisely, 28 out of 32) have such a "Help"-page .[26] As these pages contain descriptions in natural language, they are often subject to ambiguity and different interpretations. Indeed, one of the main challenges when formalizing constraint types was the understanding of the semantics behind the constraint.

It is important to note here that the Wikidata community has defined all existing property constraint types in a manner where these types and their modeling have grown organically. In our formalizations of such constraints, we tried to stay as close as possible to the – partially heterogeneous – interpretations derivable from the natural language descriptions of constraint types. We could find and document several cases of textual ambiguity, leaving room for different interpretations and as a result, for different implementations of the respective constraint checks. We illustrate some of these issues.

---

[23] https://w.wiki/7Kgs

[24] i.e., additionally using qualifier *property* (P2305) restricting P' to *instance of* (P31).

[25] Interestingly, other sub-types of musical ensembles, such as *string quartet* (Q207338) are not explicitly listed in this conflicts-with constraint.

[26] The description page of our running example *item-requires-statement* is available at https://www.wikidata.org/wiki/Help:Property_constraints_portal/Item.

2.4.1. (Non-)consideration of subclasses and subproperties

Let us first note that in Wikidata constraint type descriptions, it is rarely explicitly/uniformly specified whether *subclass of* (P279) relationships should be interpreted transitively, or whether *instance of* (P31) relationships should also affect instances of (transitive) subclasses.

Indeed, some of these constraints resemble known RDFS axioms: for instance, the *subject type constraint* (Q21503250) is logically equivalent to the constraint reading of an *rdfs:domain* statement, which intuitively poses restrictions on the entities allowed in the domain of the property P. Analogously, the *value-type constraint* (Q21510865) resembles *rdfs:range*.

For instance in our formalizations of the *subject type constraint* (Q21503250), and likewise the analogous *value-type* constraint, we took a choice interpreting the relationship (*pq:2309*) *instance of* (Q21503252) as a property path *wdt:P31/wdt:P279**, i.e., including transitive "subclass of"-reasoning. Note this is similar to the RDFS encoding by query rewriting in [11]. This particular interpretation was driven by the following natural language description on a separate Help-page for Q21503250:[27]

> *"Subclass relations according to* subclass of *(P279) are taken into account: if a constraint demands that an item should be an instance of* building *(Q41176), it is not a violation if the item is an instance of* skyscraper *(Q11303), because there is a* subclass of *(P279) path from* skyscraper *(Q11303) to* building *(Q41176). (If an indirect relation should not be permitted,* item-requires-statement constraint *(Q21503247) can be used.)"*

The interested reader might have noted the last sentence in parentheses, which indirectly informed our respective interpretation of *IRS constraints* (Q21503247). Here we did **not** consider subclasses of instances, in case an IRS constraint via qualifier (P2306) requiring the subject to be an *instance of* (P31) a particular value (via qualifier P2305). Interestingly, at the time of writing, more than 1000 IRS constraints refer to property P31 (via the P2306 qualifier) and a specific class:[28] whether each of the authors of these IRS constraints was aware of the implicit choice to only consider direct instances here, and no instances of subclasses, remains unclear to us.

This potential issue is, by the way, not restricted to IRS constraints, cf. Footnote 25 on p. 2349, which illustrates a similar questionable example for the potential non-consideration of subclass-inferencing in the context of a *conflicts-with* constraint.

Along these lines, similar issues of interpretation may arise, in the context of (sub-)properties (P). Recall the *allowed qualifiers constraint* (Q21510851) states that when using a particular property, only a limited set of properties can be used *as qualifiers* through the reification mechanism: for instance, the *capital* (P36) property has an allowed qualifiers constraint permitting *start time* (pq:580) and *end time* (pq:582) qualifiers.

Yet, the claim that *Stralsund* (Q4065) was the capital of *Swedish Pomerania* (Q682318) until 1815 is considered a violation of this constraint since the *temporal range end* (P524) qualifier is used to mark the end of the period. The Wikidata UI reports a violation, although P524 is a subproperty of P582, i.e.,

```
(wd:P524, wdt:P1647, wd:P582)
```

We consequently do not consider (transitive) subproperty relationships in our formalization, in line with the observed behavior within Wikidata: indeed, although *subproperty of* (P1647) is semantically similar to *subclass of* (P279) when it comes to the hierarchy of properties, the page that describes the *allowed qualifiers* constraint[29] does not mention the use of the subproperty hierarchy.

2.4.2. Interpreting separators

The description page for *single-value constraint* states:[30]

> *"specifies that a property generally only has a single value. [...] A qualifier can be defined as a* separator *(P4155). This allows for multiple values when using such qualifiers. [...] If specified, multiple statements with*

[27] https://www.wikidata.org/wiki/Help:Property_constraints_portal/Subject_class
[28] https://w.wiki/7Gd4
[29] https://www.wikidata.org/wiki/Help:Property_constraints_portal/Qualifiers
[30] https://www.wikidata.org/wiki/Help:Property_constraints_portal/Single_value

the same value do not constitute a violation of this constraint as long as they have different qualifiers for the properties specified here."

We emphasize that this leaves room for interpretation. I.e., possible interpretations include:

I1 Values to be considered different if they use different values for *all* (common) qualifier properties.
I2 Values to be considered different if they use different values for *some* (common) qualifier.

indeed, the latter interpretation (**I2**) is the correct one as we found out mostly experimentally, checking respective (non-)violations on specific Wikidata items.

2.4.3. Handling exceptions, deprecated, or suggested constraints

It is arguable whether we should consider exceptions – marked as *Exception to the constraint* (P2303) – in our formalisation: while exceptions are indeed not marked as violations in the UI, we cannot for sure determine whether they are counted in the Wikidata Database Reports: the reports only contain aggregated counts of constraint violations, but not all single violations are linked from the database reports pages. We will therefore discuss handling of exceptions as an optional "feature" in our formalization.

Deprecated Constraints are, interestingly, still being tested and reported by Wikidata's Database Reports; likewise, no distinction is made on the constraint status – denoted by *constraint status* (P2303) – in the reports; our formalization therefore will consider violations independent of their status. As an interesting side note, we refer to the fact that there are various constraint definitions that report a *reason for deprecated rank* (P2241) while the constraint definitions themselves have a non-deprecated rank.[20] We consequently decided not to treat deprecated constraints, nor constraints with a non-normal status in any special way in our formalisation.

2.4.4. Differences in Wikidata RDF serialization

Another remarkable finding arose for us when taking a closer look at *allowed entity types* (Q52004125) constraints; as per its description, this type of constraint limits the subject of a respective property to certain listed *entity types*, such as:

- Wikibase Item (`wikibase:Item / wd:Q29934200`);
- Wikibase property (`wikibase:Property / wd:Q29934218`);
- Wikibase MediaInfo (`wikibase:MediaInfo / wd:Q59712033`);
- Lexeme (`ontolex:LexicalEntry / wd:Q51885771`);
- Sense (`wikibase:Sense` / `ontolex:LexicalSense` / `wd:Q54285715`);
- Form (`wikibase:Form` / `ontolex:Form` / `wd:Q54285143`); or also
- Wikidata Item (`wd:Q16222597`).

Notably, though, these types can only be partially checked, depending on the RDF serialisation, for various reasons:

- incoherent serialisation with respect to different entity types;
- differences between RDF serialisations on the SPARQL endpoint vs. RDF dumps;
- checks requiring unintuitive workarounds

Incoherent serialisation with respect to different entity types : neither the entity types that are part of the Wikibase base ontology (`wikibase`:-prefixed nor the `wd`:-prefixed types) appear consistently in neither Wikidata's RDF export nor on the Wikidata endpoint. For instance, while, on the one hand, the query

```
SELECT * { ?s a wikibase:Property }
```

returns over 10000 results (apparently covering all properties), on the other hand, the query

```
SELECT * { ?s a wikibase:Item }
```

returns 0 results on the Wikidata SPARQL endpoint.

Differences between RDF serialisations: In addition, interestingly, there are apparently even differences in Wikidata's RDF serialization via RDF dumps (cf. Footnote 12) vs. the RDF triples served on the public SPARQL endpoint. For instance, as manually verified, Wikidata's RDF latest N-Triples RDF dump did not contain any triples using the `ontolex:` prefix, whereas on the SPARQL endpoint, apparently respective triples are present.[31]

Checks requiring unintuitive workarounds The documentation of Wikdata's query service[32] suggests some alternative workarounds to match certain Wikibase entity types, arguing that the respective direct triples have been omitted for performance reasons: for example, to filter for `wikibase:Senses` in the Wikidata query service, it is necessary to use `ontolex:LexicalSense` instead. As mentioned above, this workaround does not function on the RDF dumps, where `ontolex:LexicalSense` is also omitted. As a second example of such a workaround, the documentation suggests to "*use `wikibase:sitelinks []` instead of a `wikibase:Item`*" in order to filter for items on the SPARQL query service, since "`only items have a number of sitelinks`".

Indeed, these subtleties and incoherence within Wikidata's RDF serialization(s) make it hard to define generic parameters to test allowed entity types constraints automatically, forcing case-by-case implementations (either in terms of SHACL shapes or in a SPARQL query), depending on whether working on the SPARQL endpoint or on Wikidata's RDF dump.

In the context of this paper, we have designed both our SHACL-Core formalization (cf. Fig. 12 below) of *allowed entity types* (a)s well as the respective SPARQL query (cf. the query referenced in first line of Table 2) to conform with the data available in the Wikidata query service, implementing all documented workarounds.

In summary, all of these examples and issues should motivate the following disclaimer: the SHACL-Core shapes and SPARQL queries proposed in this paper were created from the available descriptions and aim to reduce the margin of interpretation in dealing with Wikidata constraints while keeping as close as possible to the documented interpretations. As such, all our SHACL and SPARQL formalizations discussed in the following Sections 3 and 4 (and linked from Table 2) reflect our best-effort *interpretations* of the respective natural language definitions. Yet, the goal and scope of our work is to contribute to these interpretations in an unambiguous, declarative manner.

3. Expressing Wikidata constraints with SHACL

In this section, we will present Wikidata property constraint types in terms of SHACL shapes, i.e., deploying the official W3C standarised language to express constraints on RDF graphs.

To this end, we will first provide some necessary background on RDF and SHACL, introducing some notions that will be useful in the rest of the section (Section 3.1), whereafter we will dive into details of formalization and expressibility of particular Wikidata property constraint types (Section 3.2), again mostly driven by examples; for a full list of SHACL formalizations per constraint type, we refer to Table 2.

3.1. SHACL validation

The SHACL standard specifies constraints through so-called *shapes*, which are to be validated against an RDF graph. SHACL shapes themselves are represented as RDF Graphs, and may contain a wide variety of *constraint components* that allow the construction of possibly complex expressions to be checked over the input data. As usual in the majority of the SHACL works, in this paper, we focus on the *core* constraint components of SHACL and refer to it as SHACL-Core; an extension to this core language that allows the addition of full SPARQL queries for constraint checking will be introduced later, in Section 3.4.

A *shapes graph* contains shapes paired with *targets* specifying the *focus nodes* in the RDF graph that should be checked for validation. In a nutshell, an RDF data graph *validates* a shapes graph if these targets conform with the constraints specified in the corresponding shape. We illustrate these notions with a first example of a shapes graph implementing an *item-requires-statement* constraint type.

[31] For instance, the SPARQL endpoint returned over 13M instances of `ontolex:Form` at the time of writing, cf. https://w.wiki/8hku.
[32] https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format#WDQS_data_differences

```
1  :P1469_ItemRequiresStatementShape
2      a sh:NodeShape ;
3      sh:targetSubjectsOf wdt:P1469 ;
4      sh:property [
5          sh:path wdt:P106 ;
6          sh:qualifiedValueShape [
7              sh:in (wd:Q937857 wd:Q18515558
8                  wd:Q21057452 wd:Q628099) ;
9          ] ;
10         sh:qualifiedMinCount 1 ;
11     ] .
```

(a)

```
1  :P1469_ItemRequiresStatementShape
2      a sh:NodeShape ;
3      sh:targetSubjectsOf wdt:P1469 ;
4      sh:or (
5          [sh:in(wd:Q370014);]
6          [sh:property [
7             sh:path wdt:P106 ;
8             sh:qualifiedValueShape [
9              sh:in (wd:Q937857 wd:Q18515558
10             wd:Q21057452 wd:Q628099) ;
11            ] ;
12            sh:qualifiedMinCount 1 ;
13         ]]
14     ) ;
```

(b)

Fig. 10. SHACL shape *item-requires-statement* constraint for the (a) *FIFA player ID* (wdt:P1469) and (b) a variant with exceptions.

Example 9. The shapes graph in Fig. 10a describes the shape :*P1469_ItemRequiresStatementShape*, which defines its targets as nodes in the subject of the wdt:P1469 property (line 3), that is, nodes that have a wdt:P1469-outgoing edge. Intuitively, a data graph validates this shapes graph if each target node has at least one wdt:P106-edge to one of the nodes (constants) listed in *sh:in* (lines 7 and 8).

Consider the RDF graph represented by Fig. 8b:[33] this data graph clearly validates the shapes graph. Intuitively, *Lionel Messi* (i.e., wdt:Q615) is the only target node and it has an *occupation*-edge to wd:Q937857 (*association football player*), which is indeed included in the list provided in the constraint.

This is not the case for the RDF graph in Fig. 8c, since *Thiago Neves* has only one *occupation*-edge to a node that is not listed in the constraint. However, this second data graph validates the shapes graph shown in Fig. 10b: toughly speaking, this variant of the original shape relaxes the constraint with an additional sh:or component stating an exception for the target node *Thiago Neves* (wd:Q370014).

Overall, the shapes introduced in Example 9 define exactly the intended semantics of the *item-requires-statement* constraint represented in Fig. 8a, where we already give a hint, that even optional exception handling can be easily expressed in SHACL-Core.

The SHACL specification allows for shapes to refer to other shapes which may result even in cyclic references and *recursive* constraints. In this context, we note that the official specification only provides semantics for non-recursive constraints, therefore, in line with our goal, to create shapes with standard validators, we may consider it as a requirement to avoid such recursive shapes. Indeed, the shapes that we obtain in this paper are all *non-recursive*; the fact that current Wikidata property constraints apply locally (per Subject Item) and do not transitively or even recursively depend on the fulfillment of other constraints, plays in our favor: as our example shows, we can express an *item-requires-statement*, and as we will show most other constraint types, in single shapes.

The constructs defining a shapes graph can syntactically be viewed as concepts in expressive Description Logics [9], a well-known family of decidable fragments of first-order logic. That is, the shape components can be viewed as logical constructs, such as existential and universal quantifications, qualified number restrictions, constants, or regular path expressions. For instance, the shapes graph in Fig. 10a can be expressed as the tuple containing the target $t = \exists \mathsf{wdt} : \mathsf{P1469}.\top$ and DL concept: $\varphi = \exists \mathsf{wdt} : \mathsf{P106}.(\{\mathsf{wd} : \mathsf{Q937857}\} \vee \{\mathsf{wd} : \mathsf{Q18515558}\} \vee \{\mathsf{wd} : \mathsf{Q21057452}\} \vee \{\mathsf{wd} : \mathsf{Q628099}\})$ defining the shape P1469_ItemRequiresStatementShape. The validation test can intuitively be viewed as the concept inclusion $t \sqsubseteq \varphi$ evaluated over the input data graph, that is checking whether the nodes obtained from the evaluation of the target expression t over the data graph are included in the evaluation of the shape expression φ over the data graph. For details on a formal logic-based representation of the syntax and semantics of SHACL-Core, we refer to [21]. The complete list of all SHACL constraint components

[33] Note that the nodes and edges are labeled with both names and their (namespace-abbreviated) URIs, but we assume the corresponding RDF graph is implicitly clear to the reader.

and their semantics can be found in the W3C SHACL specification.[34] We will present and introduce further these components in Section 3.2 as needed.

3.2. Mapping Wikidata constraints to SHACL-Core

Let us proceed to describe more systematically now, how Wikidata property constraints can be translated to SHACL-Core shapes.

The *targets* of property constraints in Wikidata are always the subjects of the resp. constrained property P, i.e., we can use uniformly use SHACL-Core's `sh:targetSubjectsOf` construct, to define target nodes across all our formalizations of various constraint types.

Constraint types *requiring* the existence of a specific statement, such as our running example's *item-requires-statement* constraint (and, likewise *required qualifier* constraints, *one-of* constraints...), are naturally captured by SHACL-Core constraint components. Roughly speaking, these types of constraints can be represented by choosing a target node, verifying the existence of an `sh:path`, and, possibly, verifying the existence of a value using `sh:minCount 1` or a qualified minimum count `sh:qualifiedMinCount 1`, as shown above.

On the contrary, constraints *forbidding* certain statements, such as *conflicts-with*, or likewise *property scope* constraints can modeled analogously with a combination of `sh:path` and `sh:maxCount 0`. As an example let us illustrate a simple *conflicts-with* (Q21502838) constraint for the property *family name* (P734); according to the constraint property, it should not be used together with the property P1560 (given name version for another gender), as expressible concisely in the following SHACL shape:

```
1   :P734_ConflictsWithShape
2       a sh:NodeShape ;
3       sh:targetSubjectsOf wdt:P734;
4       sh:property [
5           sh:path wdt:P1560 ;
6           sh:maxCount 0 ].
```

Further building upon and extending the discussion of Example 9, we discuss and illustrate translations to SHACL guided by the constraint qualifier properties to parameterize them in the following, where we go through the constraint qualifiers in the same order as in Section 2.3.

– ***item-of-property-constraint* (P2305) and *property* (P2305)**: We note that Example 9 clarifies that it is possible to encode allowed values in SHACL, something that was considered uncertain in [55]: indeed, amongst the qualifier parameters in constraints, discussed in all values of the *item-of-property-constraint* (P2305) qualifier, can be turned into an `sh:in` expression (lines 7–8 in Fig. 10a); the respective *property* (P2306) qualifier is captured by an `sh:path`, where the respective path existence requirement for $P' =$ implied by the IRS constraint can be implemented by means of the `sh:qualifiedMinCount 1` restriction on this path (lines 5 and 10 in Fig. 10a). Obviously, arbitrary instantiations of IRS constraints can be turned into SHACL-Core shapes analogously, by collecting the resp. P2305 and P2306 qualifiers as parameters.
  As noted in Section 2.3.1, the P2306 qualifier may also in the context of other constraint types, refer to more generic paths P', likewise expressible through `sh:path` which indeed can represent arbitrary regular paths, for instance, the following shape for representing the afore-mentioned *allowed units* constraint on *height* (P2048), illustrates this in line 5:

```
1   :P2048_AllowedUnitsShape
2           a sh:NodeShape ;
3           sh:targetObjectsOf p:P2048 ;
4           sh:property [
5                 sh:path (psv:P2048 wikibase:quantityUnit) ;
6                 sh:in ( wd:Q174789 wd:Q174728 wd:Q11573 ... ) ;
7       ].
```

[34] https://www.w3.org/TR/shacl/#core-components

where the listed allowed units in line 6 denote *milimtre* (Q174789), *centimetre* (Q174728), *metre* (Q11573), etc.

– ***format as a regular expression*** **(P1793)**): in SHACL, this qualifier can be expressed through the *sh:pattern*, illustrated for the afore-mentioned *Spotify user ID* (P11625) property as follows:

```
1  :P11625_FormatConstraintShape a sh:NodeShape;
2      sh:targetSubjectsOf wdt:P11625;
3      sh:property [
4          sh:path wdt:P11625 ;
5          sh:pattern "[a-z\d_.-]+" ;
6      ].
```

– ***language*** **(P424)**): the following example shape illustrates how this qualifier can be expressed through the `sh:languageIn` construct in SHACL, for the afore-mentioned *Libary of Congress authority* (P244) property as follows:

```
1  :P244_LabelInLanguageShape
2          a sh:NodeShape ;
3          sh:targetSubjectsOf wdt:P244 ;
4          sh:property [
5              sh:path rdfs:label ;
6              sh:qualifiedValueShape [
7                  sh:languageIn ( "en" ) ;
8              ] ;
9              sh:qualifiedMinCount 1 ;
10         ] .
```

– ***separator*** **(P4155)**: To the best of our knowledge, there is no direct equivalent SHACL-Core component to model composite keys, i.e., we cannot directly express respective constraint types such as *single-value* constraints, in case they use the separators; while a simplified version of a SHACL-Core shape to express a *single-value constraint* without separators is illustrated in Fig. 13a below, let us defer the discussion about expressing separators to later for now.

– ***class*** **(P2308) and** ***relation*** **(P2309)**: in general, the SHACL component *sh:class* can be used to check the type of an item, also including hierarchical inference to check instances of subclasses. Yet, the subclasses mechanism is based on *rdfs:subClassOf* and *rdf:type* and requires an adaptation to work with WD's *wdt:P279* and *wdt:P31*: instead, in our formalization we use a `sh:path` together with `sh:hasValue` or `sh:in` for instance to encode *subject type* constraints (where, if you recall, we need to check the path *wdt:P31/wdt:P279** for testing with an *instanceOf* relation), as illustrated by the following example:

```
1  :P569_SubjectTypeConstraintShape a sh:NodeShape;
2      sh:targetSubjectsOf wdt:P569;                          # date of birth
3      sh:property [
4          sh:path (wdt:P31 [sh:zeroOrMorePath wdt:P279]) ; # instance of
5          sh:qualifiedValueShape [
6              sh:in (wd:Q5 wd:Q95074 ... )                   # human, fictional character, etc.
7          ] ;
8          sh:qualifiedMinCount 1 ;
9      ].
```

– **Range checking qualifiers**: in SHACL range restrictions are representable with the `sh:minExclusive` and `sh:maxExclusive` construct for open intervals, and `sh:minInclusive` and `sh:maxInclusive` for closed intervals; we illustrate the use of these in a shape to validate a range constraint on the afore-mentioned *atomic number* (P1086) property:

```
1  :P1086_RangeShape a sh:NodeShape;
2      sh:targetSubjectsOf wdt:P1086; # atomic number
3      sh:property [
4          sh:path wdt:P1086 ; # instance of
5          sh:minInclusive 0 ;
6          sh:maxInclusive 155 ;
7      ].
```

– ***property Scope* (P5314)**: When creating a corresponding SHACL Shape for a constraint using this qualifier, we exploit the finite number of used namespaces in the Wikidata RDF dump, and explicitly disallow the respective non-allowed prefix(es) with separate shapes, with SHACL-Core's `sh:maxCount` construct set to 0; different disallowed uses are conjoined by `sh:and`. For instance, the following shape implements the *property scope constraint* on *reference URL* (P854), disallowing its use both in the *wdt*: and *pq*: variant, respectively:

```
:P854_propertyScopeReference_MainAndQualifierShape a sh:NodeShape ;
    sh:and (
      [
        sh:targetSubjectsOf wdt:P854 ;
        sh:property [
            sh:path wdt:P854 ;
            sh:maxCount 0 ;
        ]
      ]
      [
        sh:targetSubjectsOf p:P854 ;
        sh:property [
            sh:path p:P854 ;
            sh:maxCount 0 ;
        ]
      ]
    ).
```

This concludes the discussion of the treatment of *core constraint qualifiers* (Section 2.3.1) in our translation: essentially, we have created templates for each constraint type using SHACL-Core expressible constraint qualifiers. I.e., these templates can be instantiated with the respective qualifiers used as parameters in an automated fashion. Before we present a prototype implementing this automatic translation in the next subsection (Section 3.3), let us also briefly elaborate on the treatment of *constraint exception qualifiers* (Section 2.3.2) and *descriptive qualifiers* (Section 2.3.3), which, as we will see, also partially can be cast into respective SHACL-Core constructs.

– ***exception to the constraint* (P2303)**: we note that Shenoy et al. [55] argue that it is unclear if SHACL can encode exceptions in property constraints, yet single exceptions to constraints turn out to be easily expressible in SHACL-Core: based on our running IRS example (Example 9), and its initial SHACL-Core formalization depicted in Fig. 10a, let us suppose that Thiago Neves (Q370014) is the single exception to the constraint; as we already discussed a simple combination of the `sh:or` and `sh:in` constructs can emulate such exception, validating the respective excepted target nodes, illustrated in Fig. 10b (lines 4+5). Again, this template "recipe" is applicable to any constraints that list explicit exceptions.
– ***reason for deprecated rank* (P2241) and *wikibase:DeprecatedRank***: Despite SHACL's boolean component *sh:deactivated* would allow disabling a shape, we note that it is not possible to represent specific deprecation reasons with this SHACL property. While similar to the descriptive qualifiers below, we could leverage additional SHACL-Core constructs such as `sh:description` or likewise `rdfs:seeAlso`, we do not treat this qualifier explicitly in our current translations.
– ***group By* (P2304)**: Despite SHACL-Core has a component *sh:group* to indicate that a shape belongs to a group of related property shapes, this construct does not serve to split *violations* found by the same shape into different groups based on a specific property; as such, we deem the P2304 by qualifier, which anyway does not carry semantic meaning in terms of verifying violations, not directly representable in SHACL and do not consider it explicitly in our translation either.
– ***constraint status* (P2316)**: the most similar SHACL construct that can be used to express status in the spirit of P2316 is *sh:severity* which allows one of the three possible severity degrees: *sh:Info*, *sh:Warning*, or *sh:Violation*; we could, for instance mark "mandatory" constraints as `sh:Violation` as follows.

```
:P512_TypeConstraintShape a sh:NodeShape;
    sh:targetSubjectsOf wdt:P512; # academic degree
    sh:property [
        sh:path (wdt:P31 [sh:zeroOrMorePath wdt:P279]) ; # instance of
        sh:in (wd:Q5 wd:Q95074 wd:Q215627) ;
        sh:severity sh:Violation ;
    ].
```

Yet, along the lines of our discussion in Section 2.4.3 above, where we remarked that Wikidata does not seem to make an explicit distinction between constraint statuses in its Database Reports, we also do not consider the *constraint status* (P2316) in our translation.

– **syntax clarification (P2916)** and **constraint clarification (P6607)**: Although these qualifiers do not have any formal meaning, we can use the SHACL component *sh:description* to provide descriptions of the respective property in a given context, in natural language. We illustrate this with the following example:

```
:P27_NoneOfConstraintShape a sh:NodeShape ;
    sh:targetSubjectsOf wdt:P27; # country of citizenship
    sh:not [
        sh:property [
            sh:path wdt:P27 ;
            sh:in (wd:Q36704) ; # Yugoslavia
            sh:description "this state had separate citizenships for different periods: 1918--1929 ...";
        ] ;
    ].
```

– **replacement property (P6824) and replacement value (P9729)**: SHACL components are designed to capture inconsistencies but not to directly provide "hints" to fix them, i.e., this kind of information present in Wikidata property constraints can not be represented with SHACL explicitly; however, we note at this point that this qualifier information could inform choosing/computing specific repairs (as, for instance, recently discussed in [4,5]).

In summary, by "templating" the respective qualifier parameter translations from Wikidata's constraint representation to SHACL shapes based on the illustrating above examples, we can cover most constraint types, and – additionally carry over also some useful descriptive information to dedicated SHACL-Core constructs, that are not strictly needed for validation as such, but can be used by validators to generate explaining output.

A prototype, reading constraint definitions in Wikidata's representation and accordingly creating their shapes representation on-the-fly is presented in Section 3.3 below. Table 2 presents the entire set of analyzed constraint types, their Wikidata IDs, as well as a column to state whether it was possible to map the constraint type to SHACL (and SPARQL, respectively, see Section 4). The particular SHACL encodings, created by using the above-introduced "mappings" between Wikidata qualifiers and SHACL-Core components, can be found in an online GitHub repository accompanying our paper.[35]

3.3. Tool to automatically convert Wikidata constraints to SHACL

In order to demonstrate the feasibility of an automated translation, we developed *wd2shacl*, a demo tool to automatically convert constraints from the Wikidata model to the corresponding SHACL shape (for those that could be represented). This allows for the creation of a large, real-world SHACL benchmark from Wikidata, thus also addressing the scarcity of respective benchmarks for SHACL-Core currently available. Wd2shacl allows testing Wikidata property constraints with SHACL validators and generates verifiable shapes for any Wikidata property, extracting its associated constraints.

In a nutshell, we first generalized the example SHACL shapes (such as the one in Fig. 10a to become templates for specific constraint types, e.g. by replacing the specific qualifier values assigned to a single property, illustrated by the following "template abstraction" for IRS constraints:

```
:P_ItemRequiresStatementShape
    a sh:NodeShape ;
    sh:targetSubjectsOf wdt:P ;
    sh:property [
        sh:path wdt:[pq:P3206] ;
        sh:qualifiedValueShape [
            sh:in ( [pq:P2305] ) ;
        ] ;
        sh:qualifiedMinCount 1 ;
    ] .
```

[35] https://github.com/nicolasferranti/wikidata-constraints-formalization

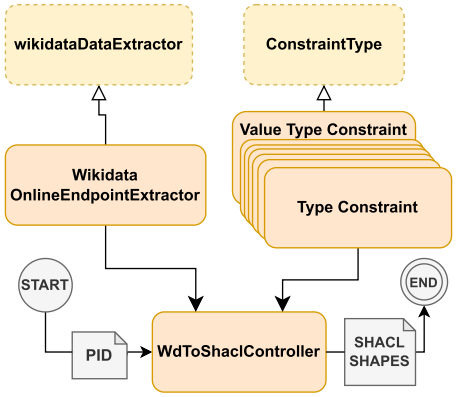


Fig. 11. Wikidata to SHACL architecture. Dashed lines represent abstract classes.

Our wd2shacl tool then populates these templates according to the actual qualifier values instantiated for a specific property P. The architecture of wd2shacl is shown in Fig. 11. As input, we provide the P = PID of the desired property. The controller (WdToShaclController) uses a data extractor to collect all constraint types from the property, as well as the respective qualifiers describing them. The data extractor (wikidataDataExtractor) directly queries the respective qualifier statements from a Wikibase instance via SPARQL, where our current implementation directly uses Wikidata's endpoint (WikidataOnlineEndpointExtractor). Alternative inherited extractor classes can be created to consume data from an RDF dump in a predefined format if necessary (e.g. to query from HDT archived versions of Wikidata[36]), or from another endpoint, for instance, to query alternative Wikibase instances. Indeed, other Wikibase instances, such as the EU Knowledge Graph[37] re-use the Wikidata property constraint mechanism and could be likewise checked using the tool via such alternative extractors.

After collecting constraint types and qualifiers for the input property P, the controller instantiates the template to create respective SHACL constraints for the extracted applicable constraint types. A specific class (a concrete subclass of ConstraintType) is implemented for each SHACL-expressible constraint type to create the SHACL shape by combining the template with the given qualifiers. The controller returns the populated SHACL templates as SHACL Turtle files, containing the required prefixes and a list of SHACL shapes, written using SHACL-Core language, for all the translatable constraint types associated with the input PID. The tool is freely available online within our GitHub repository.[38]

3.4. *Limitations of SHACL-Core for checking Wikidata constraints*

As shown in Table 2 the vast majority of Wikidata constraint types can be rewritten into SHACL-Core Shapes (26 out of 32); yet, three could only be partially translated, one cannot be expressed in a reasonable way, while for further two we did not find any way to express them in SHACL-Core at all. Let us discuss these, and the involved challenges in more detail:

- *difference-within-range* (Q21510854): not expressible
- *allowed qualifiers* (Q21510851): not reasonably expressible
- *allowed entity type* (Q52004125): only partially verifiable
- *single-value* (Q19474404): only partially expressible
- *distinct-values* (Q21502410): only partially expressible
- *single-best-value* (Q52060874): not expressible

[36] Available at https://www.rdfhdt.org/datasets/.
[37] https://linkedopendata.eu/wiki/The_EU_Knowledge_Graph
[38] https://github.com/nicolasferranti/wikidata-constraints-formalization/tree/main/shacl-generator

Firstly, the *difference-within-range* (Q21510854) constraint requires the difference between two values to be calculated and compared to a predefined range. Despite SHACL-Core having components to check for equalities (`sh:equals`) and inequalities (`sh:disjoint`, and `sh:lessThan`), arithmetic operations for computing differences are not included, which yields SHACL-Core unusable for expressing this constraint.

Next, the *allowed qualifiers* (Q21510851) constraint is also not directly/reasonably expressible in SHACL-Core. This constraint type specifies that only the listed qualifiers should be used when a certain statement is made, meaning that the use of all *other* qualifiers is disallowed. The problem here lies in the fact that SHACL-Core does not have direct means to *query* non-allowed paths (e.g. by referring to a path/property via a specific type).[39] We present an admittedly "clumsy" workaround, i.e., listing all non-allowed qualifiers explicitly in our GitHub repository.[40] Apart from the ridiculous length of the resulting shape description, this approach seems impractical as it does not depend on the data graph and parameters describing the constraint itself, but on a complete list of qualifiers in Wikidata (reduced by those provided as parameters).

The *allowed entity type* (Q52004125) originally seemed easily expressible in SHACL-Core to us. However, as mentioned in Section 2.4.4, it needs a number of workarounds to be executable on Wikdata's query service (and might not generalise to other Wikidata RDF serialisations). Figure 12 shows a respective translation of an allowed entity type constraint on the property *object has role* (P3831), restricting its subjects to the following entity types:

- Wikibase item
- Wikibase MediaInfo
- Wikibase lexeme
- Wikibase form
- Wikibase sense
- Wikibase property

```
:P3831_AllowedEntityTypesShape
    a sh:NodeShape ;
    sh:targetSubjectsOf wdt:P3831;
    sh:or ( [
            sh:path  wikibase:sitelinks;          # workaround to check wikbase:Item
            sh:minCount 1
        ] [
            sh:path  rdf:type;
            sh:hasValue wikibase:MediaInfo  ;
            sh:minCount 1
        ] [
            sh:path  rdf:type;
            sh:hasValue ontolex:LexicalEntry  ;   # workaround to check wikbase:Lexeme
            sh:minCount 1
        ] [
            sh:path  rdf:type;
            sh:hasValue ontolex:Form  ;           # workaround to check wikbase:Form
            sh:minCount 1
        ] [
            sh:path  rdf:type;
            sh:hasValue ontolex:LexicalSense  ;   # workaround to check wikbase:Sense
            sh:minCount 1
        ] [
            sh:path  rdf:type;
            sh:hasValue wikibase:Property  ;
            sh:minCount 1
        ]
    ) .
```

Fig. 12. SHACL-Core shape for verifying an *allowed entity type* (Q52004125) constraint on property *object has role* (P3831) , incl. workarounds defined in Wikidata's query service documentation to check entity types missing in the RDF serialization (cf. Section 2.4.4).

[39] We leave it as an open question at this point whether there exists a more concise formulation in terms of more complex, possibly nested SHACL Shapes.

[40] Don't try this at home! https://github.com/nicolasferranti/wikidata-constraints-formalization/blob/main/constraints-formalization/allowed%20qualifiers/shacl_shape_P6.ttl.

The last three constraint types in our problematic list (*single-value* (Q19474404)) *distinct-values* (Q21502410), and *single-best-value* (Q52060874)) are those using the *separator* (P4155) qualifier. We recall from Section 2.4.2 this qualifier is difficult to express in itself.

On the contrary, it is straightforward to verify the uniqueness or difference of a property value with respect to the claimed subject in the absence of separators: as such, both (*single-value* (Q19474404)) and *distinct-values* (Q21502410) constraints are easily expressible in SHACL-Core, as long as they do not specify separators. We illustrate this with a simple single-value "shape" in Fig. 13a.

The scenario changes though when a separator qualifier property is introduced. According to both possible interpretations **I1** and **I2** it is necessary to compare the values obtained through (shared) separators to assess uniqueness. However, it is not possible to compare the values of different paths that correspond to unique combinations of separators in SHACL-Core, i.e., it is not possible to distinguish different nodes matching the same regular path expression.

To illustrate this, consider again the instantiation of the separator qualifier for the *single-value* (Q19474404) constraint on the property *capital* (P36) from Fig. 9. In order to not have to distinguish between **I1** and **I2**, it is sufficient to discuss the case with at most one separator here (i.e., a simplification of the actual constraint which (possibly) also involves other separator qualifiers). Intuitively speaking, a data graph validates this constraint if, for each node a of the graph, the following conditions hold: (i) node a has at most one *capital*-outgoing edge to a node that does not have separators defined, and (ii) if node a has *capital*-outgoing edges to two distinct nodes, then they must have *start time* edges and they must not have the same filler node for the *start time* edges... Clearly, (i) is easily representable in SHACL-Core by a combination of *sh:maxCount 1* and negation *sh:not*:. E.g., in abstract syntax, the shape expression could be represented by the DL concept: $\leqslant_1$ *capital*.($\neg\exists$*start time*). However, to express condition (ii), we need mechanisms for identifying nodes along a path and for asserting identity properties on them, which are not supported in standard DLs. There has been significant research to extend DLs with *identification constraints* [17] or some sophisticated forms of *path-based identification constraints* [16]; the latter allows to consider complex paths with possibly inverse and non-functional properties, such as those in the Wikidata constraints. However, to the best of our knowledge, these DL extensions are not covered in SHACL-Core.

Therefore, the variants of all three, *single-value* (Q19474404)) *distinct-values* (Q21502410), and *single-best-value* (Q52060874) constraints which include separators cannot be represented in SHACL-Core. We additionally mention that, for analogous reasons, the *single-best-value* (Q52060874) constraint, used to specify that the property P shall only have a single value claim marked with *wikibase:BestRank* as *wikibase:rank*, is not expressible in SHACL-Core either, even without separators, due to a similar dependence on the rank.

We show in the next section how these remaining constraint types can be expressed with formalisms beyond SHACL-Core.

3.5. Beyond SHACL-Core: SHACL-SPARQL

Beyond its core language, SHACL provides a mechanism to refine constraints in terms of full SPARQL queries through a SPARQL-based constraint component (*sh:sparql*). In order to illustrate this feature, we refer to Figs 13a and 13b: both these SHACL shapes have the same semantics, while the shape in Fig. 13a uses only SHACL-Core language components, the one in Fig. 13b uses the *sh:sparql* extension to state that only one value is expected by means of a SPARQL query (starting at line 6): here, the reserved variable `?this` refers to the target node, whereas the variables `?path` and `?value` denote the path and value pointing to a violation. We further demonstrate the versatility of this extension by gradually refining the query of Fig. 13b.

Figure 13c extends the SHACL-SPARQL shape the existence and – in case – equality of the *start time* (P580) qualifier, thereby implementing an extension towards handling a single separator.

However, as there may be several qualifiers, this shape is not sufficient. Figure 13d generalizes the SHACL shape to consider as violations entities that have qualifiers any with equal values, as such implementing interpretation **I1** from Section 2.4.2. However, as we discussed there, this is again not the semantics used by Wikidata: finally, Fig. 13e implements the constraint according to interpretation **I2** from Section 2.4.2. While we do not go into details to explain the relatively complex SPARQL query in the `sh:sparql` at this point (hang on until Section 4.2), we

```
@prefix : <http://example.org/> .
@prefix ex: <http://semantics.id/ns/example#> .
@prefix wdt: <http://www.wikidata.org/prop/direct/>.
@prefix sh: <http://www.w3.org/ns/shacl#> .

# capital P36
ex:P36_SingleValueConstraintShape a sh:NodeShape ;
    sh:targetSubjectsOf wdt:P36;
    sh:property [
        sh:path wdt:P36;
        sh:maxCount 1 ] .
```

(a) *single-value* constraint in SHACL-Core, no separators.

```
ex:P36_SingleValueConstraintShape a sh:NodeShape ;
    sh:targetSubjectsOf wdt:P36;
    sh:sparql [
        a sh:SPARQLConstraint ;
        sh:select """
            SELECT ?this (wdt:P36 AS ?path) ?value
            WHERE {
                $this wdt:P36 ?value, ?value2 .
                FILTER (?value != ?value2 )
            } """ ] .
```

(b) Same as (a) written with SHACL-SPARQL, no separators.

```
ex:P36_SingleValueConstraintShape a sh:NodeShape ;
    sh:targetSubjectsOf wdt:P36;
    sh:sparql [
        a sh:SPARQLConstraint ;
        sh:select """
            SELECT ?this (p:P36 AS ?path) ?value
            WHERE {
                $this p:P36 ?n1,?n2.
                ?n1 ps:P36 ?value.
                FILTER (?n1 != ?n2 ).
                FILTER (
                    NOT EXISTS {?n1 pq:P580 [].
                    ?n2 pq:P580 []}
                    ||
                    EXISTS {?n1 pq:P580 ?sep.
                    ?n2 pq:P580 ?sep}
                )
            } """ ] .
```

(c) Extension of (b) capturing violations in two cases: either there is no qualifier or there is a *start time* (P580) with the same value.

```
ex:P36_SingleValueConstraintShape a sh:NodeShape ;
    sh:targetSubjectsOf wdt:P36;
    sh:sparql [
        a sh:SPARQLConstraint ;
        sh:select """
            SELECT ?this (p:P36 AS ?path) ?value
            WHERE {
              $this p:P36 ?n1,?n2.
              ?n1 ps:P36 ?value.
              FILTER (?n1 != ?n2 ).
              wd:P36 p:P2302 ?constraint_node.
              ?constraint_node ps:P2302 wd:Q19474404.
              FILTER (
                NOT EXISTS {
                ?constraint_node
                    pq:P4155/wikibase:qualifier ?pq_sep.
                ?n1 ?pq_sep [].
                ?n2 ?pq_sep [].           }
                ||
                EXISTS {
                ?constraint_node
                   pq:P4155/wikibase:qualifier ?pq_sep.
                ?n1 ?pq_sep ?sep.
                ?n2 ?pq_sep ?sep.          }           )
            } """ ] .
```

(d) Includes the equity test for all separators but still incomplete, as one equal separator is enough to consider a violation.

```
ex:P36_SingleValueConstraintShape a sh:NodeShape ;
    sh:targetSubjectsOf wdt:P36;
    sh:sparql [
        a sh:SPARQLConstraint ;
        sh:select """
            SELECT ?this (p:P36 AS ?path) ?value
            WHERE {
              $this p:P36 ?n1,?n2.
              ?n1 ps:P36 ?value
              FILTER (?n1 != ?n2).
              wd:P36 p:P2302 ?constraint_node.
              ?constraint_node ps:P2302 wd:Q19474404. # single-value constraint
              ?n1 ps:P36 ?o1_entity.
              ?n2 ps:P36 ?o2_entity.
              FILTER (?o1_entity != ?o2_entity)
              FILTER (
                NOT EXISTS {
                   ?constraint_node pq:P4155/wikibase:qualifier ?pq_sep.
                   ?n1 ?pq_sep [].
                   ?n2 ?pq_sep [].
                }
                ||
                NOT EXISTS {
                   ?constraint_node pq:P4155/wikibase:qualifier ?separator_property_qualifier.
                   ?n1 ?separator_property_qualifier ?sep1.
                   ?n2 ?separator_property_qualifier ?sep2.
                   FILTER (?sep1 != ?sep2) }                                    )
            } """ ] .
```

(e) Full SHACL-SPARQL shape considering all separators correctly (as composite keys).

Fig. 13. SHACL shapes encoding: from simple SHACL-Core shapes to the SPARQL formalization.

may ask the question first, whether SHACL with its limitations is at all the most adequate formalism for the kind of constraint checking we are looking for.

3.6. Towards SPARQL

In summary, we note the following limitations for the implementation of Wikidata property constraints via SHACL.

In summary, we observe that not all Wikidata constraints were possible to directly map as shapes in SHACL-Core.

- Firstly, we can cover only a subset of SHACL-expressible Wikidata property constraints in SHACL-Core;
- Secondly, our approach introduced so far instantiates separate SHACL shapes for each property constraint definition, even if we used SHACL-SPARQL;
- Thirdly, the capacity of checking these constraints (there were over 72K constraint definitions in total at the time of writing) against the whole Wikidata graph – to the best of our knowledge – goes beyond the scalability (and feature coverage) of existing SHACL validators.

As for the first item, clearly, the above-mentioned issues regarding the expressivity of Wikidata property constraints within SHACL-Core limit its applicability. Moreover, non-core features are unfortunately not mandatorily (and thus rarely) implemented by SHACL validators so far.

As for the second and third items, we argue that due to the limitations associated with the expressibility of the SHACL constraints and the lack of tools capable of efficiently validating large graphs, a direct SPARQL translation potentially presents itself as a more generic, flexible, and operationalizable approach for validating Wikidata Property constraints.

Let us demonstrate this idea with a straightforward SPARQL translation of a simple *conflicts-with* (Q21502838) constraint for the property *family name* (P734), which, according to the constraint *property* (P2306) qualifier should not be used together with the property *given name version for another gender* (P1560), as expressible relatively concisely in the following SHACL shape:

```
@prefix wdt:    <http://www.wikidata.org/prop/direct/> .
@prefix sh:     <http://www.w3.org/ns/shacl#> .

    [
        a sh:NodeShape ;
        sh:targetSubjectsOf wdt:P734;
        sh:property [
        sh:path wdt:P1560 ;
        sh:maxCount 0 ;
    ]
].
```

An even more direct and crisp, and also executable formulation of this constraint can be easily constructed by the following SPARQL query:

```
SELECT * WHERE {
    ?T wdt:P734 [] .
    ?T wdt:P1560 [] . }
```

In fact, we claim that violations of this particular constraint type, i.e. the *conflicts-with* constraint, could be checked more generally on *all* properties in one go, with a single SPARQL query:[41]

```
SELECT DISTINCT ?T ?wdtprop ?conflicting_wdtprop WHERE {
    ?wdprop wikibase:directClaim ?wdtprop;
    p:P2302 [ ps:P2302 wd:Q21502838 ;
              pq:P2306/wikibase:directClaim ?conflicting_wdtprop ].
    ?T ?wdtprop [].
    ?T ?conflicting_wdtprop [] . }
```

[41] https://w.wiki/6LU5

Indeed, this single query checks *all* violations of the conflicts-with constraint type and returns all violations of conflicts-with constraints at once. Intuitively, lines 2–3 lookup properties `?wdtprop` and the corresponding entity `?wdprop` (cf. the illustration in Fig. 7) having a conflict-with constraint (line 3) and retrieving the respective conflicting property `?conflicting_wdtprop` (line 4). Finally, lines 5–6 check the existence of a statement using both conflicting properties for the target `?T`. The query is executable directly on the Wikidata SPARQL endpoint, cf. Footnote 41, with the slight limitation that we need to LIMIT the results retrieved, as overall too many such violations exist to be retrieved via the UI at once; more details on that will be provided in our experiments section (Section 5). We stress that SHACL is not amenable to this approach: as a declarative constraint language, it does not have the querying capability to extract all constrained properties and their violations at once.

Overall, we hope the illustrative examples in this section have sufficiently motivated that a direct translation of property constraints to SPARQL has advantages over SHACL for various reasons. That is, while in this section we in principle have made a case for using (a subset of) Wikidata's property constraints as a "playground" to automatically generate a large testbed for SHACL(-Core) validators (and have also sketched how to extend this approach to SHACL-SPARQL), we also hope to have convinced the reader that this approach is not (yet) practically feasible, and in the end have made a case for direct generalizing our approach to *fully* operationalize Wikidata constraint validation via SPARQL.

4. Operationalizing Wikidata constraints with SPARQL

As opposed to the prototypical nature of the previous section, here we aim at a fully *operationalizable* formalization. We propose SPARQL as a constraint representation formalism that fulfills both the requirements to be (i) operationalizable – in the sense of being able to compute and report inconsistencies, in a similar fashion as the existing Wikidata database reports – as well as (ii) declarative – in the sense of an unambiguous, exchangeable formalization, capable of understanding the meaning of constraints.

The availability of Wikidata's database reports web page,[42] which presents statistics about the number of violations of a set of properties for all property constraints types, demonstrates that it is indeed in the interest of the Wikidata community that inconsistencies are identified and resolved. It also indicates that indeed there is an operationalized workflow to check these constraints already. The current reports provide access to a separate page for each property listed where one can take a detailed look at the inconsistent claims per violated property.[43] Unfortunately, though, the result of the operationalization shown on Wikidata's database reports pages are only available in HTML format, and moreover, the code behind is not publicly available. That is, the current operationalization is neither declarative nor is the code openly available.

As a summary of these database reports, Fig. 14 shows the development of property constraints over time for the 10 most violated constraints: according to the Wikidata database reports web page, we observed that since the introduction of Wikidata property constraints in 2015, the total number of constraints has grown from 19 in 2015 to 32 in 2023; new constraints were created, evolved, or ceased to exist. Data in Fig. 14 point to an increase in the number of violations for the *one-of* (Q21510859) constraint type. *Required qualifier* (Q21510856) constraints were introduced and began to be analyzed in 2019 only and are already emerging among the main causes of violations. The modeling of constraints is constantly evolving, for instance, *used for values only* (Q21528958), *used as reference* (Q21528959), and *used as qualifier* (Q21510863) constraints no longer exist and were migrated to a single constraint type: *property scope* (Q53869507). As such, we emphasize that our endeavor to model and formalize constraint types as SPARQL queries should not be viewed as a once-off exercise.

Rather we aim at proposing to rethink the *process* of developing such constraints themselves in terms of such SPARQL queries to be included as an (i) operationalizable and (ii) declarative means for their definitions:

[42] https://www.wikidata.org/wiki/Wikidata:Database_reports/Constraint_violations/Summary

[43] For instance, our example item-requires statement constraint on FIFA player ID is reported at https://www.wikidata.org/wiki/Wikidata:Database_reports/Constraint_violations/P1469, reporting **192 violations**, retrieved 13 Jan 2023.

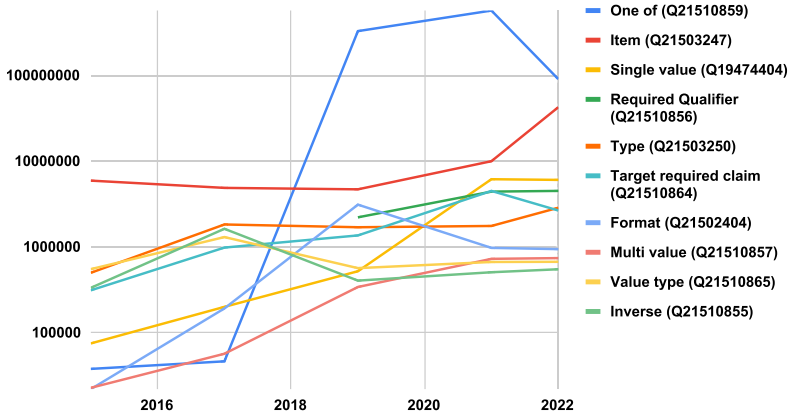
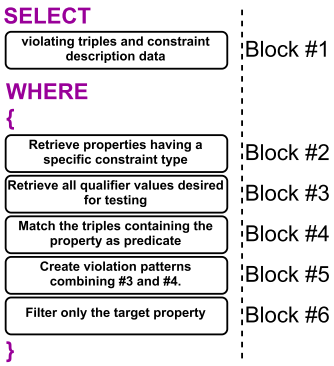
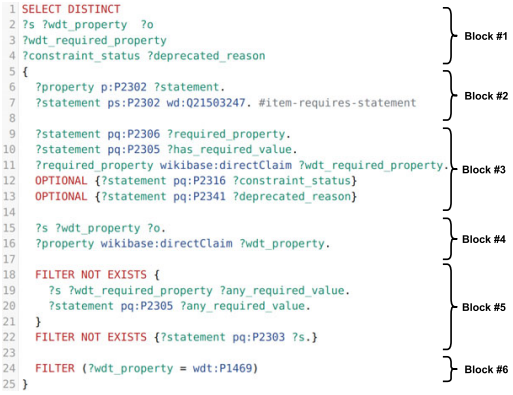


Fig. 14. #violations for top 10 most violated constraint types (logarithmic scale).



(a) Generic structure adopted by all the provided SPARQL queries



(b) SPARQL query that retrieves inconsistent data for *item-requires-statement* constraints (Q21503247) for FIFA Player ID (P1469) with a required value

Fig. 15. SPARQL queries general template with exemplification.

- as for (i), Wikidata as an RDF graph can be queried through a SPARQL query service – by expressing constraint violations per constraint type as SPARQL queries, we can benefit from the query language's operationalizable nature, and various existing SPARQL implementations, that scale to billions of triples.
- as for (ii), SPARQL itself is a declarative language, with well understood theoretical properties and mappable to other locigal languages, such as Datalog [8,49,51]

4.1. Expressing and validating Wikidata constraints in SPARQL

In this section, we describe the overall structure of our Wikidata constraint validation approach using SPARQL queries. We again illustrate it via our running example from Fig. 8.

Figure 15a presents a generic structure followed by each SPARQL query proposed in this paper. We generalize queries into different "blocks", such that each block can contain multiple triple patterns as exemplified in Fig. 15b, which fulfill different functions. Figure 15b represents the concrete query for the *item-requires-statement* constraint:

for this constraint type a required property and its required value(s) need to be checked. Each block of the query structure of Fig. 15b is detailed as follows.

Block #1, i.e., the SELECT clause, represents the information to be returned by the query describing the violation; usually it is composed of the claim containing the property that is violating the constraint (Fig. 15b line 2), plus extra information about the triple or about the constraint – in our case, the property missing for the subject – (line 3), and finally, we also return the constraint status or reason for deprecation (line 4), if given.

Block #2 i.e., the beginning of *WHERE* clause, matches the properties (and associated statements) that use the particular constraint type (lines 6 and 7) – in our case for the *item-requires-statement* (Q21503247). To retrieve properties using another specific property constraint type, one obviously only needs to adapt the constraint id in line 7.

Block #3: statements from Block #2 are then used to retrieve the required constraint qualifiers for the specific constraint type (lines 9 and 10), i.e., in our case the required property (line 9) and value (line 10), as explained in Section 3 above, plus optional qualifiers such as the constraint status or information about constraint deprecation (lines 12 and 13). We note that these additional OPTIONAL parts may affect query performance, and could be potentially left out, but we suggest retrieving them if present for additional detailed information about actual violations.

Block #4 matches the actual statements that must be checked for constraint verification (line 15). As we can see in this example, we need to navigate between the different property namespaces described in Fig. 7 above before matching the subject and object (line 16).

Block #5 combines the statement qualifiers values from Block #3 and the triples from Block #4 to check the actual violation. For the *item-requires-statement* constraint, the goal is to check the non-existence of the required property along with the required value (lines 19 and 20) while removing the explicit exceptions (P2303) to the constraint (line 22).

Block #6 is optional with the intention to parameterize the query. The FILTER restricts the query to retrieve violations for one specific property, in our case, "FIFA player ID" (line 24). Although our queries are designed to retrieve violations for all properties of a single constraint type, we recommend execution for a specific property at a time due to Wikidata's size and the limitation of resources available on the online endpoint.

We have encoded all 32 constraint types (some of which are in separate queries for different variations) in SPARQL queries, following similar patterns corresponding to **Block#1**–**Block#6** in our example. These queries are designed to retrieve information about any violations (somewhat orthogonal to the SHACL encodings that model *conformance* instead). The full list of these SPARQL queries can be found in Table 2 – as shortcut links to our GitHub repository (cf. Footnote 35), and executable on Wikidata's query service. The list contains examples for checking particular properties per constraint type. For instance, our query https://short.wu.ac.at/72ng implementing the FIFA Player ID's item-requires-statement constraint returned **190 violations**, as opposed to the 192 on Wikidata's database report page (cf. Footnote 43) at the time of writing. In our formalization, we divide the item-requires statement constraint checking into two queries, one including required properties and another including required properties and values. The second query https://short.wu.ac.at/rmba retrieves the remaining **2** violations. Although both checks could be combined in one UNION query, we prefer this design with multiple queries in case of different possible constraint variations, cf. the respective lines of Table 2 pointing to multiple queries.

Again, we note that apart from Wikidata itself, there are an increasing number of other Wikibase instances listed in the Wikibase Registry[44] that partially re-use Wikidata's property constraints. Obviously, for Wikibase instances that declare constraints using the Wikidata property constraints model, it should likewise be possible to check constraint violations on Wikibase instances using our SPARQL templates. For instance, The EU Knowledge Graph[45] is a Wikibase knowledge graph containing information about institutions of the European Union, countries, projects financed by the EU and their beneficiaries [23], which imports Wikidata's property constraints. As an example, we used our SPARQL template to search for *format constraint* violations on this KG, with our query https://short.wu.ac.at/xmsm returning 472 violations at the time of writing.

[44] https://wikibase-registry.wmflabs.org/wiki/Main_Page
[45] https://linkedopendata.eu/wiki/The_EU_Knowledge_Graph

4.2. Formalizing constraint types not captured with SHACL-Core in SPARQL

We now show how to formally express the constraint types that could not be directly represented with SHACL-Core in SPARQL.

Single-value Used to specify that one property generally contains a single value per concept, we already discussed the partial representability within SHACL-Core, when there are no separators defined. The following SPARQL query for the property *capital* (P36) shows how to capture violations in two scenarios: either there are multiple different values with no separators or there are separators with equal values. Block 4 binds multiple statements that are further tested in block 5. In block 5, it is tested if the values are different (line 14), if there are no separators (lines 16–19), or if it does not exist a separator (line 23) where the separator values (lines 24 and 25) for two different statements are different (line 26). Overall, this captures the intended semantics **I2**, described in Section 2.4.2. The same principle also applies for *distinct-values* and *single-best-value*, the two other constraint types that make use of separator qualifiers and can be expressed in SPARQL similarly, for details we refer to the links in Table 2.

```
1  SELECT DISTINCT # Block 1
2  ?subject
3  ?wds1 ?wds2
4  ?o1_entity ?o2_entity
5  WHERE {
6     wd:P36 p:P2302 ?statement. # Block 2
7     ?statement ps:P2302 wd:Q19474404. # single-value
8
9     ?subject p:P36 ?wds1,?wds2. # Block 4
10    ?wds1 ps:P36 ?o1_entity.
11    ?wds2 ps:P36 ?o2_entity.
12
13    # Block 5
14    FILTER (?o1_entity != ?o2_entity)
15    FILTER (
16       NOT EXISTS {
17         ?statement pq:P4155/wikibase:qualifier ?pq_sep.
18         ?wds1 ?pq_sep [].
19         ?wds2 ?pq_sep [].
20       }
21       ||
22       NOT EXISTS {
23         ?statement pq:P4155/wikibase:qualifier ?separator_property_qualifier.
24         ?wds1 ?separator_property_qualifier ?o1_qualifier.
25         ?wds2 ?separator_property_qualifier ?o2_qualifier.
26         FILTER (?o1_qualifier != ?o2_qualifier)
27       }
28    )
29 }
```

Allowed qualifiers This constraint type specifies that only the listed qualifiers should be used when a certain statement is made, meaning that the use of all *other* qualifiers needs to be restricted. Since there is no way to list non-allowed paths implicitly (e.g. by referring to a path/property via a specific type), this constraint could not be expressed with SHACL-Core. However, when using SPARQL, it is possible to test all the predicates where the statement node is a subject. The following query presents the SPARQL formalization for property *party chief representative* (P210), where Block 4 binds all the statements about P210 (line 6) and their respective qualifiers (lines 7 and 8). Next, Block 5 creates the violation pattern, where the statement of Block 4 is considered a violation if at least one found qualifier is not part of the set of expected ones.

```
SELECT DISTINCT ?object_statement # Block 1
WHERE {
    wd:P210 p:P2302 ?constraint_statement. # Block 2
    ?constraint_statement ps:P2302 wd:Q21510851. ## allowed qualifiers

    ?subject p:P210 ?object_statement. # Block 4
    ?object_statement ?pq_qualifier ?value.
    ?wd_qualifier wikibase:qualifier ?pq_qualifier.

    # Block 5
    FILTER NOT EXISTS {?constraint_statement pq:P2306 ?wd_qualifier}
}
```

Difference-within-range The constraint requires the difference between two values to be calculated and compared to a predefined range. SHACL-Core provides functionalities for checking equalities and inequalities, but it does not encompass arithmetic operations as part of its components. Below we present a simplified version of the query to check *difference-within-range* violations for *date of burial or cremation* (P4602). Block 3 binds all the necessary qualifiers, such as the minimum and maximum value (lines 10 and 11) and the property necessary to create a valid range (line 12), which in this case is *date of death* (P570). Block 4 binds the values for P4602 and P570. Finally, the interval is compared with the expected period (block 5).

```
SELECT DISTINCT # Block 1
    ?subject ?wdt_property_to_check_differ
    ?min_val ?max_val
    ?date_1 ?date_2
    (COALESCE( (ABS(xsd:date(?date_1) - xsd:date(?date_2)))/365  , "null" ) as ?diff)
WHERE {
    ?wd_property p:P2302 ?statement . # Block 2
    ?statement ps:P2302 wd:Q21510854 . ## difference-within-range

    ?statement pq:P2313 ?min_val . # Block 3
    ?statement pq:P2312 ?max_val .
    ?statement pq:P2306 ?property_to_check_differ .
    ?property_to_check_differ wikibase:directClaim ?wdt_property_to_check_differ.
    ?property_to_check_differ wikibase:claim ?p_property_to_check_differ.

    ?subject ?wdt_property ?date_1; # Block 4
             ?wdt_property_to_check_differ ?date_2.
    ?wd_property wikibase:directClaim ?wdt_property .

    FILTER ( # Block 5
        COALESCE(   (ABS(xsd:date(?date_1) - xsd:date(?date_2)))/365  , "null" ) < ?min_val ||
        COALESCE(   (ABS(xsd:date(?date_1) - xsd:date(?date_2)))/365  , "null" ) > ?max_val
    )

    FILTER (?wdt_property = wdt:P4602) #Block 6
}
```

5. Experiments

We designed an experiment to evaluate the semantics of our SPARQL queries to verify our approach against the Wikidata database reports. We compared the violations obtained by our queries with the violations published in the Wikidata Database reports (cf. Footnote 5). Unlike DBpedia, where a version of the KG is pragmatically generated and made available every three months,[46] Wikidata's dynamic approach causes the KG to be constantly updated with new statements. This approach makes difficult the comparison between strategies to capture violations because of the uncertainty of the KG's state at the time violations are collected. Thus, checking and comparing *all* constraint violations turns infeasible, since collecting them via the SPARQL endpoint takes too long to keep in sync, even disregarding (unfortunately increasingly common) timeouts on the SPARQL query endpoint.

In order to still ensure comparability of results as far as possible, the conducted experiment was designed on a sample of constraint violations collected according to the following steps:

[46] https://www.dbpedia.org/resources/snapshot-release/

1. We identified the top-5 most violated constraint types from Wikidata's violation statistics table on December 16, 2022: *One-of* (Q21510859), *item-requires-statement* (Q21503247), *Single value constraints* (Q19474404 and Q52060874), *required qualifier* (Q21510856), and *value-requires-statement* (Q21510864).
2. We ranked the associated properties in descending order of the number of violations for each of these constraint types.
3. We executed our SPARQL queries to collect the violations of five different properties for the five constraint types, totaling 25 violation sets available in our GitHub repository.[47]
4. The ad-hoc violation checking system used in Wikidata takes about a day to execute and publish results, thus our queries were executed one day before the data was available. Consequently, we extract the set of corresponding violations published by the Wikidata portal referring to the same properties on the next day. For instance, the *FIFA player ID* (P1469) property-specific violations are made available in the Wikidata report page.[48]
5. Finally, we structured and compared the violations reported by the Wikidata Database reports with the violations retrieved by the SPARQL queries on the Wikidata endpoint.

As the queries were executed on the SPARQL endpoint and our target was the properties with the highest numbers of violations, we also had to consider timeout-related issues due to limitations of the Wikidata environment itself: due to the high number of triples associated with some of the targeted properties, the limit of 60 seconds for a query, established by Wikidata's SPARQL endpoint is not enough to process the entire target set. Therefore, it was necessary to discard the target properties that timed out and proceed with the subsequent one with the next highest number of violations (in steps 2+3 above), to arrive at 5 properties for each of the 5 chosen constraint types. Note that in order to have a reasonable basis for comparison, the SPARQL endpoint is the only option at the moment, since the database reports are computed on this state of the KG. In future work, we intend to create a benchmark to facilitate the testing of different approaches to collecting violations including testing of other engines and environments; more on that in the related and future work sections below.

6. Results

In the next subsections, we provide a table for every constraint type containing the list of properties analyzed (Property ID), the total number of violations the Wikidata database reports claimed to have found (# of violations), the total number of violations made available by the database reports on the specific HTML pages for each property, the number of violations available (**VA**), the number of violations found by our SPARQL queries (**OV**), as well as the execution time for running the queries (Runtime), and the number of intersections **VA** $\cap$ **OV**. Unfortunately, the Wikidata database reports portal provides a maximum of 5001 violations for each pair (Property, Constraint type). Therefore the comparison of results was performed in terms of the partial results of the database reports and the full results of the approach using SPARQL. For results that were found by the database report, but not in our approach, i.e. for **VA** $\setminus$ **OV**, we did a manual inspection of the deviations.

6.1. One-of constraint

The first results concern the **One-of constraint**, and they are available in Table 3. Three properties were skipped for this constraint type due to timeout issues on the Wikidata SPARQL endpoint.

For ***genre* (P136)**, 11 violations were not found by our approach because the SPARQL queries (cf. explanation of **Block#4** above) are checking constraints on the exported *truthy* statements in the Wikidata RDF dump,[49] i.e., the *non-deprecated* statements in the wdt: namespace, which include the *PreferredRank* values (when there is one) when there are multiple values and the *NormalRank* values when there is no PreferredRank. Due to this, 8 out of

[47] https://github.com/nicolasferranti/wikidata-constraints-formalization/tree/main/experiment_data
[48] https://www.wikidata.org/wiki/Wikidata:Database_reports/Constraint_violations/P1469
[49] https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format#Truthy_statements

Table 3
One of constraint violations

Property ID	Wikidata database reports		Our SPARQL approach		VA ∩ OV
	# of violations	# of violations available (VA)	# of violations (OV)	Runtime (s)	
P136	810411	5001	805854	19,07	4990
P518	516308	5001	194	0,31	147
P437	389460	5001	193865	4,35	4997
P641	336724	5001	328934	35,72	4997
P512	337460	5001	84522	3,60	4991

11 subjects were found by our approach but associated with different values. For instance, we found *Baroque music* (Q8361) as the PreferredRank and a violation for the entity *Johann Sebastian Bach* (Q1339), while for the Wikidata Reports *western classical music* (Q9730), a NormalRank object is a violation. In ***applies to part* (P518)**, the 147 violations identified by both approaches contain the property P518 used as the main property (wdt: prefix), and these violations are also highlighted by the Wikidata page of each entity in the UI,[50] e.g. *elder abuse* (Q427883) has three different values for P518; all of their violations to the constraint. Notably, the remaining reported violations not identified by the SPARQL query are also not considered violations in the Wikidata pages in the UI, because the property P518 is used as a *qualifier* in these cases. For instance, *Catalan Countries* (Q234963) has *Italy* (Q38) as *country* (P17), but this *applies to part* (P518) *Alghero* (Q166282). Currently, this is not displayed as a violation by the Wikidata pages, but the Database reports are testing the constraint also for qualifiers. In our case, it would be necessary to adapt the triple pattern "`?s wdt:P518?o`" to alternatively also test the qualifier namespace (pq:) if one wants to test the query also for qualifiers.

The four violations not captured by our approach for the property ***distribution format* (P437)** are due to the same reason highlighted for P136. We identified these four subjects but since the property has multiple values and there is one PreferredRank value, the SPARQL query computes the violation for the value in the PreferredRank. For instance, *The Simpsons* (Q886) has *video on demand* (Q723685) and *terrestrial television* (Q175122) as *distribution format* (P437) but *terrestrial television* (Q175122) is marked as the PreferredRank. While the SPARQL query captures a violation for *terrestrial television* (Q175122), Wikidata Reports captures a violation for *video on demand* (Q723685). Again, for ***sport* (P641)**, the four violations are regarding multiple values with different Ranks. For instance, *Tove Alexandersson* (Q113200) has *sport* (P641) *orienteering* (Q29358), *ski orienteering* (Q428242), *skyrunning* (Q3962667), and *ski mountaineering* (Q1075998). In addition, *orienteering* (Q29358) is the PreferredRank. While the SPARQL query captures *orienteering* (Q29358) as the violation, the Wikidata reports capture the other values. The analysis of the one-of constraint for the property P641 reveals that an improvement option would be: instead of listing every possible sports type, the constraint could refer to a superclass of sports and include hierarchical class inference when testing the constraint. Therefore, all the orienteering types would be under the same subclass of sport and it would not be necessary to add a new type of sport to the constraint once this type is connected to a superclass.

Lastly, in ***academic degree* (P512)**, the 10 missing violations stem again from the lack of existence of the triple pattern "`?s wdt:P512?o`" between the tested subject and object (checking only preferred or truthy statements, as discussed above). A simple adaptation of the query to also check non-preferred statements could be achieved by replacing "`?s wdt:PID?o`" with "`?s p:PID/ps:PID?o`". This allows for consideration of all possible statement nodes as demonstrated in query https://w.wiki/6HKA. This query shows that the proposed adaptation is able to find all the four violating values described for *Tove Alexandersson* (Q113200) and *sport* (P641).

6.2. Item-requires-statement constraint

For *Item requires statement* constraints (IRS), which are very common, ten properties were skipped due to the timeout in the Wikidata SPARQL endpoint. The values displayed in the VA column of Table 4 can exceed the value

[50] Note: it is necessary to be logged in in Wikidata to see violations in the UI).

Table 4
Item requires statement constraint violations

Property ID	Wikidata database reports		Our SPARQL approach		VA ∩ OV
	# of violations	# of violations available (VA)	# of violations (OV)	Runtime (s)	
P1559	492396	5001	497889	17,45	5001
P1976	215643	10004	215645	21,07	10004
P2539	197512	40066	197517	16,48	40066
P1053	197168	5265	200091	35,34	5260
P814	139178	19476	135647	12,35	19386

Table 5
Single value/best single value constraints violations

Property ID	Wikidata database reports		Our SPARQL approach		VA ∩ OV
	# of violations	# of violations available (VA)	# of violations (OV)	Runtime (s)	
P881	55694	5001	55693	14,63	5000
P7015	53020	5001	53019	27,35	5000
P1540	44366	5342(5073 unique)	271(245 unique)	46,86	232
P1539	44190	5178(5076 unique)	284(258 unique)	44,76	244
P2227	39581	5001	39577	7,23	4999

5001 for this constraint type because the same property can have multiple IRS instances. Moreover, such instances can request the existence of only one property or a pair ⟨property, value⟩. Therefore, the Wikidata database reports present a list of violations for each instance of IRS, which we summed up in tables to report the respective **VA** numbers.

In Table 4, note that for the top 3 properties (P1559, P1976, and P2539), our approach found all the available violations and some extra violations that unfortunately cannot be compared because the results available in the Wikidata database reports are incomplete. For ***ResearcherID* (P1053)**, five statements were not captured by our queries due to deprecated values, i.e., again, Wikidata does not match the pattern "?s wdt:P1053?o" when?o is deprecated (i.e., non-truthy). Further, on property P1053, Wikidata database reports point to four violations on the IRS with the required property *instance of* (P31) and *value human* (Q5). Our approach identified these four violations and two more not reported by the database reports: *Milieu Intérieur Consortium* (Q86498220) and *Wolfgang Wagner* (Q73833983). The first one is an instance of *project* (Q170584), and the second one has *conflation* (Q14946528) as the PreferredRank value. In this case, we can also notice a bad usage of the constraint, because the IRS constraint was used to restrict the type of the subject, simulating the behavior of what should actually be a *subject type* constraint.

The 90 violations not found by our SPARQL query for ***IUCN protected areas category* (P814)** are because the objects are empty values. Therefore, as the Wikidata RDF dump does not contain "?s wdt:P814 ⟨empty⟩" for empty objects we can not retrieve them. For instance, *Sandgrube Seligenthal* (Q2220711) has an empty value for *IUCN protected areas category* (P814). We demonstrate, using query https://short.wu.ac.at/5be5, that again our approach could be easily adapted to include empty objects by replacing the "?s wdt:PID []" pattern by "?s p:PID []". Yet again, whether empty values should be reported as violations here or not is in our opinion a matter of interpretation.

6.3. Single-value constraint

The statistic table of the Wikidata database reports points to Single-value constraint as the third most violated constraint type. We notice that this statistic takes into account *Single-value* (Q19474404) and *Single-Best-value* (Q52060874) constraints. Therefore, it was necessary to use the queries designed for these two types of constraints to perform the experiment. Eleven properties were skipped for presenting timeout in the SPARQL endpoint for at least one of the query types. The results are presented in Table 5, where, unlike the previous tables, we include in columns VA and OV the total number of unique entities found, i.e., the total number without repeated entities.

In ***type of variable star* (P881)**, *V Sagittae* (Q56303735) is pointed as a violation by the database reports but not by the Wikidata entity page and not by our query. *V Sagittae* (Q56303735) has indeed two values for *type of variable star* (P881): *nova-like star* (Q9283100) and *eclipsing binary star* (Q1457376). However, *nova-like star* (Q9283100) is marked as the PreferredRank value, therefore we do not take this as a violation. According to the definition of single-best-value, the property generally contains a single "best" value per item, though other values may be included as long as the "best" value is marked with PreferredRank. For ***surface gravity* (P7015)**, *SDSS J1539+0239* (Q4048714) is pointed as a violation by the database reports but, interestingly neither by the Wikidata entity page UI nor by our query. *SDSS J1539+0239* (Q4048714) has indeed two values for *surface gravity* (P7015): "1,450 centimeter per square second" and "3±0.15". However, "3±0.15" is marked as the PreferredRank value, therefore we do not consider this a violation because it is in accordance with the definition.

The analysis of ***male population* (P1540)** reveals that there are 5073 unique entities reported by Wikidata database reports as violations and not reported by our approach. 4841 of them have a PreferredRank defined, therefore we do not consider them violations. The other 232 we captured with our query. The same is also the case for ***female population* (P1539)**, where 5076 unique entities were reported by the Wikidata database as violations and not reported by our approach. 4832 of them have a PreferredRank and the other 244 we identified. Finally, for the property ***metallicity* (P2227)**, the two entities that database reports consider violations are *HD 1461* (Q523743) and *SDSS J1539+0239* (Q4048714). Again, our approach does not consider them as violations because, although they have multiple values, in both cases there is a value marked with PreferredRank. In fact, the respective pages in the Wikidata UI also do not highlight violations for these statements.

The occurrence of properties from the astronomy domain, such as *type of variable star* (P881) and *surface gravity* (P7015), was expected, since the astronomy community in Wikidata uses deprecation and different rankings to represent historical data, as also observed in [55]. Therefore, it is common to find statements with multiple values, where the higher-ranked ones represent more accurate or currently accepted data by the community.

6.4. Required qualifier constraint

The required qualifier constraint has the same principle described for IRS: the same property can have multiple instances of the required qualifier constraint, each one of them requiring a different property to be used as a qualifier for a given statement. For this constraint type, which again is very common, three properties were skipped due to timeout on the Wikidata SPARQL endpoint, where the properties with the next highest violation rates were selected. The results are available in Table 6, showing that the whole set of available violations (VA) was found by our SPARQL approach (OV).

6.5. Value-requires-statement constraint

Finally, *Value-requires statement* constraints (VRS) are similar to IRS, but instead of requiring the existence of a statement in the subject, these quite common constraints require a statement in the object. Ten overly common properties were skipped due to timeouts in the SPARQL endpoint. Two different queries were used for each property, one checking required properties and another checking pairs of required properties and values. The main results are available in Table 7.

Table 6
Required qualifiers constraint violations

Property ID	Wikidata database reports		Our SPARQL approach		VA ∩ OV
	# of violations	# of violations available (VA)	# of violations (OV)	Runtime (s)	
P996	496933	5001	496930	10,13	5001
P1539	106515	8769(2888 unique)	104632	27,33	2888
P1540	105294	7213(3189 unique)	105295	18,4	3189
P6	53884	5001	53895	3,07	5001
P1618	46548	10002(9944 unique)	46548	3,68	9944

Table 7
Value requires statement constraint violations

Property ID	Wikidata database reports		Our SPARQL approach		VA ∩ OV
	# of violations	# of violations available (VA)	# of violations (OV)	Runtime (s)	
P1435	1828277	5001(4823 unique)	1815937	28,25	4810
P680	35141	5001(4958 unique)	41224	24,43	4958
P10254	49169	5018	49163	17,81	5018
P1598	29219	14792(13627 unique)	29469	49,69	13609
P747	13920	5001(4992 unique)	13952	3,61	4984

For ***molecular function*** **(P680)** and ***associated cadastral district*** **(P10254)**, the intersection is equal to the number of violations published by the Wikidata database reports. The eight statements not found for ***has edition or translation*** **(P747)** and the 18 for ***consecrator*** **(P1598)** are due to the existence of one PreferredRank value among multiple values. Lastly, for the most violated property, ***heritage designation*** **(P1435)**, the 13 statements claimed as violations by database reports and not reported by our approach fall into the same category. There are multiple values and one of them is marked as PreferredRank value, therefore the pattern "`?s wdt:P1435?o`" does not capture the remaining values. For instance, *Vatican City* (Q237) has as *heritage designation* (P1435) the values *UNESCO World Heritage Site* (Q9259) and *Cultural property under special protection* (Q26086651), however *UNESCO World Heritage Site* (Q9259) is marked as the PreferredRank. Our query can be easily adapted to focus on the statement nodes instead of the direct value, as illustrated in query https://short.wu.ac.at/xasn. It is only necessary to replace "`?s wdt:P1435?o`" by "`?s p:P1435/ps:P1435?o`".

In summary, common reasons for mismatches include – as a matter of interpretation – whether only truthy statements or also non-preferred and deprecated statements should be checked for constraint violations. Also, other deviations could arguably be identified as a matter of interpretation. As we also discussed, our constraints could be adapted to the respective different interpretations relatively easily with minor modifications of our query patterns. Overall, while we only conducted these analyses on a sample, we argue that the experiment has confirmed our opinion that a declarative and adaptable formulation of Wikidata property constraints in terms of SPARQL queries is both feasible and could add to the clarification of the constraints' actual semantics. The deviations between the Wikidata UI pages and the Wikidata database reports confirm our opinion that such clarification is dearly needed.

Regarding the practical feasibility of the proposed approach, it is important to acknowledge the generic nature of the SPARQL queries proposed in this study. The absence of hard-coded parameters ensures adaptability to diverse constraint parameters, with queries exclusively relying on the Wikidata data model to match the triple patterns that generate specific constraint violations. This flexibility enables the applicability of generic queries checking violations across all properties instantiating a specific constraint type. On the downside, as we have discussed, such generic queries potentially lead to scalability problems for current SPARQL engines. As a practical workaround, constraints can also be checked by instantiating our generic queries per property, mitigating scalability challenges to a certain extent: in the context of the Wikidata ecosystem, the envisaged solution is therefore promising for deployment as a background process capable of processing batches of properties. By doing so, the system can systematically uncover candidate inconsistencies, according to the available resources and relevance of properties.

7. Related work

Constraints play an important role in specifying rules for data, defining the requirements to prevent it from becoming corrupt, and ensuring its integrity. There has been significant research on the development of constraint representations and validation techniques specifically for knowledge graphs.

7.1. Constraint languages for graph data

RDF has long served as the W3C-recommended graph-based data model for presenting information in the Semantic Web, whereas a standardized language to express and validate data graphs has only recently been introduced.

Ontology languages like RDFS, OWL, and its sublanguages, which have been standardized along with RDF, have been widely used for modeling the data through axiomatic structures. For instance, DBpedia, like other open knowledge graphs (e.g. YAGO, GeoNames), makes use of ontologies to model the data, which have been employed also for detecting (some) inconsistencies (e.g., [11,48]). However, ontologies have been particularly criticized for their limited use when checking the conformance of data graphs. Indeed, the primary utility of ontologies lies in facilitating deductive reasoning tasks, such as node classification or evaluating overall satisfiability, and not in describing constraints on KGs. With the growing emphasis on data accuracy for graph-based applications, the absence of constraint languages similar to those found in relational [3] and semi-structured data [2] contexts became noticeable. To address this gap, multiple strategies have emerged. Hogan [38] used rule-based fragments of OWL/RDFS for scalable inconsistency identification and repair suggestions. The idea to use scalable variants of bespoke Datalog-based reasoning for constraint checking and verification originally imposed in Hogan's thesis may be argued to be not unlike our approach: SPARQL has been shown to be equally expressive as non-recursive Datalog with negation [8], where features like property paths only mildly add harmless, linear recursion [51]. Another line of research regards extending ontology languages to treat axioms as integrity constraints under the closed-world assumption [45,58].

In particular, to address the lack of dedicated constraint languages for graph data, novel schema formalisms for RDF graph validation like the Shape Expressions language (ShEx) [13,29,56,57] were proposed before SHACL became a W3C recommendation. ShEx is a formal modeling and validation language for RDF data, which allows for the declaration of expected properties, cardinalities, and the type and structure of their objects. ShEx is closely related to SHACL, and in some cases, it is possible to translate SHACL shapes into ShEx shape expressions since their expressiveness is similar for common cases [29]. For instance, the shapes graph presented in Fig. 10 can be respectively represented in ShEx as follows:

```
:P1469_ItemRequiresStatementShape {
    wdt:P106 [ wd:Q937857 wd:Q1851558 wd:Q21057452 wd:Q628099]
}

{ FOCUS wdt:P1469 _}@:P1469_ItemRequiresStatementShape

:P1469_ItemRequiresStatementShape [ wd:Q370014 ] OR {
    wdt:P106 [ wd:Q937857 wd:Q1851558 wd:Q21057452 wd:Q628099]
}
```

Yet, we leave a full discussion about whether our approach, and therefore all existing constraint types transfer over to ShEx as an open question to future work. Additionally, validation languages based on ShEx supporting the Wikibase data model have been recently proposed in the literature [28], however, they still lack support for many Wikibase constructs and there is no operational validator yet.

Furthermore, SPARQL-based approaches to validate knowledge graphs can also be found in the literature [42,59], including the SPARQL Inferencing Notation (SPIN) [51] framework. SPARQL can also be used to validate numerical and statistical computations [29]. While in [42] the authors use SPARQL queries to validate and compute index data for linked data applications, in [59] SPARQL is used to specifically validate epidemiological data on Wikidata. Corman et al. proposed in cf. [20,22] a SPARQL-based method for validating possibly recursive SHACL shapes by translating them to SPARQL queries; non-recursive shapes are translated into single SPARQL queries. These queries are then directly evaluated on a SPARQL endpoint. They present a prototype implementation of their approach, called SHACL2SPARQL. Since in this work we translate the Wikidata constraints to SHACL shapes, a natural approach would have been to employ SHACL2SPARQL to generate the corresponding SPARQL queries. However, the primary focus of their algorithm is to handle fragments of recursive shapes, which makes validation significantly more involved. As a result, for the – in principle simple – non-recursive shapes we obtain in the present work, which however use a variety of SHACL-Core constructs not covered in SHACL2SPARQL, the prototype yields, in general, infeasible translations. More specifically, their approach generates SPARQL queries reporting violations by `NOT EXISTS` sub-queries expressing the conditions to be verified by the respective targets specified by the shapes graph. The construction of these subqueries is modularly defined via the SHACL grammar. For instance, consider a simple

[51] https://spinrdf.org/

conflicts-with (Q21502838) constraint on the property *family name* (P734). According to the constraint property, it should not be used together with the property P1560 (given name version for another gender), as expressible concisely in the following SHACL shape:

```
1  @prefix wdt:    <http://www.wikidata.org/prop/direct/> .
2  @prefix sh:     <http://www.w3.org/ns/shacl#> .
3
4      :P734_conflictWithConstraint
5          a sh:NodeShape ;
6          sh:targetSubjectsOf wdt:P734;
7          sh:property [
8              sh:path wdt:P1560 ;
9              sh:maxCount 0 ] .
```

According to Corman et al.'s translation, `sh:maxCount 0` is again translatable to a `NOT EXISTS` query, yielding overall a query like the following with nested `NOT EXISTS` operators:

```
1  SELECT * WHERE {
2      ?T wdt:P734 [] .
3      FILTER NOT EXISTS { ?T wdt:P734 [] .
4                          FILTER NOT EXISTS {  ?T wdt:P1560 [] . } } }
```

Unfortunately, this query currently times out on Wikidata's SPARQL endpoint, mainly due to the nested negation yielding from a modular translation. The `NOT EXISTS` operator is particularly hard to evaluate for SPARQL engines. Also note that our SPARQL formulation, to be executable at all, needs to "copy" the target definition within the verification part (line 3), due to the broken recursive correlation semantics of the `(NOT) EXISTS` operator in SPARQL, cf. [35,47]. A more direct and crisp, and also executable formulation of this query can be easily constructed:

```
1  SELECT * WHERE {
2      ?T wdt:P734 [] .
3      ?T wdt:P1560 [] . }
```

In addition, not all Wikidata constraints could be directly represented in SHACL-Core. We therefore had to devise specific SPARQL queries for each of the 32 Wikidata constraint types to generate viable and functional solutions.

7.2. Constraints in Wikidata

Data restrictions within Wikidata are also discussed by the community and implemented through further projects using other pre-established technologies. For instance, the *Wikidata Schemas* project[52] relies on ShEx. As opposed to property constraints, the *Schemas Project* is focused on defining entity (Wikidata concepts) restrictions. At the time of writing, Wikidata has more than 200k classes[53] and the *Schemas* project counts with 375 ShEx schemas.[54] This is a ratio of approximately only 0.2% of all classes having a defined ShEx schema. On the other hand, 99% of properties (10788[55] out of 10812[56]) are restricted by at least one property constraint type. These numbers illustrate that the impact of understanding the semantics of property constraints is – at the moment – more significant than ShEx schemas. A separate analysis would be required for analyzing the *Schemas project* in more detail: taking, for instance, the entity schema *E10*[57] for the class *human* as an example, using ShEx as rather descriptive than prescriptive constraint language [14], some properties are highlighted as "desired" properties only in the schema. For instance, in the mentioned schema E10, the property *mother* (P25) is defined with a Kleene star

[52] https://www.wikidata.org/wiki/Wikidata:WikiProject_Schemas
[53] https://short.wu.ac.at/p2zn
[54] https://www.wikidata.org/wiki/Wikidata:Database_reports/EntitySchema_directory
[55] https://short.wu.ac.at/g2ya, last accessed 13 February 2023.
[56] https://www.wikidata.org/wiki/Wikidata:List_of_properties, last accessed 13 February 2023.
[57] https://www.wikidata.org/wiki/EntitySchema:E10

```
wdt:P25 @<human> *;
```

meaning that the absence of a "mother" does not lead to inconsistencies, which indicates that the objective of such schema is rather to assist in the "design" of classes than constraint checking in the strict sense. Also, although there are some ShEx to SHACL conversion tools,[58] additional challenges related to recursion impose further limitations on the process [29], as a lot of the theoretical work on SHACL either restricts or excludes recursive shapes (e.g. [20, 22]. The mentioned entity schema E10, for instance, restricts fathers, mothers, siblings, etc. also (recursively) to be humans. The SHACL shapes we obtain from the translation of Wikidata constraints are all non-recursive and make use of also SHACL-Core features that have not been yet universally implemented in validators.

Erxleben et al. [24] exploit properties describing taxonomic relations in Wikidata to extract an OWL ontology from Wikidata. The authors also propose the extraction of schematic information from property constraints and discuss their expressibility in terms of OWL axioms. However, whereas we focus herein concretely on covering all property constraints as a means to find possible violations in the data, Erxleben and colleagues rather stress the value of their corresponding OWL ontology as a (declarative) high-level description of the data, without claiming complete coverage of all Wikidata property constraints.

Martin and Patel-Schneider [44] discuss the representation of Wikidata property constraints through multi-attributed relational structures (MARS), as a logical framework for Wikidata. Constraints are represented in MARS using extended multi-attributed predicate logic (eMAPL), providing a logical characterization for constraints. Despite covering 26 different constraint types, to the best of our knowledge, the authors have not performed experiments to evaluate the accuracy of the proposed formalization, nor its efficiency, and do not discuss implementability. In fact, the theoretical framework partially skips over the subtleties of checking certain constraints in practice. As an example, the translation of *allowed entity types constraints* in the extended version of [44] assumes that entity types in Wikidata can be checked via simple instance-of type checking. Our SPARQL query shows that this is not the case in practice for all entity types as they are differently represented in the actual Wikidata RDF dump.[59] Our work – focusing on the *practical* implementability of property constraints in SPARQL – complements such theoretical approaches.

Abián et al. [1] propose a definition of contemporary constraint that was indeed later adopted by Wikidata property constraints. Shenoy et al. [55] present a quality analysis of Wikidata focusing on correctness, checking for weak statements under three main indicators: constraint violation, community agreement, and deprecation. The premise is that a statement receives a low-quality score when it violates some constraint, highlighting the importance of constraints for KG refinement. Boneva et al. [12] present a tool for designing/editing shape constraints in SHACL and ShEx suggesting Wikidata as a potential use case, but – to the best of our knowledge – without exhaustively covering or discussing the existing Wikidata property constraints.

Apart from works specifically on constraint for Wikidata, in [48] the authors systematically identify errors in DBpedia, using the DOLCE ontology as background knowledge to find inconsistencies in the assertional axioms. They feed target information extracted from DBpedia and linked to the DOLCE ontology into a reasoner checking for inconsistencies. Before, Bischof et al. [11] already highlighted logical inconsistencies in DBpedia which can be detected using OWL QL, rewritten to SPARQL 1.1 property paths – not unlike our general approach.

7.3. SHACL and SPARQL benchmarks

Despite the partially negative result that some of our SPARQL queries time out, and also – as we discussed above – we did not find SHACL validators that would allow us to check our constraint violations at the scale of Wikidata, we believe, besides our primary goal of clarifying Wikidata property constraint semantics, our results should be considered as a real-world challenge *benchmark* for both SPARQL engines and SHACL validators.

As for SHACL, real-world performance benchmarks still seem to be rare. Schaffenrath et al. [53] have presented a benchmark consisting of 58 SHACL shapes over a graph with 1M N-quads sample from a tourism knowledge graph, evaluated with different graph databases, emphasizing that "larger data exceeded [...] available resources".

[58] https://rdfshape.weso.es/shexConvert
[59] cf. https://short.wu.ac.at/pdhs.

The shapes we present are on the one hand targeting an (orders of magnitude) larger dataset, but on the other hand can also be evaluated locally on a per entity level, thus providing a benchmark of quite different nature. Also, the evolving nature of Wikidata makes a dynamic/evolving benchmark that can be evaluated/scaled along the natural evolution and growth of Wikidata itself. Next, [27] presents a synthetic SHACL benchmark derived from the famous LUBM ontology benchmark, while also emphasizing the current lack of real-world benchmarks for SHACL.

Closest but also orthogonal in focus to our own work is a recent paper by Rabbani et al. [52], which focuses on the orthogonal problem of automatically *extracting* shapes (representable as SHACL) from large KGs such as Wikidata in a data-driven manner, rather than on the formalization of community-driven constraints as we do.

Finally, apart from serving as a basis for novel benchmarks for SHACL, our SPARQL formalization particularly extends, and in our opinion complements, the existing landscape of real-world SPARQL benchmarks. Indeed, the – to the best of our knowledge – only benchmark for Wikidata, WDbench [6] covers a significantly different kind of Wikidata queries than we do. WDbench is a benchmark extracted from Wikidata query logs, focusing on queries that time out on the regular Wikidata query endpoint, but it is restricted to queries on truthy statements only, that is for instance not covering queries on qualifiers. Our queries on the contrary, by definition all relate to querying qualifiers and, as such they require the whole Wikidata graph and cannot be answered on the truthy statements alone. Yet, similar to the WDbench queries, many of the queries we present, particularly on very common properties, suffer from timeouts, as our experiments confirm. Thus, while the approach we present shows in principle feasible, it calls for novel more scalable approaches to efficiently solve such SPARQL queries that currently time out. We hope, as the queries we focus on typically only affect local contexts of entities and properties, they could hopefully be solved, e.g., by clever modularisation and partitioning techniques.

8. Conclusions and future work

We have formalized all 32 different property constraint types of Wikidata using SPARQL and discussed ways to encode them with W3C's recommendation mechanism for formalizing constraints over RDF Knowledge Graphs, SHACL. This study made it possible to clarify to which extent SHACL-Core can represent community-defined constraints of a widely used real-world KG. One of our results is a collection of practical SHACL-Core constraints that can be used in a large and growing real-world dataset. Indeed the non-availability of practical SHACL performance benchmarks has already been emphasized by [27], where we believe our work could be a significant step forward towards leveraging Wikidata as a large benchmark dataset for SHACL validators. Other results include clarifications of heretofore uncertain issues, such as the representability of permitted entities and exceptions in Wikidata property constraints within SHACL [55]. We also could argue the non-expressibility of certain Wikidata constraints, due to the impossibility of comparing values obtained through different paths matching the same regular path expression within SHACL-Core.

As we could show, all these issues could be addressed when using SPARQL to formalize and validate constraints, where all 32 constraints could in principle be formalized. In this context, as a partially negative result, one of the main limitations of the work was the increasing performance limitations of Wikidata's query endpoint, which calls for more scalable query interfaces and bespoke evaluation mechanisms. On the positive side, these limitations give rise to further research considering property constraint violation detection as a performance SPARQL benchmark as such. As a first next step in this direction, we aim to compare our results from the Wikidata SPARQL endpoint with a local installation, comparing different graph databases or lightweight query approaches such as HDT [25] to support a queryable version of Wikidata constraints checks independent of the SPARQL endpoint, which, for reasons of immediate comparison with the current Wikidata violation reports, was beyond our scope of the present paper.

Wikidata property constraints are dynamically evolving and maintained by the community, as shown by new constraint types such as *Citation needed* (Q54554025), a constraint type not even yet reported by Wikidata's official constraint reporting tool, cf. Footnote 42. We believe that our formalization and operationalization of property constraints in a declarative way, using SPARQL, establish a mutual relationship with the Wikidata community. Analyzing the formalization helps to enrich the way constraints are modeled and vice versa clarifies their semantic interpretation, as opposed to the current, partially ambiguous natural language definitions, verifiable only through

the partially disclosed Wikidata database reports. We further hope that this article stimulates discussions in the community to enrich the representation of constraints that still might have subjective interpretations.

In future work, we plan to use and build on the results of this paper to further systematically collect and analyze the kinds of constraint violations in Wikidata and study their patterns as well as their evolution over time. Understanding data that violates the constraints and its evolution is fundamental to identifying modeling or other systematic data quality issues and proposing further refinements, but also repairs, especially in collaboratively and dynamically created KGs such as Wikidata. Proposing refinements is a process that can be envisioned when taking into account the repair information declaratively represented in and retrievable through operationalizable constraints.

We have established SPARQL, as a declarative and operationalizable means to implement Wikidata's property constraints and also briefly discussed its relationship to other potential formalisms, such as Datalog and Description Logics. In order to further clarify the exact formal properties of Wikidata's property constraints, further research on a concise and bespoke formal language, e.g. in terms of extended DLs, which captures all and only the required features, would be an interesting route for further work; attempts such as MARS [44] provide promising starting points already in this direction.

Acknowledgements

The present paper is based on a preliminary workshop paper originally presented at the Wikidata workshop 2022 [26]; we thank the workshop organizers and participants for their feedback during the workshop. We thank the reviewers, for their valuable comments which helped us to considerably improve the original submission of this paper. This work is supported by funding in the European Commission's Horizon 2020 Research Program under Grant Agreement Number 957402 (TEAMING.AI) and Shqiponja Ahmetaj was supported by the Austrian Science Fund (FWF) and netidee SCIENCE project T1349-N.

References

[1] D. Abián, J. Bernad and R.T. Lado, Using contemporary constraints to ensure data consistency, in: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019*, Limassol, Cyprus, April 8–12, 2019, C. Hung and G.A. Papadopoulos, eds, ACM, 2019, pp. 2303–2310. doi:10.1145/3297280.3297509.

[2] S. Abiteboul, P. Buneman and D. Suciu, *Data on the Web: From Relations to Semistructured Data and XML*, Morgan Kaufmann, 1999. ISBN 1-55860-622-X.

[3] S. Abiteboul, R. Hull and V. Vianu, *Foundations of Databases*, Addison-Wesley, 1995, http://webdam.inria.fr/Alice/. ISBN 0-201-53771-0.

[4] S. Ahmetaj, R. David, M. Ortiz, A. Polleres, B. Shehu and M. Simkus, Reasoning about explanations for non-validation in SHACL, in: *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Online Event*, November 3–12, 2021, 2021, pp. 12–21. doi:10.24963/KR.2021/2.

[5] S. Ahmetaj, R. David, A. Polleres and M. Simkus, Repairing SHACL constraint violations using answer set programming, in: *The Semantic Web – ISWC 2022 – 21st International Semantic Web Conference, Virtual Event*, October 23–27, 2022, U. Sattler, A. Hogan, C.M. Keet, V. Presutti, J.P.A. Almeida, H. Takeda, P. Monnin, G. Pirrò and C. d'Amato, eds, Proceedings, Lecture Notes in Computer Science, Vol. 13489, Springer, 2022, pp. 375–391. doi:10.1007/978-3-031-19433-7_22.

[6] R. Angles, C.B. Aranda, A. Hogan, C. Rojas and D. Vrgoč, WDBench: A Wikidata graph query benchmark, in: *The Semantic Web – ISWC 2022*, U. Sattler, A. Hogan, M. Keet, V. Presutti, J.P.A. Almeida, H. Takeda, P. Monnin, G. Pirrò and C. d'Amato, eds, Springer International Publishing, 2022, pp. 714–731. ISBN 978-3-031-19433-7. doi:10.1007/978-3-031-19433-7_41.

[7] R. Angles, A. Bonifati, S. Dumbrava, G. Fletcher, K.W. Hare, J. Hidders, V.E. Lee, B. Li, L. Libkin, W. Martens, F. Murlak, J. Perryman, O. Savkovic, M. Schmidt, J.F. Sequeda, S. Staworko and D. Tomaszuk, PG-keys: Keys for property graphs, in: *SIGMOD'21: International Conference on Management of Data, Virtual Event*, China, June 20–25, 2021, ACM, 2021, pp. 2423–2436. doi:10.1145/3448016.3457561.

[8] R. Angles and C. Gutierrez, The expressive power of SPARQL, in: *The Semantic Web – ISWC 2008, 7th International Semantic Web Conference, ISWC 2008*, Karlsruhe, Germany, October 26–30, 2008, A.P. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T.W. Finin and K. Thirunarayan, eds, Proceedings, Lecture Notes in Computer Science, Vol. 5318, Springer, 2008, pp. 114–129. doi:10.1007/978-3-540-88564-1_8.

[9] F. Baader, I. Horrocks, C. Lutz and U. Sattler, *An Introduction to Description Logic*, Cambridge University Press, 2017. ISBN 978-0-521-69542-8.

[10] D. Beckett, T. Berners-Lee, E. Prud'hommeaux and G. Carothers, RDF 1.1 Turtle: Terse RDF Triple Language, 2014, available at: https://www.w3.org/TR/turtle/.

[11] S. Bischof, M. Krötzsch, A. Polleres and S. Rudolph, Schema-agnostic query rewriting in SPARQL 1.1, in: *The Semantic Web – ISWC 2014 – 13th International Semantic Web Conference*, Riva del Garda, Italy, October 19–23, 2014, Proceedings, Part I, Lecture Notes in Computer Science, Vol. 8796, Springer, 2014, pp. 584–600. doi:10.1007/978-3-319-11964-9_37.

[12] I. Boneva, J. Dusart, D. Fernández-Álvarez and J.E.L. Gayo, Shape designer for ShEx and SHACL constraints, in: *Proceedings of the ISWC 2019 Satellite Tracks (Posters & Demonstrations, Industry, and Outrageous Ideas) Co-Located with 18th International Semantic Web Conference (ISWC 2019)*, Auckland, New Zealand, October 26–30, 2019, CEUR Workshop Proceedings, Vol. 2456, CEUR-WS.org, 2019, pp. 269–272, https://ceur-ws.org/Vol-2456/paper70.pdf.

[13] I. Boneva, J.E. Labra Gayo and E.G. Prud'hommeaux, Semantics and validation of shapes schemas for RDF, in: *ISWC*, Springer, 2017. doi:10.1007/978-3-319-68288-4_7.

[14] A. Bonifati, P. Furniss, A. Green, R. Harmer, E. Oshurko and H. Voigt, Schema validation and evolution for graph databases, in: *Conceptual Modeling – 38th International Conference, ER 2019*, Salvador, Brazil, November 4–7, 2019, A.H.F. Laender, B. Pernici, E. Lim and J.P.M. de Oliveira, eds, Proceedings, Lecture Notes in Computer Science, Vol. 11788, Springer, 2019, pp. 448–456. doi:10.1007/978-3-030-33223-5_37.

[15] D. Brickley and R.V. Guha, RDF Schema 1.1, 2014, http://www.w3.org/TR/rdf-schema/.

[16] D. Calvanese, G.D. Giacomo, D. Lembo, M. Lenzerini and R. Rosati, Path-based identification constraints in description logics, in: *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008*, Sydney, Australia, September 16–19, 2008, G. Brewka and J. Lang, eds, AAAI Press, 2008, pp. 231–241, http://www.aaai.org/Library/KR/2008/kr08-023.php.

[17] D. Calvanese, G.D. Giacomo and M. Lenzerini, Identification constraints and functional dependencies in description logics, in: *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001*, Seattle, Washington, USA, August 4–10, 2001, B. Nebel, ed., Morgan Kaufmann, 2001, pp. 155–160.

[18] J.J. Carroll, C. Bizer, P.J. Hayes and P. Stickler, Named graphs, *Journal of Web Semantics* **3**(4) (2005), 247–267. doi:10.1016/J.WEBSEM.2005.09.001.

[19] P. Cimiano, J.P. McCrae and P. Buitelaar, Lexicon Model for Ontologies: Community Report, 2016, available at: https://www.w3.org/2016/05/ontolex/.

[20] J. Corman, F. Florenzano, J.L. Reutter and O. Savkovic, SHACL2SPARQL: Validating a SPARQL endpoint against recursive SHACL constraints, in: *Proceedings of the ISWC 2019 Satellite Tracks (Posters & Demonstrations, Industry, and Outrageous Ideas) Co-Located with 18th International Semantic Web Conference (ISWC 2019)*, Auckland, New Zealand, October 26–30, 2019, M.C. Suárez-Figueroa, G. Cheng, A.L. Gentile, C. Guéret, C.M. Keet and A. Bernstein, eds, CEUR Workshop Proceedings, Vol. 2456, CEUR-WS.org, 2019, pp. 165–168, http://ceur-ws.org/Vol-2456/paper43.pdf.

[21] J. Corman, J.L. Reutter and O. Savkovic, Semantics and validation of recursive SHACL, in: *The Semantic Web – ISWC 2018*, D. Vrandecic, K. Bontcheva, M.C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L. Kaffee and E. Simperl, eds, Lecture Notes in Computer Science, Vol. 11136, Springer, 2018. doi:10.1007/978-3-030-00671-6_19.

[22] J. Corman, J.L. Reutter and O. Savkovic, Semantics and validation of recursive SHACL, in: *The Semantic Web – ISWC 2018 – 17th International Semantic Web Conference*, Monterey, CA, USA, October 8–12, 2018, D. Vrandecic, K. Bontcheva, M.C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L. Kaffee and E. Simperl, eds, Proceedings, Part I, Lecture Notes in Computer Science, Vol. 11136, Springer, 2018, pp. 318–336. doi:10.1007/978-3-030-00671-6_19.

[23] D. Diefenbach, M.D. Wilde and S. Alipio, Wikibase as an infrastructure for knowledge graphs: The EU knowledge graph, in: *The Semantic Web – ISWC 2021 – 20th International Semantic Web Conference, ISWC 2021, Virtual Event*, October 24–28, 2021, A. Hotho, E. Blomqvist, S. Dietze, A. Fokoue, Y. Ding, P.M. Barnaghi, A. Haller, M. Dragoni and H. Alani, eds, Proceedings, Lecture Notes in Computer Science, Vol. 12922, Springer, 2021, pp. 631–647. doi:10.1007/978-3-030-88361-4_37.

[24] F. Erxleben, M. Günther, M. Krötzsch, J. Mendez and D. Vrandecic, Introducing Wikidata to the linked data web, in: *The Semantic Web – ISWC 2014 – 13th International Semantic Web Conference*, Riva del Garda, Italy, October 19–23, 2014, P. Mika, T. Tudorache, A. Bernstein, C. Welty, C.A. Knoblock, D. Vrandecic, P. Groth, N.F. Noy, K. Janowicz and C.A. Goble, eds, Proceedings, Part I, Lecture Notes in Computer Science, Vol. 8796, Springer, 2014, pp. 50–65. doi:10.1007/978-3-319-11964-9_4.

[25] J.D. Fernández, M.A. Martınez-Prieto, C. Gutiérrez, A. Polleres and M. Arias, Binary RDF representation for publication and exchange (HDT), *Journal of Web Semantics* **19**(2) (2013). doi:10.1016/j.websem.2013.01.002.

[26] N. Ferranti, A. Polleres, J.F. de Souza and S. Ahmetaj, Formalizing property constraints in Wikidata, in: *Proceedings of the 3rd Wikidata Workshop 2022 Co-Located with the 21st International Semantic Web Conference (ISWC2022), Virtual Event*, Hanghzou, China, October 2022, L. Kaffee, S. Razniewski, G. Amaral and K.S. Alghamdi, eds, CEUR Workshop Proceedings, Vol. 3262, CEUR-WS.org, 2022.

[27] M. Figuera, P.D. Rohde and M. Vidal, Trav-SHACL: Efficiently validating networks of SHACL constraints, in: *WWW'21: The Web Conference 2021, Virtual Event*, Ljubljana, Slovenia, April 19–23, 2021, J. Leskovec, M. Grobelnik, M. Najork, J. Tang and L. Zia, eds, ACM / IW3C2, 2021, pp. 3337–3348. doi:10.1145/3442381.3449877.

[28] J.E.L. Gayo, WShEx: A language to describe and validate Wikibase entities, in: *Proceedings of the 3rd Wikidata Workshop 2022 Co-Located with the 21st International Semantic Web Conference (ISWC2022), Virtual Event*, Hanghzou, China, October 2022, L. Kaffee, S. Razniewski, G. Amaral and K.S. Alghamdi, eds, CEUR Workshop Proceedings, Vol. 3262, CEUR-WS.org, 2022, https://ceur-ws.org/Vol-3262/paper3.pdf.

[29] J.E.L. Gayo, E. Prud'hommeaux, I. Boneva and D. Kontokostas, *Validating RDF Data, Synthesis Lectures on the Semantic Web: Theory and Technology*, Morgan & Claypool Publishers, 2017. ISBN 978-3-031-79477-3. doi:10.2200/S00786ED1V01Y201707WBE016.

[30] C. Gutierrez, C. Hurtado and A. Vaisman, Temporal RDF, in: *The Semantic Web: Research and Applications*, A. Gómez-Pérez and J. Euzenat, eds, Springer, Berlin Heidelberg, 2005, pp. 93–107. ISBN 978-3-540-31547-6. doi:10.1007/11431053_7.

[31] A. Haller, A. Polleres, D. Dobriy, N. Ferranti and S.J.R. Méndez, An analysis of links in Wikidata, in: *The Semantic Web – 19th International Conference, ESWC 2022*, Hersonissos, Crete, Greece, May 29–June 2, 2022, P. Groth, M. Vidal, F.M. Suchanek, P.A. Szekely, P. Kapanipathi, C. Pesquita, H. Skaf-Molli and M. Tamper, eds, Proceedings, Lecture Notes in Computer Science, Vol. 13261, Springer, 2022, pp. 21–38. doi:10.1007/978-3-031-06981-9_2.

[32] S. Harris and A. Seaborne, SPARQL 1.1 Query Language, 2013, available at: http://www.w3.org/TR/sparql11-query/.

[33] O. 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, J.L. Reutter and D. Srivastava, eds, CEUR Workshop Proceedings, Vol. 1912, CEUR-WS.org, 2017, https://ceur-ws.org/Vol-1912/paper12.pdf.

[34] P.J. Hayes and P. Patel-Schneider, RDF 1.1 Semantics, 2014, https://www.w3.org/TR/rdf11-mt/.

[35] D. Hernández, C. Gutierrez and R. Angles, The Problem of Correlation and Substitution in SPARQL – Extended Version, Technical Report, CoRR, 2018, arXiv:1801.04387.

[36] D. Hernández, A. Hogan and M. Krötzsch, Reifying RDF: What works well with Wikidata? in: *Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems*, CEUR Workshop Proceedings, Vol. 1457, CEUR-WS.org, 2015, pp. 32–47, http://ceur-ws.org/Vol-1457/SSWS2015_paper3.pdf.

[37] P. Hitzler, M. Krötzsch, P.F.P.-S. Bijan Parsia and S. Rudolph, *OWL 2 Web Ontology Language Primer*, 2nd edn, 2012, http://www.w3.org/TR/owl-primer/.

[38] A. Hogan, *Reasoning Techniques for the Web of Data, Studies on the Semantic Web*, Vol. 19, IOS Press, 2014. ISBN 978-1-61499-382-7. doi:10.3233/978-1-61499-383-4-I.

[39] A. Hogan, M. Arenas, A. Mallea and A. Polleres, Everything you always wanted to know about blank nodes, *Journal of Web Semantics* **27–28** (2014), 42–69. doi:10.1016/J.WEBSEM.2014.06.004.

[40] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutierrez, S. Kirrane, J.E.L. Gayo, R. Navigli, S. Neumaier, A.N. Ngomo, A. Polleres, S.M. Rashid, A. Rula, L. Schmelzeisen, J.F. Sequeda, S. Staab and A. Zimmermann, *Knowledge Graphs, ACM Comput. Surv.* **54**(4) (2022), 71:1–71:37. doi:10.1145/3447772.

[41] H. Knublauch and D. Kontokostas, Shapes Constraint Language (SHACL), 2017, http://www.w3.org/TR/shacl/.

[42] J.E. Labra Gayo and J.M. Alvarez Rodríguez, Validating statistical index data represented in RDF using SPARQL queries, in: *RDF Validation Workshop. Practical Assurances for Quality RDF Data*, Cambridge, Ma, Boston, Citeseer, 2013.

[43] T. Lebo, S. Sahoo, D. McGuinness, K. Belhajjame, J. Cheney, D. Corsar, D. Garijo, S. Soiland-Reyes, S. Zednik and J. Zhao, PROV-O: The PROV Ontology, 2013, available at: http://www.w3.org/TR/prov-o/.

[44] D.L. Martin and P.F. Patel-Schneider, Wikidata constraints on MARS, in: *Proceedings of the 1st Wikidata Workshop (Wikidata 2020) Co-Located with 19th International Semantic Web Conference(OPub 2020), Virtual Conference*, November 2–6, 2000, L. Kaffee, O. Tifrea-Marciuska, E. Simperl and D. Vrandecic, eds, CEUR Workshop Proceedings, Vol. 2773, CEUR-WS.org, 2020, https://ceur-ws.org/Vol-2773/paper-12.pdf.

[45] B. Motik, I. Horrocks and U. Sattler, Bridging the gap between OWL and relational databases, in: *Proceedings of WWW*, ACM, 2007. doi:10.1145/1242572.1242681.

[46] V. Nguyen, O. Bodenreider and A.P. Sheth, Don't like RDF reification?: Making statements about statements using singleton property, in: *23rd International World Wide Web Conference, WWW'14*, Seoul, Republic of Korea, April 7–11, 2014, ACM, 2014, pp. 759–770. doi:10.1145/2566486.2567973.

[47] P.F. Patel-Schneider and D. Martin, EXISTStential aspects of SPARQL, in: *Proceedings of the ISWC 2016 Posters & Demonstrations Track Co-Located with 15th International Semantic Web Conference (ISWC 2016)*, Kobe, Japan, October 19, 2016, T. Kawamura and H. Paulheim, eds, CEUR Workshop Proceedings, Vol. 1690, CEUR-WS.org, 2016, http://ceur-ws.org/Vol-1690/paper72.pdf.

[48] H. Paulheim and A. Gangemi, Serving DBpedia with DOLCE – more than just adding a cherry on top, in: *The Semantic Web – ISWC 2015 – 14th International Semantic Web Conference*, Bethlehem, PA, USA, October 11–15, 2015, M. Arenas, Ó. Corcho, E. Simperl, M. Strohmaier, M. d'Aquin, K. Srinivas, P. Groth, M. Dumontier, J. Heflin, K. Thirunarayan and S. Staab, eds, Proceedings, Part I, Lecture Notes in Computer Science, Vol. 9366, Springer, 2015, pp. 180–196. doi:10.1007/978-3-319-25007-6_11.

[49] J. Pérez, M. Arenas and C. Gutierrez, Semantics and complexity of SPARQL, *ACM Transactions on Database Systems* **34**(3) (2009). doi:10.1145/1567274.1567278.

[50] A. Piscopo and E. Simperl, Who models the world?: Collaborative ontology creation and user roles in Wikidata, *Proc. ACM Hum. Comput. Interact.* **2**(CSCW) (2018), 141:1–141:18. doi:10.1145/3274410.

[51] A. Polleres and J. Wallner, On the relation between SPARQL1.1 and answer set programming, *Journal of Applied Non-Classical Logics (JANCL)* **23**(1–2) (2013), 159–212, Special issue on Equilibrium Logic and Answer Set Programming. doi:10.1080/11663081.2013.798992.

[52] K. Rabbani, M. Lissandrini and K. Hose, Extraction of validating shapes from very large knowledge graphs, *Proc. VLDB Endow.* **16**(5) (2023), 1023–1032, https://www.vldb.org/pvldb/vol16/p1023-rabbani.pdf. doi:10.14778/3579075.3579078.

[53] R. Schaffenrath, D. Proksch, M. Kopp, I. Albasini, O. Panasiuk and A. Fensel, Benchmark for performance evaluation of SHACL implementations in graph databases, in: *Rules and Reasoning*, V. Gutiérrez-Basulto, T. Kliegr, A. Soylu, M. Giese and D. Roman, eds, Springer International Publishing, Cham, 2020, pp. 82–96. ISBN 978-3-030-57977-7. doi:10.1007/978-3-030-57977-7_6.

[54] G. Schreiber and Y. Raimond, RDF 1.1 Primer, 2014, http://www.w3.org/TR/rdf11-primer/.

[55] K. Shenoy, F. Ilievski, D. Garijo, D. Schwabe and P.A. Szekely, A study of the quality of Wikidata, *J. Web Semant.* **72** (2022), 100679. doi:10.1016/J.WEBSEM.2021.100679.

[56] S. Staworko, I. Boneva, J.E.L. Gayo, S. Hym, E.G. Prud'hommeaux and H.R. Solbrig, Complexity and expressiveness of ShEx for RDF, in: *18th International Conference on Database Theory, ICDT 2015*, Brussels, Belgium, March 23–27, 2015, M. Arenas and M. Ugarte, eds, LIPIcs, Vol. 31, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015, pp. 195–211. doi:10.4230/LIPICS.ICDT.2015.195.

[57] S. Staworko, I. Boneva, J.E. Labra Gayo, S. Hym, E.G. Prud'hommeaux and H. Solbrig, Complexity and expressiveness of ShEx for RDF, in: *ICDT*, LIPIcs. http://drops.dagstuhl.de/opus/volltexte/2015/4985. doi:10.4230/LIPIcs.ICDT.2015.195.

[58] J. Tao, E. Sirin, J. Bao and D.L. McGuinness, Integrity constraints in OWL, in: *AAAI*, 2010, http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1931.

[59] H. Turki, D. Jemielniak, M.A.H. Taieb, J.E.L. Gayo, M.B. Aouicha, M. Banat, T. Shafee, E. Prud'hommeaux, T. Lubiana, D. Das and D. Mietchen, Using logical constraints to validate statistical information about disease outbreaks in collaborative knowledge graphs: The case of COVID-19 epidemiology in Wikidata, *PeerJ Comput. Sci.* **8** (2022), e1085. doi:10.7717/PEERJ-CS.1085.

[60] F. Vargas-Rojas, A. Polleres, L. Cabrera-Bosquet and D. Symeonidou, PhyQus: Automatic unit conversions for Wikidata physical quantities, in: *Proceedings of the Wikidata Workshop 2023 Co-Located with 22nd International Semantic Web Conference (ISWC 2023)*, Athens, Greece, November 13, 2023, L. Kaffee, S. Razniewski, K. Alghamdi and H. Arnaout, eds, CEUR Workshop Proceedings, Vol. 3640, CEUR-WS.org, 2023, https://ceur-ws.org/Vol-3640/paper9.pdf.

[61] A. Zimmermann, N. Lopes, A. Polleres and U. Straccia, A general framework for representing, reasoning and querying with annotated Semantic Web data, *J. Web Semant.* **11** (2012), 72–95. doi:10.1016/J.WEBSEM.2011.08.006.

Chapter 2

Formalizing Repairs for Wikidata Constraint Violations: A Taxonomy and Empirical Analysis

Published in the Proceedings of The 24th International Semantic Web Conference. Cham: Springer Nature Switzerland, 2025. p. 369-387.

Formalizing Repairs for Wikidata Constraint Violations: A Taxonomy and Empirical Analysis*

Nicolas Ferranti[1][0000−0002−5574−1987], Dayane Guimarães[3][0009−0005−8300−7578], Jairo Francisco de Souza[3][0000−0002−0911−7980], and Axel Polleres[1,2][0000−0001−5670−1146]

[1] Vienna University of Economics and Business, Austria
[2] Complexity Science Hub Vienna, Austria
[3] LApIC Research Group, Federal University of Juiz de Fora, Brazil

Abstract. Collaboratively maintained knowledge graphs like Wikidata rely on property constraints to detect data inconsistencies. This paper systematically formalizes potential repairs for Wikidata constraint violations, presenting a comprehensive taxonomy of repair strategies encompassing both instance-level (A-box) and terminological-level (T-box) changes. T-box repairs, which alter constraint definitions or Wikidata's class hierarchy, can simultaneously address multiple violations and, to the best of our knowledge, have not been investigated in detail before. We observe repairs over time and evaluate how specific patterns within our taxonomy are applied in practice. Our analysis of historical data reveals insights into the prevalence of repair patterns in Wikidata's collaborative environment. The results indicate that T-box repairs are particularly relevant for certain constraint types and the overall consolidation of Wikidata, where modifying constraint definitions can reduce the number of recurring violations.

Keywords: Knowledge Graphs · Wikidata · Data Quality · Constraints · Knowledge Graph Refinement.

1 Introduction

Knowledge Graphs (KGs) [9] power applications such as search engines, reasoning, and data integration [2]. They model real-world knowledge as graphs, with entities as nodes and relationships as edges. Despite their versatility, commonsense KGs like Wikidata often suffer from data quality issues, undermining their usefulness. Their large scale and collaborative nature make ensuring data correctness a performance and scalability challenge [1, 6, 21].[4]

This paper focuses on RDF-based KGs [3] exported from Wikibase, such as Wikidata (WD) [24] and the EU Knowledge Graph [4]. These graphs rely on

* This work was funded by the Austrian Science Fund (FWF) [10.55776/COE12].

[4] As of July 2025, Wikidata contained approximately 118 million items and 2 billion edits. Source: `https://www.wikidata.org/wiki/Wikidata:Statistics`. A full RDF serialization of Wikidata amounts to over 14b triples [6].

so-called property constraints, a common mechanism for detecting inconsistencies by enforcing or prohibiting specific data patterns [16, 21]. Broadly, *completeness* constraints *require* certain information (e.g., "an item used as place of birth also *must* be of type location"), while *consistency* constraints *prohibit* specific conflicting statements (e.g. explicitly forbidding certain items to be used as country of citizenship). Items violating these constraints are considered *violations*.

Shape Expressions (ShEx) [26] and the W3C standard Shapes Constraint Language (SHACL) [25] are the primary languages for expressing constraints in RDF KGs. However, Wikidata established its own property constraint model – developed within the Wikibase ecosystem – prior to the development of these standards. Although not fully expressible in SHACL [6], WD's property constraints have been formally studied using SPARQL [6] and MAPL [12]. These constraint types have also been adopted by other Wikibase-based KGs, such as the EU Knowledge Graph, highlighting the relevance of research on their properties and repairs. Previous work [2, 21] has explored instance (A-box) repairs based on a simplified Description Logics (DL) formalization, but did not address repairs involving modifications to constraint definitions or Wikidata's foundational "ontology axioms". Building upon these efforts, this paper formally defines both A-box and T-box repair patterns for WD property constraints, leveraging existing formalizations in MAPL [12] and SPARQL [6]. Furthermore, we analyze the impact of T-box repair patterns, which offer the potential to fix multiple violations at once, contrasting with individual A-box repairs.

Our main contributions are as follows: (1) A complete formalization of possible repair patterns covering both A-box and T-box repairs – to the best of our knowledge, this is the first work to analyze T-box repairs in Wikidata; (2) A systematic analysis of these repair patterns based on historical edits; (3) Additionally, we provide an accompanying dataset of historical repairs performed by the Wikidata community from Jun-2019 to May-2023 for further use, cf. the *Supplementary Material* statement in the end of the paper.

The rest of the paper is structured as follows: Section 2 covers the Wikidata data model and its constraint representation. Section 3 defines constraint violations. Section 4 presents our formalization of Wikidata constraint repairs, covering both A-box and T-box, with general and constraint-specific repair patterns. Section 5 details our experiments on Wikidata, tracking data evolution, violations, and repair patterns. Section 6 explores data quality in KGs, including inconsistency detection and formal constraint representation. Finally, Section 7 summarizes the findings and future directions.

2 Preliminaries

Wikidata's KG consists of two primary components: items and properties. Items represent concrete or abstract entities, e.g. Messi, Chemistry, or California. Properties denote relationships between items, such as date of birth or country of citizenship. These relations are used to create *direct claims*, i.e., subject-property-value triples, which we will also write as property(subject,value) predicates. Here,

the subject is an item and a value can be either an item or a literal. Items and properties are identified by alphanumeric IDs, where item IDs are of the form Qx (e.g. Q615 for Messi) and property IDs of the form Px (e.g. P27 for country of citizenship). Fig. 1 illustrates WD's RDF data model in two layers: single-lined edges depict *direct claims* as item- property-value triples, while the double-lined edges incorporate a mechanism to add so-called *statement qualifiers* for ranks, references and other metadata, using WD's reification mechanism, such as the date (*start time* property) on which an individual's citizenship became effective (*country of citizenship* property), for enhanced description. Further details can be found in [6], which also explains the respective distinct namespaces used for direct claims (wdt:) and qualifier statements (p:, ps:, pq:, etc.).

While Wikidata does not separate between A-Box and T-Box, we borrow from this abstraction commonly made in Description Logics, to distinguish "assertional" data (A-Box), i.e., statements made about individual instance entities, from "terminology" definitions (T-Box), i.e., axioms about classes and properties, such as Wikidata's subclass hierarchy and property constraint definitions.[5]

A-Box. In analogy with standard RDF ontologies, we will call direct claims as well as statement qualifiers related to these (depicted as single and double line edges with ▲-tips in Fig. 1), the WD **A-Box**.

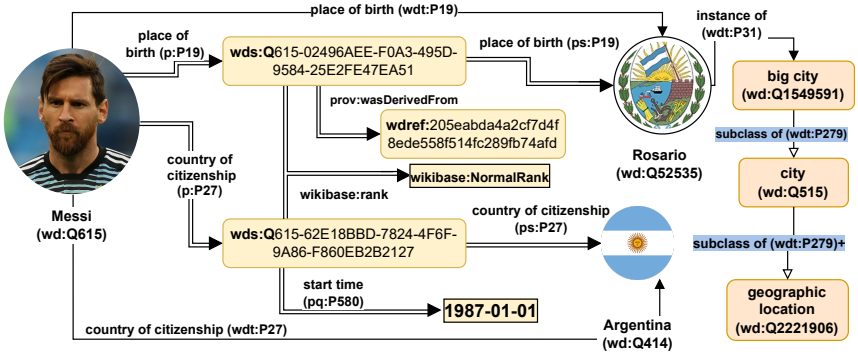
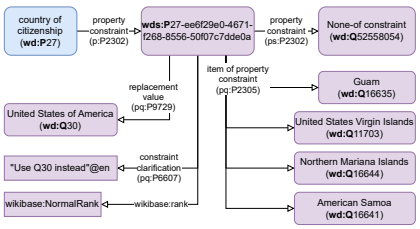


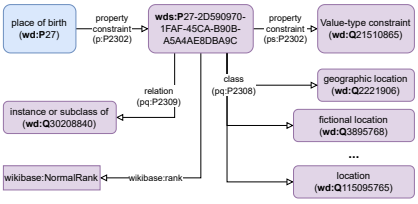
Fig. 1: Wikidata data model: single-line arrows indicate direct A-box (▲-tips) and T-box (△-tips) claims, respectively. Double line arrows denote A-box statement qualifiers, i.e., contextual information.

T-Box. Similar, but slightly different to standard RDFS and OWL ontologies, the parts of Wikidata's *terminology* (or T-Box) relevant for property constraint evaluation are described by special statements about *classes* and *properties* using

[5] subproperties are not covered herein, because these are not considered by/relevant for any Wikidata property constraint definitions.



(a) None-of constraint for country of citizenship. Since the listed items are US territories, US citizenship should be used instead.



(b) Value-type constraint for place of birth. There must be at least one value that is an instance or subclass of one of the items listed in the class qualifier.

Fig. 2: Examples of WD property constraint definitions (part of the T-Box)

a particular vocabulary: (1) items used in instance of (P31) A-box claims form a *class hierarchy*: that is, we count all subclassOf (P279) direct claims (some of which are illustrated in the righthand side of Fig. 1) part of the T-Box.

The second part of what we call WD's T-Box herein are (2) *property constraint definitions*, as illustrated in Fig. 2, which shows property constraints on the properties country of citizenship and place of birth: each such definition comprises a qualified property constraint (pc) type (P2303) statement along with additional, constraint-type specific qualifiers: Fig. 2a shows a None-of constraint, stating that none of the four item of property constraint (iopc) should be used when describing citizenship. Such constraints may also contain hints for repairing, in the form of additional constraint type specific qualifiers, such as replacement value (P9729): since the prohibited entities are US territories, the value United States of America should be used instead. Fig. 2b exemplifies a Value-type constraint for property place of birth, where there must be at least one value that is an instance or subclass of one of the listed class values. Other constraint types for instance include Item-requires-statement (IRS) or Value-requires-statement (VRS) constraints; overall, 30+ property constraint types have been defined and are constantly evolving in WD.[6] We herein selected 13 constraint types – the 10 analyzed by Tanon et al. [21] and included three constraint types not previously covered: the complement of One-of, which we refer to as None-of, and two qualifier-based constraint types, Required qualifier and Allowed qualifiers, which shall show the complementary aspects of our analysis with regards to earlier works.

The A-Box data in Fig. 1 complies to the *consistency constraint* in Fig. 2a, as the single country of citizenship claim has none of the forbidden values. Likewise, the A-Box claim place_of_birth(Messi, Rosario) shown in Figure 1 conforms to the *completeness constraint* in Fig. 2b since the additional claim instance_of (Rosario, big city) *exists* and there is a subclass path from big city to geographic location; i.e., we see that for compliance checking of property constraints also the subclass hierarchy part of the T-Box is relevant.

[6] `https://www.wikidata.org/wiki/Help:Property_constraints_portal`

3 Defining Violations With Witness Patterns

We formally define constraint violations before discussing repairs. Here, we focus on the constraint types used in earlier works and try to generalize, building on these earlier formalizations [6, 12, 14, 21].

Marx et al. [14] have introduced Multi-Attributed Relational Structures (MARS), especially as a formal data model for generalized Property Graphs like WD. They also developed Multi-Attributed Predicate Logic (MAPL) for expressing semantic knowledge within these structures. MAPL extends First-Order Logic (FOL) to support multi-valued attributes via *set terms*. Specifically, MAPL enhances the standard components of FOL (constants, terms, atoms, and formulae) as follows [12], where we only use binary predicates herein:

1. A *set term* is either a *set variable* or a set of attribute-value pairs $\{a_1 : b_1, ..., a_n : b_n\}$, where each a_i, b_i is an *object term*. Object terms are the usual basic terms of FOL, and can be either constants or *object variables*.
2. A relational atom is written as $p(a, b)@S$, where p is a binary predicate, a, b are object terms and S is a set term. Herein, we also allow SPARQL property path expressions for p.[7]
3. A set atom is an expression $(a : b) \in S$, where a, b are object terms and S is a set term.

For further details, on the syntax and semantics of MAPL formulae, including basic concepts such as models, entailment, satisfaction and consistency, we refer to [14]. In the following, constants are denoted in a serifless font (e.g. location, country_of_citizenship, etc.), object variables use lower case letters (e.g., $x, y, z, ...$), and set variables are denoted by uppercase letters (e.g., $S, Q, ...$).

To illustrate how we can use MAPL to formalize the semantics of particular property constraint (pc) types, consider the None-of constraint from [12]:

$$\mathsf{pc}(p, \text{None-of})@CQ \wedge (\mathsf{iopc} : v) \in CQ \rightarrow \neg\exists s.p(s, v)$$

This MAPL rule states that if a property p is constrained by a None-of constraint, specified by constraint qualifiers CQ, and CQ includes the value v as an item of the property constraint, then there should not exist any subject s that has the value v for property p. As the rule ensures data consistency, its logical negation, easily derivable via logical equivalences and De Morgan's laws, can be formulated as a conjunction reading all variables as existential to define what we call a *violation witness pattern*:

$$p(s, v) \wedge \mathsf{pc}(p, \text{None-of})@CQ \wedge (\mathsf{iopc} : v) \in CQ$$

In MAPL, A WD statement $p(s, v)@SQ$ extends the basic $p(s, v)$ claim by incorporating statement qualifiers and references (SQ). To further analyze constraint violations, we will distinguish between: (1) the *base statement*, $p(s, v)@SQ$,

[7] This slight extension of MAPL can be seen as a form of syntactic shorthand for (linearly recursive) rules used to represent path expressions. See, for example, [19] and rule (3) in [15].

Table 1: Witness patterns in MAPL and SPARQL

Constr. Type	Witness Pattern	SPARQL Pattern
One-of (Q21510859)	$\underline{p(s,v)} \wedge \mathsf{pc}(p, \text{One-of})@CQ \wedge (\mathsf{iopc} : v) \notin CQ$	[wikibase:directClaim ?P; p:P2302 ?CQ]. ?S ?P ?V. ?CQ ps:P2302 wd:Q21510859. FILTER NOT EXISTS {?CQ pq:P2305 ?V.}
None-of (Q52558054)	$\underline{p(s,v)} \wedge \mathsf{pc}(p, \text{None-of})@CQ \wedge (\mathsf{iopc} : v) \in CQ$	[wikibase:directClaim ?P; p:P2302 ?CQ]. ?S ?P ?V. ?CQ ps:P2302 wd:Q52558054. ?CQ pq:P2305 ?V.
IRS (Q21503247) *(no value)*	$\mathsf{pc}(p, \text{IRS})@CQ \wedge (\text{property} : p_c) \in CQ \wedge \underline{p(s,v)} \wedge \neg\exists v_c.\underline{p_c(s,v_c)} \wedge \neg\exists v_{cq}.(\mathsf{iopc} : v_{cq}) \in CQ$	[wikibase:directClaim ?P; p:P2302 ?CQ]. ?S ?P ?V. ?CQ ps:P2302 wd:Q21503247. ?CQ pq:P2306 []. FILTER NOT EXISTS {?CQ pq:P2305 [].} FILTER NOT EXISTS { ?CQ pq:P2306/wikibase:directClaim ?PC. ?S ?PC ?VC.}
IRS (Q21503247) *(with value)*	$\mathsf{pc}(p, \text{IRS})@CQ \wedge (\text{property} : p_c) \in CQ \wedge \underline{p(s,v)} \wedge \neg\exists v_c.(\underline{p_c(s,v_c)} \wedge (\mathsf{iopc} : v_c) \in CQ)$	[wikibase:directClaim ?P; p:P2302 ?CQ]. ?S ?P ?V. ?CQ ps:P2302 wd:Q21503247. ?CQ pq:P2306/wikibase:directClaim ?PC. ?CQ pq:P2305 []. FILTER NOT EXISTS { ?S ?PC ?VC. ?CQ pq:P2305 ?VC.}
VRS (Q21510865) *(no value)*	$\mathsf{pc}(p, \text{VRS})@CQ \wedge (\text{property} : p_c) \in CQ \wedge \underline{p(s,v)} \wedge \neg\exists v_c.\underline{p_c(v,v_c)} \wedge \neg\exists v_{cq}.(\mathsf{iopc} : v_{cq}) \in CQ$	[wikibase:directClaim ?P; p:P2302 ?CQ]. ?S ?P ?V. ?CQ ps:P2302 wd:Q21510865. ?CQ pq:P2306 []. FILTER NOT EXISTS {?CQ pq:P2305 [].} FILTER NOT EXISTS {?CQ pq:P2306/wikibase:directClaim ?PC. ?V ?PC ?VC.}
VRS (Q21510865) *(with value)*	$\mathsf{pc}(p, \text{VRS})@CQ \wedge (\text{property} : p_c) \in CQ \wedge \underline{p(s,v)} \wedge \neg\exists v_c.(\underline{p_c(v,v_c)} \wedge (\mathsf{iopc} : v_c) \in CQ)$	[wikibase:directClaim ?P; p:P2302 ?CQ]. ?S ?P ?V. ?CQ ps:P2302 wd:Q21510865. ?CQ pq:P2306/wikibase:directClaim ?PC. ?CQ pq:P2305 []. FILTER NOT EXISTS {?V ?PC ?VC. ?CQ pq:P2305 ?VC.}
Inverse (Q21510855)	$\mathsf{pc}(p, \text{Inverse})@CQ \wedge (\text{property} : p_c) \in CQ \wedge \underline{p(s,v)} \wedge \neg\underline{p_c(v,s)}$	[wikibase:directClaim ?P; p:P2302 ?CQ]. ?S ?P ?V. ?CQ ps:P2302 wd:Q21510855. ?CQ pq:P2306/wikibase:directClaim ?PC. FILTER NOT EXISTS {?V ?PC ?S}
Symmetric (Q21510862)	$\mathsf{pc}(p, \text{Symmetric})@CQ \wedge \underline{p(s,v)} \wedge \neg\underline{p(v,s)}$	[wikibase:directClaim ?P; p:P2302 ?CQ]. ?S ?P ?V. ?CQ ps:P2302 wd:Q21510862. FILTER NOT EXISTS {?V ?P ?S}
Conflicts-with (Q21502838) *(no value)*	$\mathsf{pc}(p, \text{Conflicts-with})@CQ \wedge (\text{property} : p_c) \in CQ \wedge \neg\exists v_{cq}.(\mathsf{iopc} : v_{cq}) \in CQ \wedge \underline{p(s,v)} \wedge \underline{p_c(s,v_c)}$	[wikibase:directClaim ?P; p:P2302 ?CQ]. ?S ?P ?V. ?CQ ps:P2302 wd:Q21502838. ?CQ pq:P2306/wikibase:directClaim ?PC . FILTER NOT EXISTS {?CQ pq:P2305 []}. ?S ?PC ?VC.
Conflicts-with (Q21502838) *(with value)*	$\mathsf{pc}(p, \text{Conflicts-with})@CQ \wedge (\text{property} : p_c) \in CQ \wedge (\mathsf{iopc} : v_c) \in CQ \wedge \underline{p(s,v)} \wedge \underline{p_c(s,v_c)}$	[wikibase:directClaim ?P; p:P2302 ?CQ]. ?S ?P ?V. ?CQ ps:P2302 wd:Q21502838. ?CQ pq:P2306/wikibase:directClaim ?PC . ?CQ pq:P2305 ?VC. ?S ?PC ?VC.
Distinct-values (Q21502410)	$\mathsf{pc}(p, \text{Distinct-values})@CQ \wedge \underline{p(s,v)} \wedge \underline{p(s_c,v)} \wedge s \neq s_c$	[wikibase:directClaim ?P; p:P2302 ?CQ]. ?S ?P ?V. ?CQ ps:P2302 wd:Q21502410. ?SC ?P ?V. FILTER (?S != ?SC).
Single-value (Q19474404) *(no seperator)*	$\mathsf{pc}(p, \text{Single-value})@CQ \wedge \neg\exists q.(\text{separator} : q) \in CQ \wedge \underline{p(s,v)} \wedge \underline{p(s,v_c)} \wedge v \neq v_c$	[wikibase:claim ?PSQ; p:P2302 ?CQ.] ?S ?PSQ ?SQ.?SQC. FILTER(str(?SQ) != str(?SQC)) ?CQ ps:P2302 wd:Q19474404. FILTER NOT EXISTS {?CQ pq:P4155 []}
Single-value (Q19474404) *(with separator)*	$\mathsf{pc}(p, \text{Single-value})@CQ \wedge (\text{separator} : q) \in CQ \wedge \underline{p(s,v)}@SQ \wedge \underline{p(s,v_c)}@SQ_C \wedge (v \neq v_c) \wedge \neg\exists v_q, v_{qc}.(\underline{\underline{(q:v_q) \in SQ}} \wedge \underline{\underline{(q:v_{qc}) \in SQ_C}} \wedge v_q \neq v_{qc})$	[wikibase:claim ?PSQ; p:P2302 ?CQ.] ?S ?PSQ ?SQ.?SQC. FILTER(str(?SQ) != str(?SQC)) ?CQ ps:P2302 wd:Q19474404. ?CQ pq:P4155 []. FILTER NOT EXISTS { ?SQ ?Q ?VQ. ?SQC ?Q ?VQC. ?CQ pq:P4155/wikibase:qualifier ?Q. FILTER (?VQ != ?VQC) }
Required qualifier (Q21510856)	$\mathsf{pc}(p, \text{Required qualifier})@CQ \wedge (\text{property} : q) \in CQ \wedge \underline{p(s,v)}@SQ \wedge \neg\exists v_q.(\underline{\underline{(q : v_q) \in SQ}})$	[wikibase:claim ?PSQ; p:P2302 ?CQ]. ?S ?PSQ ?SQ. ?CQ ps:P2302 wd:Q21510856. ?CQ pq:P2306/wikibase:qualifier ?Q. FILTER NOT EXISTS { ?SQ ?Q [] }
Allowed qualifiers (Q21510851)	$\mathsf{pc}(p, \text{Allowed qualifiers})@CQ \wedge \underline{p(s,v)@SQ} \wedge (\underline{\underline{(q : v_q) \in SQ}} \wedge (\text{property} : q) \notin CQ)$	[wikibase:claim ?PSQ; p:P2302 ?CQ]. ?S ?PSQ ?SQ. ?SQ ?Q []. ?CQ ps:P2302 wd:Q21510851. FILTER NOT EXISTS { ?CQ pq:P2306/wikibase:qualifier ?Q }
Type$_{rel}$ (Q21503250)	$\underline{p(s,v)} \wedge \mathsf{pc}(p, \text{Type})@CQ \wedge (\text{relation} : rel) \in CQ \wedge \neg\exists c.\ ((\text{class} : c) \in CQ \wedge PATH_{rel}(s,c))$	[wikibase:directClaim ?P; p:P2302 ?CQ]. ?S ?P ?V. ?CQ ps:P2302 wd:Q21503250. ?CQ pq:P2309 wd:Q_{rel}. FILTER NOT EXISTS { ?CQ pq:P2308 ?C. ?S $PATH_{rel}$?C.}
Value-type$_{rel}$ (Q21510865)	$\underline{p(s,v)} \wedge \mathsf{pc}(p, \text{Value-type})@CQ \wedge (\text{relation} : rel) \in CQ \wedge \neg\exists c.\ ((\text{class} : c) \in CQ \wedge PATH_{rel}(v,c))$	[wikibase:directClaim ?P, p:P2302 ?CQ]. ?S ?P ?V. ?CQ ps:P2302 wd:Q21510865. ?CQ pq:P2309 wd:Q_{rel}. FILTER NOT EXISTS { ?CQ pq:P2308 ?C. ?V $PATH_{rel}$?C.}

Table 2: relation paths for Type and Value-type constraints described in Table 1.

rel	$PATH_{rel}$ (MAPL)	$PATH_{rel}$ (SPARQL)	Q_{rel}
instance_of	$\underline{\text{P31}}(v, v_c) \wedge \text{P279} * (v_c, c)$	(wdt:P31/wdt:P279*)	Q21503252
subclass_of	$\text{P279} * (v, c)$	(wdt:P279*)	Q21514624
instance_or_subclass_of	$(\underline{\text{P31}}(v, v_c) \wedge \text{P279} * (v_c, c)) \vee (\text{P279} * (v, c))$	(wdt:P31?/wdt:P279*)	Q30208840

which directly violates a property constraint, and (2) a *context statement*, denoted as $p_c(s, v_c)@SQ_c$ or $p_c(v, v_c)@SQ_c$, which is an additional statement that provides supporting information about the subject s or value v of the *base statement*, thereby aiding in violation identification. For example, identifying a Conflicts with constraint violation requires the presence of a prohibited specific *context statement* using a prohibited context property within the constraint definition qualifiers. Likewise, for the Value-type constraint, a supporting instance of (P31) statement may be required.

Definition 1 (Constraint Violation). *A statement $p(s, v)@SQ \in \mathcal{A}$ is considered a* constraint violation *if it satisfies at least one witness pattern defined in Table 1.*

Table 1 shows the complete list of witness patterns for all analyzed constraint types, including their Wikidata IDs, with Table 2 further explaining different relational paths for subtypes of Type and Value-type constraints. We "mark" different A-Box components within MAPL witness patterns with underlining, while non-underlined elements represent T-Box (constraint definition) components:

$\underline{\quad}$ *base statement* (S) $\quad\underline{\underline{\quad}}$ *base statement qualifier* (SQ)
$\underline{\quad}$ *context statement* (S_c) $\quad\underline{\underline{\quad}}$ *context statement qualifier* (SQ_c)

Additional note: truthy statements. As a notational shortcut in MAPL formulas used within Table 1, we use $p(s, v)$ (or $p_c(s, v_c)$) without the $@SQ$ (or analogously, $@CQ$) suffix, as a shorthand for wikidata:directClaims, which denote "truthy" statements of the form $p(s, v)@SQ$. The omission of SQ indicates that SQ contains an active rank attribute (i.e., rank:preferredRank, or, resp., rank:normalRank without the existence of any other rank:preferredRank claims). Only these truthy statements are accessible in Wikidata's RDF serialization through property IDs qualified by the wdt:-namespace.

By exploiting the introduced notational conventions, all MAPL witness patterns can also be directly translated into SPARQL queries to detect constraint violations [6], e.g., for the None-of running example:

```
[wikibase:directClaim ?P; p:P2302 ?CQ].   Retrieve constraint definition node
                                          and base statement property ?P.
?S ?P ?V.                                 Select the base statement using ?P.
?CQ ps:P2302 wd:Q52558054.                Filter for None-of constraint type.
?CQ pq:P2305 ?V.                          Match forbidden values.
```

The unified MAPL (object and set) variable names and SPARQL variable names in Tables1+2 illustrate the corresponding semantics. That is, the queries presented in Table 1 resemble those introduced by [6] but have been adapted in this paper to use a more uniform and standardized variable naming scheme aligned with their corresponding witness pattern roles. However, the original semantics are preserved.

We note the following for further explanation of the SPARQL queries: whereas a pattern using wikibase:directClaim points to the direct claim's wdt:-prefixed predicate p, a blank node pattern using wikibase:claim yields the p:-prefixed property that links entities to wds:-prefixed statement nodes representing SQ, just as the variable ?CQ is bound (via p:P2302) to a wds:-prefixed statement node representing the constraint CQ; for further details, please refer to Figures 1 and 2 or the more in-depth explanations of Wikidata's custom RDF reification model in [6].

4 Formalizing Repairs

Completeness violation witness patterns check for the *non-existence* of required information, which results in negated conjuncts in MAPL or FILTER NOT EXISTS clauses in SPARQL, as for the Value-type constraint test. On the contrary, witness patterns for *consistency* violations result in conjunctive queries with only positive conjunctions to match *existence* of prohibited information. This straightforwardly suggests a definition of repairs by the following insights:

1. Consistency violations can be repaired by deletions of statements or qualifiers mentioned in positive MAPL witness pattern conjuncts (or, resp. `SPARQL NOT EXISTS` patterns).
2. Completeness violations can be repaired by additions of statements or qualifiers mentioned in MAPL witness pattern conjuncts under a $\neg$ (negated) scope, or, resp. within `SPARQL NOT EXISTS` patterns.

Based on this idea, we can further classify repairs by the (underlined vs. non-underlined) components of witness patterns affected by such additions and deletions, which we introduced in the previous section, distinguishing between A-Box and T-Box repairs:

A-Box repairs encompass modifications of the base statement (S), context statement (S_C), or their respective qualifiers (SQ, SQ_c), falling into the following repair types:

S^- – **Base statement deletion**: the deletion of the base statement obviously fixes *any* constraint violation, since the base statement $\underline{p(s,v)}$ appears positively in each witness pattern in Table 1.

S_c^- – **Context Statement deletion**(s): the deletion of a "witnessing" context statement $\underline{\underline{p_c(s,v_c)}}$ fixes any consistency constraints which contain a positive context statement in their witness pattern, i.e., this potentially affects Conflicts-with, Distinct-values, and Single-Value constraints.

S_c^+ – **Context Statement addition**(s): likewise, the addition of a "missing" context statement fixes any completeness violations which contain a negated context statement, i.e., IRS, VRS, Inverse, and Symmetric constraints, but also includes instance of (P31) additions for Type and Value-Type constraint violations, cf. Table 2.

SQ^- – **Base qualifier deletion**: the deletion of a "witnessing" base statement qualifier fixes any consistency violations for witness patterns with a positive, $\underline{\underline{(q : v_q) \in SQ}}$ qualifier in Table 1, i.e., a removal of a non-Allowed qualifiers.

SQ^+ – **Base qualifier addition**: likewise, the addition of a "missing" base statement qualifier fixes witness patterns with a negated, $\underline{\underline{(q : v_q) \in SQ}}$ qualifier, i.e., a separator addition for Single-Value, or an addition of a Required qualifier.

SQ_c^+ – **Context qualifier addition** - the addition of a "missing" context statement qualifier fixes consistency violations that contain a negated, $\underline{\underline{(q_c : v_{qc}) \in SQ_c}}$ qualifier, i.e., specifically this applies to the addition of a separator for Single-Value (on the context statement).

Since no context statement qualifiers appear positively in witness patterns, SQ_c^- is not a repair category in the discussed WD constraint types. We note here that Tanon et al.'s work [21] (i) only considers A-Box modifications in terms of base and context statement additions/deletions but not wrt. qualifiers, and (ii) strictly distinguishes between consistency and completeness constraints. As for (i), we additionaly consider *qualifier repairs*; and as for (ii), in our more general considerations also considering qualifiers, we see that certain constraint types could be viewed as both consistency or completeness constraints. For instance, the Single-value constraint may be either fixed by removing a conflicting value or by adding a separator (qualifier). Thus, we will rather than classifying single constraint types as consistency or completeness constraints, distinguish "repair goals" as consistency repairs by deleting (or deactivating) conflicting information or completeness repairs by adding (or activating) missing information.

T-Box repairs (not considered in [21]) concern changes in constraint definitions and/or WD's class hierarchy, i.e. concerning all the non-underlined parts in the witness patterns of Table 1:

C^- – **Constraint deletion**: the removal of the constraint definition, i.e. the qualified statement property constraint (P2302) CQ (within the "parent" property definition) obviously fixes all constraint violations of this constraint instance.

CQ^+ – **Constraint Qualifiers Addition**: the addition of a constraint qualifier matching a negated atom $(q : v) \in CQ$ within a witness pattern could affect several constraint types, i.e. Allowed qualifiers, One-of, Type, and Value-Type. Note that there might be less obvious cases, for instance where an addition fixes a violation of an IRS or VRS constraint by adding $(\mathsf{iopc} : v_c) \in CQ$, which in turn might "activate" a priorly non-matching context statement.

CQ^- – **Constraint Qualifiers deletion**: the removal of a "required" constraint qualifier fixes any constraint violations which contain a positive constraint qualifier, i.e. for instance, Inverse, Conflicts-with, None-of, Required Qualifier.

$\subseteq^+$ – **Class Hierarchy addition**: as per Table 2, subclass Of additions can lead to repairs of a prior violation witness of Value-Type and Type constraints.

4.1 "Indirect" Repairs: Re-Ranking, Exceptions, and Replacements

Note that the discussion above streamlined and simplified the consideration of repairs in some sense, wrt. leaving out to further, "indirect" repairs.

Re-Ranking. First, we did not explicitly consider *re-ranking*. Changes on the rank of a statement (especially to rank:deprecatedRank) might implicitly remove or even add direct claims related to matching truthy statements: as such, we may view such rerankings as synonymous for additions/deletions, respectively, and count them in the repair categories defined above. Extending our SPARQL patterns to only consider truthy statements is straightforward, but would bloat the patterns in Table 1 – we left this out for readability.

As an exception, we explicitly considered a very common type of re-ranking that significantly impacted repairs in our experiments: *constraint deprecations* (denoted C^d), which can also be viewed as a special form of constraint deletion.

Constraint Exceptions. WD allows adding base statement subject items s as exceptions using the exception_to_constraint (P2303) constraint qualifier in constraint definitions: while this is in some sense a T-Box addition, marking instance items as exceptions might intuitively be rather seen as part of the A-Box: as such, we do not report those repairs as CQ^+ in our experiments, but rather define an own category e^+ for exceptions.

Value Replacements. So far we have classified all repairs as either additions and deletions: yet, based on the Wikidata UI, value changes of triples (statements or qualifiers) that can be done in one edit may rather be considered as a single change, combining a deletion with an addition.

In many cases, constraint types can be fixed by a *base statement value replacement* (S^r), for instance One-of, None-of, but also VRS (by implicitly changing the context statements to be considered).

Likewise, violations of Conflicts-with constraints with *value*s can be fixed by a *context statement value replacement* (S^r_c).

Single-Value constraint violations, may be fixable by a *base statement qualifier value replacement* (SQ^r) for the separator qualifier.

Finally, on the T-Box level, constraint qualifier replacements (CQ^r) may induce repairs: i.e., (Value-)Type constraint violations could be fixed by changes in the relation type or class qualifiers. Also, similar to the CQ^+ example above, changing to (property : p_{new}) $\in CQ$ by replacing the constrained property p_{old} may fix an IRS constraint, by "activating" priorly "missing" context statements. While we could classify such a repair as CQ^-, arguably the mere addition of the new property p_{new} – indirectly – would be sufficient to repair the violation.

Summarizing, repairs that are the effect of a statement or qualifier (value) *replacements* will be marked as S^r, S^r_c, SQ^r, and CQ^r, respectively.

5 Experiments

We evaluated our repair patterns by analyzing historical Wikidata constraint violations using the experimental setup described in this section. Figure 3 outlines the process.

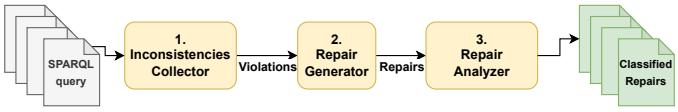


Fig. 3: Three steps experiment pipeline.

The experiment involved three steps: first, using the witness patterns, we extracted constraint violations from both 2019 and 2023 Wikidata HDT [5], denoted as $V_{ct,2019}$, and $V_{ct,2023}$ resp., per constraint type ct. Here, $V_{ct,year}$ should be understood as the set of variable bindings for the SPARQL queries in Table 1. Secondly, we identified historical repairs by comparing the violations detected on both snapshots, i.e., the set of repairs $R_{ct} = V_{ct,2019} - V_{ct,2023}$ includes those violations present in 2019 but no longer observed in 2023. Conversely, new violations are identified as $N_{ct} = V_{ct,2023} - V_{ct,2019}$. Thirdly, we developed Python scripts to analyze the repair types introduced in Section 4.

Table 3 summarizes the general statistics of our experiment. For constraint types also analyzed by Tanon et al. [21], we also observe a significant increase in data since 2018, reflected in the growing number of data items and detected violations over time.[8]

Table 4 represents the share of each repair type, detected to fix constraint violations between 2019 and 2023, distinguishing between T-Box, A-Box repairs, and added exceptions, respectively. We note that the shares per row do not necessarily add up to 100%, since some repairs might "overlap", in the sense that different actions might effect in repairing the same violation.

A-Box Repairs. The high share of A-Box repairs for None-of (92%) and Symmetric (95%) constraints shows that deletions or reranks were frequently made to the data by the community itself to correct violations. Value-type saw 52% of repairs involve adding missing type statements, reflecting how evolving knowledge in Wikidata often requires adding additional type-level information to maintain consistency.

Following the line of adding missing statements, the Inverse constraint saw 81% of repairs consisting of adding the required inverse properties to statements. Similarly, VRS witnessed the addition of required statements in 85% of the repairs. As both constraint types had small sets of constraint deletions, this may indicate

[8] Note that column $\#R_{ct}$ is not comparable, since [21] does not count the total numbers of repairs.

Table 3: #**constr**: total constraints per type. $\#S$: base statements using properties with this constraint type. $\#V_{ct,2019}/\#V_{ct,2023}$: number of violations 2019 vs. 2023. $\#R_{ct}$: identified corrections 19-23. Growth compared to columns #**constr**, #**triples**, #**violations** of Table 3 in [21] is indicated by ↑%.

Name in WD	#**constr.**		$\#S$		$\#V_{ct,2019}/\#V_{ct,2023}$		$\#R_{ct}$
One-of	173	↑66.3%	23M	↑538%	4.3k/202k	↑4950%	2.9k
None-of	432	-	314.5M	-	1.9k/508k	-	1.1k
IRS	11051	↑256.2%	522M	↑51%	5.4M/16.7M	↑350%	3.7M
VRS	352	↑44.8%	470M	↑452%	2.5M/4.7M	↑249%	1.3M
Inverse	118	↑13%	9.8M	↑200%	322k/338k	↑44%	256k
Symmetric	47		8.2M		3.7M/253k		3.7M
Conflicts-with	2052	↑241.4%	685M	↑52%	175k/1.8M	↑1167%	117k
Distinct-values	7607	↑178.8%	210M	↑275%	386k/533k	↑182%	259k
Single-value	7356	↑165.3%	265M	↑211%	2.8M/95M	↑28K%	675k
Required qualifier	477	-	11M	-	2M/4M	-	612k
Allowed qualifiers	853	-	1B41M	-	321k/6.8M	-	222k
Type	7056	↑174%	994M	↑299%	3.1M/34.3M	↑889%	2.6M
Value-type	1090	↑56.6%	246M	↑267%	636k/27.6M	↑801%	517k

that most constraint instances of those types are mature and that an effort to fix the A-box statements is in progress.

T-Box Repairs. Despite the significant number of A-Box repairs, T-Box repairs, which have been mostly ignored in prior works, play an equally critical role. These repairs indicate that the constraints are evolving alongside the data. Whether it involves adding qualifiers, adjusting property hierarchies, or deleting outdated constraints, T-Box repairs are vital for ensuring the knowledge graph remains flexible and up-to-date with changes in both structure and terminology.

As we can see, T-Box repairs are quite significant for certain constraints, reflecting how Wikidata's terminology and understanding of the schema have evolved. For instance, for the Conflicts-with constraint, 74% of the constraints were deleted, and 80% involved removing a forbidden property or value (iopc), indicating a major revision of constraints related to property conflicts. Similarly, for the Allowed qualifiers constraint, 70% of the constraints were deleted, and 19% saw qualifiers being added to the constraint definitions, which shows that the terminology itself was refined to allow for a broader range of qualifiers.

The Single-value constraint also highlights significant T-Box repair activity, with 73% of the violations being fixed by constraint deletions and 7% involving additions of separators, reflecting gradual refinement of constraints incl. qualifiers during their evolution/adoption. Some constraints indicate an ongoing effort to refine how value-related constraints are interpreted, e.g. 62% of repairs of VRS violations involve a change in the required value(s) at the T-Box level.

Lastly, also class hierarchy additions ($\subseteq^+$) seem to play a significant role, contributing to roughly one-third of repairs for (Value-)Type constraints.

Table 4: Share of repairs by type (cf. Sec. 4) in %. Values <0.01% shown as 0.

Name in WD	A-Box Repairs			T-Box Repairs			
	S^-/S^{r}	$S_c^+/S_c^-/S_c^{\mathrm{r}}$	$SQ^+/SQ^-/SQ_c^+$	C^-/C^{d}	$CQ^+/CQ^-/CQ^{\mathrm{r}}$	$\subseteq^+$	e^+
One-of	31/10	–/–/–	–/–/–	12/69	10/–/–	–	0
None-of	92/0	–/–/–	–/–/–	11/0	–/11/–	–	3
IRS	5/–	61/–/–	–/–/–	39/10	–/–/3.3	–	0
VRS	13/0	85/–/–	–/–/–	8/3	–/1.5/62	–	0
Inverse	40/–	81/–/–	–/–/–	2/28	–/–/0.05	–	0
Symmetric	95/–	4/–/–	–/–/–	0/0.01	–/–/–	–	0
Conflicts-with	17/–	–/12/2	–/–/–	74/3	–/80/–	–	0
Distinct-values	44/–	–/35/–	–/–/–	16/8	–/–/–	–	0.08
Single-value	24/–	–/16/–	0.9/–/2	73/5	7/–/0	–	0.04
Required qualifier	22/–	–/–/–	20/–/–	3/0.2	–/66/–	–	0
Allowed qualifiers	6/–	–/–/–	–/38/–	70/0	19/–/–	–	0
Type	4/–	12/–/–	–/–/–	0.9/0	31/–/63	35	0
Value-type	14/–	52/–/–	–/–/–	0.4/0	12/–/38	37	0

These overall trends seem to suggest that the meaning of WD's terminology – in particular through property constraints – observably evolves in tandem with the data itself, ensuring that the graph's semantics better align with the A-Box data as it grows and changes.

Our method, as opposed to Tanon et al.'s [21] which tracks all historical knowledge graph states from WD's start, rather analyzes repairs between two snapshots. While this makes it challenging to pinpoint the exact order in which edits that led to repairs occurred, we can confidently assert that the identified patterns were executed between the two timestamps and might have contributed to the observed repairs. While Tanon et al.'s method offers finer-grained change tracking, our two-snapshot approach is expected to be more resource-efficient and scalable for large knowledge graphs like WD, as also confirmed by [21] results, not reporting, for instance the actual total numbers of repairs. However, further experiments are needed to validate the efficiency of this approach. In particular, for the identification of *single edit* value replacements, so far we do not really track these down to single edits, as we do not analyze the whole edit history, [9] rather following a best effort approach: while this is also just an approximation, we can identify statements with changed values having the same wds: statement nodes in the graph but different values, when computing the set R_{ct} (with slightly expanded SPARQL patterns as opposed to Table 1 also returning those nodes).

5.1 Analysing most impactful T-box bulk repairs and new violations

While having analyzed T-Box changes on a coarse level, we would like to have a closer look at two further aspects concerning "most impactful" changes, as well as possible "side effects" of T-Box changes: note that we emphasize that T-box changes could also induce new violations. To assess such effects of T-box "bulk"

[9] Note that [21] also consider such *replacements*, but – based on personal communication – do also not explicitly distinguish between "single edit" replacements or combinations of additions and deletions.

repairs in more detail, we have also analyzed the top-3 constraint types (with the highest share of T-box-related repairs) (Conflicts with, Required Qualifier, and VRS), in an attempt to identify schema changes behind historical repairs on particular properties and examining newly emerged violations.

For Conflicts with constraints, out of the 80% CQ^- T-box repairs, most of those constraints where eventually deleted (i.e. overlapping with the 74% C^- repairs; interestingly, almost half of those (48%) were related to only two Conflicts-with constraints (both later deleted) on the properties located in the administrative territorial entity (P131) and headquarters location (P159). New Conflicts with violations largely stemmed from new constraints introduced post-2019 (8 of top 10), totaling 701k.

In Required qualifier constraints, T-box repairs mainly involved qualifier removal (CQ^-), but hardly any constraint deletions (C^-). Notably, here 375k repairs (93% of these repairs) stemmed from removal of the point in time (P585) qualifier from the required qualifiers for position held (P39). Conversely, we note that the addition of the language of work or name (P407) qualifier to a new Required qualifier constraint for official website (P856) and described at URL (P973) properties after 2019 led to over 1.8 million new violations.

In VRS repairs, the two constrained properties most affected by T-box-based repairs (cast member (P161) and director (P57)) also accounted for the highest number of new violations between 2019 and 2023. Both require an additional occupation (P106) statement. Changes in the list of allowed occupations (14 added and 2 removed) for cast member caused 396k repairs, but 863k new violations. Likewise, the VRS constraint on occupations for director saw 288k repairs, but also 236k new violations from 8 added and 1 removed occupation.

Overall, our findings also indicate that the community frequently deletes constraints that cause many violations, and most new violations stem from newly created constraints. Developing approaches to help the community refine constraints could reduce deletions, preserve historical evolution, and simplify the assessment of constraint changes over time. This would support a more sustainable and informed constraint management and schema evolution process.

6 Related Works and Discussion

WD's constraint system arose from practical needs, not formal logic, with property constraints remaining the most used mechanism due to their broader coverage [6]. Other constraint mechanisms in WD include, for instance, entity schemas[10] and WShEx [7].

In general, there is a lack of systematic, longitudinal analyses of practical KGs' evolution [18], which we address by analyzing and classifying repairs over time, adding a focus on schema/T-Box repairs and evolution. Other related works have derived suggestions for repairs: in addition to their partial formalization of WD A-Box repairs, Tanon et al. [21] propose a method based on rule mining

[10] `https://www.wikidata.org/wiki/Wikidata:WikiProject_Schemas`

to correct A-box constraint violations. Similarly, Chen et. al. [2] presented a general correction framework combining lexical matching, semantic embedding, soft constraint mining, and semantic consistency checks to correct entity/literal assertions in DBpedia. Beyond correction methods, Shenoy et. al. [20] proposed quality indicators based on removed statements, deprecated statements, and constraint violations in WD. While we rely mostly on "vanilla" SPARQL, Tanon et. al. [21] used a quad store to manipulate a sample of WD's edit history, and Shenoy et. al. [20] used the Knowledge Graph ToolKit [11] to compute deprecation and deletion metrics.

Regarding WD *constraint formalization*, Tanon et. al. [21] use DL to describe 10 different constraint types; however, as mentioned, the consideration of qualifiers and T-Box repairs is missing. Therefore, as we show, some constraints were tested without their full meaning. Martin and Patel-Schneider [12, 13] were the first to use MAPL to express property constraints, which we build upon to express violations (and repairs) more comprehensively. Lastly, Ferranti et. al. [6] similarly have deployed SHACL as well as SPARQL queries as a declarative approach to capture constraint violations through a WD RDF export for all the 32 constraint types, noting that DL is insufficient to express e.g. Single-Value constraints with separators. We put Martin et. al's approach to use MAPL, and Ferranti et al.'s SPARQL formalizations side by side, systematically naming the relevant components of WD's property constraint "language" to classify repairs. SPARQL, which has already been deployed in earlier, use-case-specific works on constraint checking in WD [23], also turns out to be computationally efficient enough to compute constraint violations and repairs at scale in our experiments. While SHACL has also been operationalized recently, e.g. by efficient compilation techniques to SQL for usage on large KGs[10], this is not sufficient to express all current property constraint types in WD, as noted by [6].

As for expressivity of WD's property constraints, let us remark an interesting observation: WD's current property constraints seem inherently insufficient to express all constraints checked in WD's UI: for instance, particular constraint types, such as IRS constraints are restricted to having exactly one context property qualifier,[12] whereas several context property qualifiers are allowed for Allowed qualifiers constraint type definitions.[13] As opposed to other constraint checks in WD's UI, that seem to be backed up by property constraint definitions, these ones seem to be checked/enforced independently in WD's UI, and are clearly not expressible as property constraints; discrepancies between property constraint definitions and WD's UI violation checks have already been reported earlier [6].

As for further related areas, works on pinpointing to (minimal sets of) axioms responsible for justifying (unwanted) consequences in OWL DL [11, 16], which could also be useful for finding repairs, are not directly related to what we are doing here. Our work is about analyzing and understanding repairs under constraints rather than finding repairs under deductive inference. Likewise, T-

[11] https://usc-isi-i2.github.io/kgtk/

[12] At time of writing, this was flagged as violation in WD's UI, cf. https://www.wikidata.org/w/index.php?title=Property:P582&oldid=2345339934

[13] cf. https://www.wikidata.org/w/index.php?title=Property:P1476&oldid=2342178751

box repair methods for ontologies exploring minimal axiom weakening [22] do not play a significant role in this setting: WD's constraints are designed for validation without additional inference. Therefore, while these approaches offer conceptual parallels, they would require adaptation to WD's specific framework. Here, attempts to map WD's properties to OWL[8] could be a potential, though disputed starting point, as WD deliberately does not adopt DL or a predefined formal ontology [17].

7 Conclusions

In this work, we have systematically formalized and analyzed property constraint violations and subsequent repairs in Wikidata (WD), comparing snapshots of the knowledge graph at two different points in time. We were not only able to categorize the types of repair patterns most commonly applied, but could also draw valuable insights on how both instance data (A-Box) repairs but also T-Box repairs, i.e. changes of WD's constraint definitions and class hierarchy, determine the consolidation of WD's terminology and its usage. To the best of our knowledge, we are the first to systematically consider such T-Box repairs, and additionally investigate the effects of qualifier changes at the A-Box level, by separately considering the roles of statement changes, qualifier changes, property constraint definitions, and changes in WD's class hierarchy.

In comparison to earlier works, our analysis also shows a significant increase in the adoption and repair activities related to property constraints in WD; in some cases, the number of constraint violations is growing faster than the number of triples added to WD properties instantiating the constraints. This trend underscores the importance of tracking historical repairs to develop semi-automatic refinement approaches to assist the Wikidata community in managing and evolving its data. To this end, our fine-grained analysis of repairs offers a comprehensive tool now not only to descriptively investigate past repairs, but – as a next step – shall also help to derive recommendations for repairs, as we can now systematically observe the effects of how data and schema evolve alongside. By doing so, we hope to support the improvement of WD's quality over time, strengthening its role as a reliable and ever-growing knowledge base.

Our work is fully reproducible and extensible: as for future work, a more comprehensive coverage of all 30+ current and evolving constraint types – which we had to leave out also for space restrictions – is on our agenda. Also, we plan to extend our current, illustrative investigation of two concrete WD snapshots (2019 vs. 2023), towards a regular, constant monitoring tool, for longitudinal analyses tracing WD's historic development periodically, or based on the full edit history, as it evolves.

Supplemental Material Statement: source code, queries and datasets used for our evaluations are available at `https://github.com/nicolasferranti/wikidata-repairs`, with the exception of historical WD HDT [5] snapshots.[14]

[14] available at https://www.rdfhdt.org/datasets/

References

1. Angles, R., Buil-Aranda, C., Hogan, A., Rojas, C., Vrgoc, D.: Wdbench: A wikidata graph query benchmark. In: Sattler, U., Hogan, A., Keet, C.M., Presutti, V., Almeida, J.P.A., Takeda, H., Monnin, P., Pirrò, G., d'Amato, C. (eds.) The Semantic Web - ISWC 2022 - 21st International Semantic Web Conference, Virtual Event, October 23-27, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13489, pp. 714–731. Springer (2022). https://doi.org/10.1007/978-3-031-19433-7_41
2. Chen, J., Chen, X., Horrocks, I., Myklebust, E.B., Jiménez-Ruiz, E.: Correcting knowledge base assertions. In: Huang, Y., King, I., Liu, T., van Steen, M. (eds.) WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020. pp. 1537–1547. ACM / IW3C2 (2020). https://doi.org/10.1145/3366423.3380226
3. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1 Concepts and Abstract Syntax (2014), `https://www.w3.org/TR/rdf11-concepts/`
4. Diefenbach, D., Wilde, M.D., Alipio, S.: Wikibase as an infrastructure for knowledge graphs: The EU knowledge graph. In: Hotho, A., Blomqvist, E., Dietze, S., Fokoue, A., Ding, Y., Barnaghi, P.M., Haller, A., Dragoni, M., Alani, H. (eds.) The Semantic Web - ISWC 2021 - 20th International Semantic Web Conference, ISWC 2021, Virtual Event, October 24-28, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12922, pp. 631–647. Springer (2021). https://doi.org/10.1007/978-3-030-88361-4_37
5. Fernández, J.D., Martınez-Prieto, M.A., Gutiérrez, C., Polleres, A., Arias, M.: Binary RDF Representation for Publication and Exchange (HDT) **19**(2) (2013). https://doi.org/https://dl.acm.org/doi/10.1016/j.websem.2013.01.002
6. Ferranti, N., De Souza, J.F., Ahmetaj, S., Polleres, A.: Formalizing and validating Wikidata's property constraints using SHACL and SPARQL. Semantic Web **15**(6), 2333–2380 (2024). https://doi.org/10.3233/SW-243611
7. Gayo, J.E.L.: Wshex: A language to describe and validate wikibase entities. In: Kaffee, L., Razniewski, S., Amaral, G., Alghamdi, K.S. (eds.) Proceedings of the 3rd Wikidata Workshop 2022 co-located with the 21st International Semantic Web Conference (ISWC2022), Virtual Event, Hanghzou, China, October 2022. CEUR Workshop Proceedings, vol. 3262. CEUR-WS.org (2022), `https://ceur-ws.org/Vol-3262/paper3.pdf`
8. Haller, A., Polleres, A., Dobriy, D., Ferranti, N., Méndez, S.J.R.: An analysis of links in Wikidata. In: 19th European Semantic Web Conference, ESWC 2022. Springer (May 2022). https://doi.org/https://doi.org/10.1007/978-3-031-06981-9_2
9. Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., de Melo, G., Gutierrez, C., Kirrane, S., Gayo, J.E.L., Navigli, R., Neumaier, S., Ngomo, A.N., Polleres, A., Rashid, S.M., Rula, A., Schmelzeisen, L., Sequeda, J.F., Staab, S., Zimmermann, A.: Knowledge graphs. ACM Comput. Surv. **54**(4), 71:1–71:37 (2022). https://doi.org/10.1145/3447772
10. Jakubowski, M., den Bussche, J.V.: Compiling SHACL into SQL. In: Demartini, G., Hose, K., Acosta, M., Palmonari, M., Cheng, G., Skaf-Molli, H., Ferranti, N., Hernández, D., Hogan, A. (eds.) The Semantic Web - ISWC 2024 - 23rd International Semantic Web Conference, Baltimore, MD, USA, November 11-15, 2024, Proceedings, Part II. Lecture Notes in Computer Science, vol. 15232, pp. 59–77. Springer (2024). https://doi.org/10.1007/978-3-031-77850-6_4
11. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of OWL DL entailments. In: Aberer, K., Choi, K., Noy, N.F., Allemang, D., Lee, K., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G.,

Cudré-Mauroux, P. (eds.) The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007. Lecture Notes in Computer Science, vol. 4825, pp. 267–280. Springer (2007). https://doi.org/10.1007/978-3-540-76298-0_20
12. Martin, D.L., Patel-Schneider, P.F.: Wikidata constraints on MARS. In: Kaffee, L., Tifrea-Marciuska, O., Simperl, E., Vrandecic, D. (eds.) Proceedings of the 1st Wikidata Workshop (Wikidata 2020) co-located with 19th International Semantic Web Conference(OPub 2020), Virtual Conference, November 2-6, 2020. CEUR Workshop Proceedings, vol. 2773. CEUR-WS.org (2020), `https://ceur-ws.org/Vol-2773/paper-12.pdf`
13. Martin, D.L., Patel-Schneider, P.F.: Wikidata constraints on MARS (extended technical report) (2020). https://doi.org/10.48550/arXiv.2008.03900
14. Marx, M., Krötzsch, M., Thost, V.: Logic on MARS: ontologies for generalised property graphs. In: Sierra, C. (ed.) Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. pp. 1188–1194. ijcai.org (2017). https://doi.org/10.24963/IJCAI.2017/165
15. Patel-Schneider, P.F., Martin, D.: Wikidata on MARS (2020). https://doi.org/10.48550/arXiv.2008.06599
16. Peñaloza, R.: Axiom pinpointing. In: Cota, G., Daquino, M., Pozzato, G.L. (eds.) Applications and Practices in Ontology Design, Extraction, and Reasoning, Studies on the Semantic Web, vol. 49, pp. 162–177. IOS Press (2020). https://doi.org/10.3233/SSW200042
17. Piscopo, A., Simperl, E.: Who models the world?: Collaborative ontology creation and user roles in wikidata. Proc. ACM Hum. Comput. Interact. **2**(CSCW), 141:1–141:18 (2018). https://doi.org/10.1145/3274410
18. Polleres, A., Pernisch, R., Bonifati, A., Dell'Aglio, D., Dobriy, D., Dumbrava, S., Etcheverry, L., Ferranti, N., Hose, K., Jiménez-Ruiz, E., Lissandrini, M., Scherp, A., Tommasini, R., Wachs, J.: How does knowledge evolve in open knowledge graphs? TGDK **1**(1), 11:1–11:59 (Dec 2023). https://doi.org/10.4230/TGDK.1.1.11
19. Polleres, A., Wallner, J.: On the relation between SPARQL1.1 and answer set programming. Journal of Applied Non-Classical Logics (JANCL) **23**(1–2), 159–212 (2013). https://doi.org/10.1080/11663081.2013.798992, special issue on Equilibrium Logic and Answer Set Programming
20. Shenoy, K., Ilievski, F., Garijo, D., Schwabe, D., Szekely, P.A.: A study of the quality of wikidata. J. Web Semant. **72**, 100679 (2022). https://doi.org/10.1016/J.WEBSEM.2021.100679
21. Tanon, T.P., Bourgaux, C., Suchanek, F.M.: Learning how to correct a knowledge base from the edit history. In: Liu, L., White, R.W., Mantrach, A., Silvestri, F., McAuley, J.J., Baeza-Yates, R., Zia, L. (eds.) The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019. pp. 1465–1475. ACM (2019). https://doi.org/10.1145/3308558.3313584
22. Troquard, N., Confalonieri, R., Galliani, P., Peñaloza, R., Porello, D., Kutz, O.: Repairing ontologies via axiom weakening. In: McIlraith, S.A., Weinberger, K.Q. (eds.) Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018. pp. 1981–1988. AAAI Press (2018). https://doi.org/10.1609/AAAI.V32I1.11567
23. Turki, H., Jemielniak, D., Taieb, M.A.H., Gayo, J.E.L., Aouicha, M.B., Banat, M., Shafee, T., Prud'hommeaux, E., Lubiana, T., Das, D., et al.: Using logical constraints

to validate statistical information about disease outbreaks in collaborative knowledge graphs: the case of covid-19 epidemiology in wikidata. PeerJ Computer Science **8**, e1085 (2022). https://doi.org/10.7717/peerj-cs.1085
24. Vrandecic, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Commun. ACM **57**(10), 78–85 (2014). https://doi.org/10.1145/2629489
25. W3C: SHACL Core Specification 1.0. `https://www.w3.org/TR/shacl/` (2017)
26. W3C: Shape Expressions (ShEx) 2.1 Specification. `https://shexspec.github.io/spec/` (2020)

Chapter 3

Who Can Fix This? User Recommendation for Knowledge Graph Repair via Embedding-Based Clustering

In: Proceedings of the 13th Knowledge Capture Conference 2025. 2025. p. 170-173.

Who Can Fix This? User Recommendation for Knowledge Graph Repair via Embedding-Based Clustering

Nicolas Ferranti
Vienna University of Economics and Business
Vienna, Austria
nicolas.ferranti@wu.ac.at

Dayane Guiamarães
Universidade Federal de Juiz de Fora
Juiz de Fora, Brazil
02252190698@estudante.ufjf.br

Jairo Francisco de Souza
Universidade Federal de Juiz de Fora
Juiz de Fora, Brazil
jairo.souza@ufjf.br

Axel Polleres
Vienna University of Economics and Business
Complexity Science Hub
Vienna, Austria
axel.polleres@wu.ac.at

Abstract

Maintaining the consistency of large-scale knowledge graphs (KGs) like Wikidata requires both automated methods and human expertise. In this paper, we address the task of recommending users best suited to repair a given inconsistency in a KG. Our approach leverages textual entity abstracts to compute sentence embeddings, which are clustered to identify semantically coherent regions of the KG. We introduce a framework that combines unsupervised clustering with 10-fold evaluation to test user recommendation strategies. Repair histories are linked to users, and test inconsistencies are assigned to clusters using approximate prediction. We evaluate two strategies: (i) frequency-based assignment, which recommends users based on how often they have edited entities in the predicted cluster, and (ii) embedding-based similarity, which compares the test inconsistency to past user-edited items via cosine similarity. Preliminary results show a cluster silhouette ≥ 0.5, membership hit rate of 80%, with the frequency-based approach achieving a Hits@3 of 60%. Our findings suggest that lightweight unsupervised methods can effectively recommend users, showing promise for semi-automated KG maintenance.

CCS Concepts

• **Information systems → Web searching and information discovery**; *Web data description languages.*

Keywords

Knowledge Representation, Knowledge Graphs, Wikidata, Data Quality, User Recommendation, Cluster Analysis

ACM Reference Format:
Nicolas Ferranti, Dayane Guiamarães, Jairo Francisco de Souza, and Axel Polleres. 2025. Who Can Fix This? User Recommendation for Knowledge Graph Repair via Embedding-Based Clustering. In *Knowledge Capture Conference 2025 (K-CAP '25), December 10–12, 2025, Dayton, OH, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3731443.3771363



This work is licensed under a Creative Commons Attribution 4.0 International License.
K-CAP '25, Dayton, OH, USA
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1867-0/2025/12
https://doi.org/10.1145/3731443.3771363

1 Introduction

Large-scale knowledge graphs (KGs) like Wikidata (WD) are central to semantic technologies, enabling tasks such as search, integration, and reasoning [3]. Yet, maintaining their *quality* remains challenging due to their scale, openness, and rapid evolution. Incomplete statements, inconsistent hierarchies, and incorrect type assignments can significantly affect downstream applications [4].

WD is a prominent community-driven KG that exemplifies this challenge: collaboratively maintained by thousands of users [5], it provides mechanisms such as *property constraints* and *Entity Schemas* to flag inconsistencies. These are complemented by external tools like WDShEx [2], which support schema-based validation aligned with the Wikibase model. These tools detect errors but don't help identify the best users to fix them.

This paper addresses the problem of recommending users for KG repair. Given a constraint violation in WD, can we identify users likely to provide quality fixes? We approach this by clustering entities based on semantic content and analyzing user repair histories to detect patterns of topical expertise. This strategy supports WD's collaborative model by suggesting users best suited for maintaining specific parts of the graph.

Our contributions are threefold: (1) We extend an existing dataset of Wikidata constraint repairs by linking each fix to the user who performed it, enabling user-centric analysis; (2) We propose a modular framework for recommending users to repair WD inconsistencies, combining data augmentation, semantic embedding, clustering, and ranking;[1] (3) We empirically evaluate unsupervised strategies that use textual embeddings and semantic clustering to recommend knowledgeable users for repair tasks.

The remainder of the paper is organized as follows. Section 2 introduces the WD data model, constraints, and validation mechanisms. Section 3 describes our user recommendation framework, including dataset processing, embeddings, and clustering. Section 4 reports on the experimental setup and results, with a focus on different recommendation strategies and their effectiveness. Section 5 reviews related literature on KG repair, user modeling, and WD quality. Lastly, Section 6 concludes with a summary of findings and directions for future work.

[1] https://github.com/nicolasferranti/wd_user_recommendation

2 The Wikidata Data Model

WD is a collaboratively edited KG hosted on the Wikibase platform.[2] While WD is not natively an RDF graph, the Wikibase software provides periodic RDF dumps and a SPARQL endpoint via the WD Query Service (WDQS). These exports enable interoperability with semantic web technologies by mapping WD's internal model to a standardized RDF representation.

At its core, WD models real-world knowledge using a statement-based data model. The primary elements are items (e.g., `Q42` for Douglas Adams) and properties (e.g., `P31` for *instance of*), which form statements of the form ⟨*item, property, value*⟩ [9], where a value can be either an item or a literal. Each item is identified by a unique QID (e.g., `Q42`), while each property is represented by a PID (e.g., `P31`). These statements can be enriched with qualifiers and references through a reified representation, allowing WD to express provenance and contextual nuances [1].

Constraint Mechanisms. WD ensures data quality through several constraint mechanisms: (1) Property Constraints, declarative rules (e.g., cardinality, type) attached to properties via constraint statements in WD's reification model; and (2) Entity Schemas, structural patterns defined using Shape Expressions (ShEx).

Validation. Property constraints can be validated through various means, including SPARQL queries [1], subtracting satisfying statements from the overall statements [8], querying the edit history [9], and through the WikibaseQualityConstraints extension[3]. These checks are non-enforcing, serving as guidance for editors. SPARQL has also been applied to validate domain-specific constraints [10]. While mappings to SHACL have been proposed [1], SHACL lacks full expressiveness for WD's semantics and is not yet integrated into the core ecosystem. Entity Schemas are externally validated using tools like the ShEx Simple Validator[4], but adoption remains limited compared to property constraints.

Edits/Repairs. Constraints fall into two categories [9]: consistency constraints, which prohibit certain triples (e.g., conflicting claims), and completeness constraints, which require them (e.g., mandatory `P31`). Violations are flagged in the UI and can be repaired through instance-level edits (e.g., correcting values), constraint refinements (e.g., modifying rules), or structural changes (e.g., class hierarchy updates). Each edit is logged as a revision, tied to a user or IP. Tanon et al. [9] released a dataset of historical repairs linked to revision IDs. We extend this dataset by attributing each repair to a user and enriching items with semantic descriptions, enabling our user-centered recommendation experiments.

3 Approach

This section presents our user recommendation framework for repairing Wikidata inconsistencies, comprising three components: *Data Augmentation*, *Semantic Mapping*, and *Recommendation and Evaluation*. First, we enrich a dataset of historical repairs with user attributions and semantic descriptions. Next, entities are mapped to a vector space and clustered into topical regions. Finally, we generate and evaluate user recommendations based on cluster membership and past repair activity.

[2]https://www.mediawiki.org/wiki/Wikibase
[3]https://www.mediawiki.org/wiki/Extension:WikibaseQualityConstraints
[4]https://shex-simple.toolforge.org/wikidata/

3.1 Data Augmentation

We start from a dataset of historical repairs containing `revisionId` and the repaired ⟨*item, property, value*⟩ triples. For each revision, we retrieve the corresponding username via the WD API and extract a five-sentence abstract for each QID using DBpedia sitelinks [4]. This yields triplets of the form (q, u, a), where q is the repaired item, u the user, and a the abstract.

When a QID lacks a DBpedia abstract, we query WD via SPARQL to retrieve its label, description, and relevant triples, including property labels and the labels/descriptions of their values. We exclude uninformative properties (e.g., external identifiers, media) and URLs, as they do not contribute meaningfully to a descriptive text. The retrieved data is grouped by instance type (e.g., "human", "film", "organization"), and properties are ranked by frequency within each group. For items with multiple types, rankings are combined to ensure consistent property selection. We then select up to fifteen top-ranked properties per QID, balancing informativeness and generality for semantic clustering.

This structured representation is passed to the `mixtral:8x7b` language model to generate DBpedia-style abstracts. Outputs with fewer than two sentences are discarded, ensuring a minimum quality standard. This approach provides rich, coherent descriptions when DBpedia abstracts are unavailable.

3.2 Semantic Mapping

To represent entities semantically, we embed each textual abstract into a dense vector space using the `all-distilroberta-v1` sentence transformer model. This choice is supported by recent work [4], which indicates that sentence-level embeddings outperform traditional KG embedding models in capturing KG concept similarity.

These embeddings serve as the basis for clustering entities into semantically coherent groups. The framework supports the integration of different dimensionality reduction and clustering techniques, allowing for empirical exploration of their impact on downstream recommendation performance. In our evaluation, we experiment with PCA and UMAP for dimensionality reduction, and KMeans, Gaussian Mixture Models (GMM), Spectral Clustering, and HDBSCAN for clustering. Each resulting cluster is intended to represent a latent topical region in the KG, grouping entities that share similar semantic characteristics based on their descriptions.

To enable test-time inference, new entities can be assigned to clusters using embedding similarity or approximate nearest neighbor techniques. This step ensures that unseen entities can be incorporated into the clustering-based recommendation process.

3.3 Recommendation and Evaluation

Once each item is associated with a semantic cluster, we use two strategies to recommend users likely to repair a given test item.

3.3.1 Frequency-Based Ranking. The objective is to rank users by how frequently they have interacted with items (QIDs) in a predicted cluster. Given a predicted cluster C, let $U_C = \{u_1, u_2, ..., u_n\}$ be the set of users who have interacted with items in C. Let $f(u_i, C) = \sum_{q \in C} 1_{\{(q,u_i) \in D\}}$ be the frequency count of user u_i in C, where 1 is the indicator function and D is the training dataset containing (q, u) interaction pairs.

We define the top-k most frequent users in cluster C as:

$$\mathbf{Top}^k_{freq.}(u, C) = \mathbf{argmax}_{U' \subseteq U_C,\ |U'|=k} \sum_{u \in U'} f(u, C)$$

For a test query (q^*, u^*), where q^* is an unseen item assigned to cluster C_{q^*}, we evaluate the recommendation using the Hits@k metric as follows:

$$\text{Hits@k}_{\text{freq}} = \begin{cases} 1, & \text{if } u^* \in \mathbf{Top}^k_{freq.}(u, C_{q^*}) \\ 0, & \text{otherwise} \end{cases}$$

3.3.2 Cosine Similarity-Based Ranking. Rank users based on how semantically close they are to the query item (QID), using average cosine similarity between embeddings. Let $\mathbf{v}_{q^*} \in \mathbb{R}^d$ be the embedding of the test item q^*. For each user $u \in U_C$, let $Q_u \subset C$ be the set of items the user interacted with. Let $\mathbf{v}_q$ be the embedding of $q \in Q_u$. We define the average cosine similarity between u and q^* as:

$$\text{sim}(u, q^*) = \frac{1}{|Q_u|} \sum_{q \in Q_u} \cos\left(\mathbf{v}_{q^*}, \mathbf{v}_q\right)$$

The top-k users most similar to q^* are defined as:

$$\mathbf{Top}^k_{cos.}(u, q^*) = \mathbf{argmax}_{U' \subseteq U, |U'|=k} \sum_{u \in U'} \text{sim}(u, q^*)$$

For a test query (q^*, u^*), we define the evaluation metric:

$$\text{Hits@k}_{\text{cosine}} = \begin{cases} 1, & \text{if } u^* \in \mathbf{Top}^k_{cos.}(u, q^*) \\ 0, & \text{otherwise} \end{cases}$$

3.3.3 Membership Evaluation Metric. In addition to ranking-based evaluation, we introduce a membership-based metric to assess whether a target user appears within the predicted semantic cluster of a given entity (QID). This metric captures the fundamental intuition that, if the clustering is semantically meaningful, users who have edited a specific entity should also be present in the cluster containing that entity.

Given a test query (q^*, u^*), let C denote the predicted cluster assignment for q^*, and let U_C be the set of users associated with cluster C. As u^* represents the user who edited q^*, we define a binary membership hit as:

$$\text{MembershipHit(q*)} = \begin{cases} 1, & \text{if } u^* \in U_C \\ 0, & \text{otherwise} \end{cases}$$

The final Membership Hit Rate is computed as the average over all test entities:

$$\text{HitRate}_{\text{membership}} = \frac{1}{|Q_{\text{test}}|} \sum_{q \in Q_{\text{test}}} \text{MembershipHit}(q)$$

4 Experiments & Discussion

We base our evaluation on the dataset of one-of constraint repairs compiled by Tanon et al. [9], comprising approximately 13,700 historical repairs on WD entities. Among these, 7,720 entities have an associated DBpedia abstract, while 5,989 lack a corresponding abstract, enabling a direct comparison between text-enriched and structure-only embeddings.

Table 1 reports the top clustering configurations ranked by silhouette score. We empirically tested both PCA and UMAP for dimensionality reduction and found that UMAP consistently produced better-separated clusters, hence, it is used in all reported experiments. Across all datasets, HDBSCAN outperformed fixed-k methods (KMeans, GMM), with no improvement in silhouette even when increasing the requested number of clusters up to 40. The *noAbstracts + HDBSCAN* configuration achieved the highest silhouette score (0.58) with 69 clusters on average, reflecting well-separated topical regions of QIDs. Combining all embeddings (*all + HDBSCAN*) yielded more granular clusters (114 on average) with only a slight drop in silhouette (0.57), while improving recommendation accuracy (Hit@1_{freq} = 0.46).

As shown in Table 1, cosine-based Hits@k were consistently lower than frequency-based ones (e.g., 0.10 vs. 0.44 for Hit@1 in the best configuration). This suggests that edit frequency within a cluster is a stronger signal of future repair activity than semantic embedding proximity, likely because many users repeatedly edit specific subsets of entities, making frequency a more reliable indicator of expertise than global similarity.

Finally, we analyzed user distribution across clusters to understand the recommendation challenge better. For each user, we measured how their historical edits were spread across clusters: if all edits were concentrated in one cluster, the user's distribution was highly focused; if edits were spread evenly across multiple clusters, the distribution was broader. We quantified this spread using entropy, a measure of uncertainty that increases when edits are evenly distributed across clusters and decreases when activity is concentrated in just one or very few clusters. A low entropy value therefore indicates a specialist user focused on a narrow topical area, while a higher value points to a generalist user contributing across multiple clusters.

In our dataset, a large proportion of users (≥70%) contributed edits to only a single cluster, and the mean entropy values remained low (0.43–0.56), confirming that most contributors are specialists. Despite this strong skew, the membership hit rates show that our clustering frequently placed the correct user in the same cluster as the test entity (up to 0.90 for fixed-k methods and 0.82–0.87 for HDBSCAN). This highlights that recommending the right user is a non-trivial task: clusters often contain many potential editors with similar topical focus, making intra-cluster ranking challenging. Nevertheless, our approach often retrieves the correct user among the top candidates, showing the promise of clustering-based strategies for user recommendation in WD repair.

5 Related Works

This work intersects WD quality, repair, and user modeling. On the repair side, Tanon et al. [9] develop a dataset of WD constraint violations paired with their fixes, enabling analyses of historical repairs; we extend this line by linking repairs to users to support recommendation and evaluation at the user level. Xue and Zou [11] review knowledge-graph quality management across assessment, detection, and correction methods, highlighting human-in-the-loop and rule-based repair frameworks that validate system-generated repairs with users rather than recommending users themselves. This gap motivates our focus on recommending suitable users to propose repairs. For semantic representations, Ilievski et al. [4] show that sentence embeddings capture concept similarity in WD more effectively than traditional KG embeddings, motivating our use of abstract-based sentence representations for clustering. Complementary quality frameworks detect low-quality statements via reverts, deprecation, and constraints [8], while Ferranti et al. [1]

Experiment	Algo.	Sil.	# Clusters	Membership	Hit@1,3,10$_{freq}$	Hit@1,3,10$_{cos}$	Users ≤ 1	Entropy
noAbstracts	HDBSCAN	**0.58±0.03**	69.3±3.95	0.82	**0.44**/0.66/0.77	**0.10**/0.17/0.42	**0.76**	0.43
all	HDBSCAN	**0.57±0.04**	113.6±8.81	0.83	**0.46**/0.64/0.76	**0.07**/0.20/0.39	0.70	0.56
onlyAbstracts	HDBSCAN	0.55±0.05	49.8±21.64	0.87	0.41/0.56/0.72	0.06/0.18/0.31	0.72	0.46
onlyAbstracts	KMeans$_{k=10}$	0.49±0.02	10.0	**0.90**	0.35/0.50/0.69	0.04/0.12/0.16	0.68	0.48
onlyAbstracts	GMM$_{k=10}$	0.48±0.02	10.0	**0.90**	0.35/0.50/0.70	0.04/0.12/0.21	0.68	0.48
onlyAbstracts	KMeans$_{k=20}$	0.48±0.02	20.0	0.87	0.37/0.53/0.72	0.04/0.13/0.22	0.66	0.60
onlyAbstracts	GMM$_{k=20}$	0.46±0.02	20.0	0.87	0.36/0.53/0.73	0.04/0.13/0.23	0.66	0.60
onlyAbstracts	GMM$_{k=5}$	0.46±0.04	5.0	**0.92**	0.32/0.49/0.65	0.04/0.12/0.20	0.74	0.27
all	KMeans$_{k=20}$	0.45±0.02	20.0	0.89	0.35/0.54/0.71	0.03/0.09/0.18	0.67	0.54
noAbstracts	KMeans$_{k=10}$	0.45±0.02	10.0	0.89	0.33/0.57/0.74	0.06/0.08/0.17	**0.76**	0.32

Table 1: 10-fold results: silhouette, membership, recommendation (Hit@k), and user distribution.

formalize WD's property constraints with SHACL/SPARQL, emphasizing validation rather than user assignment.

On the user side, prior work characterizes roles and editing dynamics in WD without addressing user recommendation [5, 6]; studies on bots in collaborative projects underline their substantial footprint in maintenance workflows, which we treat uniformly with humans in our modeling [12]. More broadly, surveys of evolving/open KGs synthesize dimensions of evolution, observability, and collaboration (including the roles of registered users and bots), offering useful context for our clustering-based view of topical regions and editor behavior [7]. We align with this by leveraging edit histories and semantically coherent clusters for downstream recommendation, rather than analyzing evolution per se.

To our knowledge, no prior work combines semantic clustering of entities with historical repair activity to recommend users for new inconsistencies in WD, while evaluating frequency- and embedding-based strategies under 10-fold cross-validation.

6 Conclusions

This paper introduced a lightweight, unsupervised framework for recommending WD users likely to repair newly detected inconsistencies, based on semantic clustering of historically repaired entities. By combining sentence-level embeddings, UMAP dimensionality reduction, and clustering, we linked past repairs to the users who performed them and explored frequency- and similarity-based recommendation strategies.

Our evaluation on the dataset of one-of constraint repairs by Tanon et al. [9] demonstrated that HDBSCAN with UMAP produced the best clustering quality, achieving a silhouette score of up to 0.58, high membership hit rates (0.82–0.87), and Hits@3 of 60% using frequency-based ranking. These results indicate that topic-based clusters can meaningfully group users with relevant expertise, providing a strong basis for recommending editors for new constraint violations.

While our study focused on one-of constraint repairs, we envision extensions of this framework addressing a broader range of constraint types (as in [9]), allowing for more comprehensive coverage of quality issues. Furthermore, the repaired property itself could be leveraged as an additional dimension of user expertise, capturing the users' "taxonomical" knowledge. A temporal dimension could also be introduced, considering recent edits, as user interests and activity patterns may evolve. Finally, a systematic comparison of text-based embeddings versus our fallback structural descriptions on the same dataset would shed light on which approach leads to more coherent clusters and more accurate recommendations.

Overall, our results suggest that embedding-based clustering provides a promising foundation for semi-automated user recommendation in knowledge graph maintenance, opening opportunities for richer user modeling and hybrid approaches that combine semantic, structural, and temporal data for user modeling.

Acknowledgments. This work was funded by the Austrian Science Fund (FWF) [10.55776/COE12].

References

[1] N. Ferranti, J.F. De Souza, S. Ahmetaj, and A. Polleres. 2024. Formalizing and validating Wikidata's property constraints using SHACL and SPARQL. *Semantic Web* 15, 6 (2024), 2333–2380. doi:10.3233/SW-243611

[2] J.E. Labra Gayo. 2022. WShEx: A language to describe and validate Wikibase entities. In *3rd Wikidata Workshop 2022 @ ISWC2022, Hanghzou, China, October 2022*, Vol. 3262. CEUR-WS.org. https://ceur-ws.org/Vol-3262/paper3.pdf

[3] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, et al. 2022. Knowledge Graphs. *ACM Comput. Surv.* 54, 4 (2022), 71:1–71:37. doi:10.1145/3447772

[4] F. Ilievski, K. Shenoy, H. Chalupsky, et al. 2024. A study of concept similarity in Wikidata. *Semantic Web* 15, 3 (2024), 877–896. doi:10.3233/SW-233520

[5] A. Piscopo and E. Simperl. 2018. Who Models the World?: Collaborative Ontology Creation and User Roles in Wikidata. *Proc. ACM Hum. Comput. Interact.* 2, CSCW (2018), 141:1–141:18. doi:10.1145/3274410

[6] A. Piscopo and E. Simperl. 2019. What we talk about when we talk about wikidata quality: a literature survey. In *Proceedings of the 15th International Symposium on Open Collaboration, OpenSym 2019, Skövde, Sweden, August 20-22, 2019*. ACM, 17:1–17:11. doi:10.1145/3306446.3340822

[7] A. Polleres, R. Pernisch, A. Bonifati, D. Dell'Aglio, D. Dobriy, S. Dumbrava, L. Etcheverry, N. Ferranti, et al. 2023. How does knowledge evolve in open knowledge graphs? *TGDK* 1, 1 (2023), 11–1. doi:10.4230/TGDK.1.1.11

[8] K. Shenoy, F. Ilievski, D. Garijo, et al. 2022. A study of the quality of Wikidata. *J. Web Semant.* 72 (2022), 100679. doi:10.1016/J.WEBSEM.2021.100679

[9] T.P. Tanon, C. Bourgaux, and F. Suchanek. 2019. Learning how to correct a knowledge base from the edit history. In *The World Wide Web Conference*. 1465–1475.

[10] H. Turki, D. Jemielniak, M.A.H Taieb, J.E. Labra Gayo, et al. 2022. Using logical constraints to validate statistical information about disease outbreaks in collaborative knowledge graphs: the case of COVID-19 epidemiology in Wikidata. *PeerJ Computer Science* 8 (2022), e1085. doi:10.7717/peerj-cs.1085

[11] B. Xue and L. Zou. 2023. Knowledge graph quality management: A comprehensive survey. *TGDK* 35, 5 (2023), 4969–4988. doi:10.1109/TKDE.2022.3150080

[12] L. Zheng, C.M. Albano, N.M. Vora, F. Mai, and J.V. Nickerson. 2019. The Roles Bots Play in Wikipedia. *Proc. ACM Hum. Comput. Interact.* 3, CSCW (2019), 215:1–215:20. doi:10.1145/3359317

Chapter 4

Tracing Evolving Knowledge in Wikidata through Statement Lifecycles

Under Review.

May 2026

Tracing Evolving Knowledge in Wikidata through Statement Lifecycles

Blind REVIEW [a]

Abstract. Wikidata is a large, collaboratively maintained knowledge graph whose content and schema evolve continuously through additions, deletions, and revisions of truthy, i.e., active, statements over time, including property-graph style context information (qualifiers) about these statements. Although the complete change history is publicly available as part of Wikidata's full meta-history dumps, due to its sheer scale, growth rate, and structure, these dumps are not directly amenable to temporal knowledge graph analysis and querying. We present a pipeline for reconstructing (truthy) Wikidata statement *lifecycles* from the full historical dump. The pipeline extracts statement-level records with creation and deletion provenance, reconstructs truthy triples from statement ranks, and derives analysis-ready artifacts for studying knowledge graph maintenance over time. We demonstrate the use of this machinery to investigate specialization in Wikidata editing behavior across three complementary levels of increasing complexity: (i) property-centered activity, (ii) class- (or more generally category-specific) activity, and finally (iii) activity related to constraint-maintenance. The resulting analyses show how statement lifecycle analysis can support empirical studies that are neither possible on standard query endpoints, nor on static dumps alone. Our statement-lifecycle-based "lens" shall also enable broader uses such as historical graph slicing, temporally grounded link prediction evaluation, and time-aware selection of schema information for repair models.

Keywords. Wikidata and Knowledge Graph Evolution and Statement Lifecycles and Historical Analysis and Knowledge Graph Maintenance

1. Introduction

Wikidata [1] is one of the most prominent collaboratively maintained knowledge graphs (KGs) [2]. Since its launch by the Wikimedia Foundation, it has become a central open source of structured knowledge. Its content is created and maintained by a heterogeneous community of registered users, anonymous contributors, bots, and tool-assisted editing workflows. Unlike KGs that are built mainly through centralized extraction pipelines [3], Wikidata enables open collaboration on the data model level itself, wherein items, statements, qualifiers, references, schema (properties and classes) and constraints can all evolve over time in parallel.

Wikidata is built on Wikibase, a software ecosystem that is increasingly used beyond Wikidata itself. This makes Wikidata not only an important KG in its own right, but also a large-scale case study for understanding how Wikibase-based KGs evolve. In particular, Wikidata's property system and its constraint mechanism provide a collaboratively maintained schema and quality-control layer. Apart from (i) factual statements about entities, such as USA (Q36) has capital (P36) Washington D.C. (Q61), Wikidata

also includes (ii) property (subPropertyOf (P1647)) and class (subclassOf (P279)) hierarchy statements[1], plus (iii) property constraint (P2302) statements [5], that are used to express expected usage patterns for properties. Changes to statements affect how the graph is validated, interpreted, and maintained, where obviously parallel changes on all these three levels (facts, schema, constraints) interact with each other.

Both changes in the class hierarchy (P279), or constraint definitions might affect whether constraints can be validated or not: for instance, the capital (P36) property is constrained by a subject type constraint (Q21503250), which in turn restricts the subjects to instances of (P31) particular classes, including region (Q82794). As illustrated in the graph in Figure 1 (to be explained in more detail below in Section 2) this is fulfilled for the capital of the USA, through the subclass (P279) hierarchy. Yet, the validity may change by either changing the instance statement, the class hierarchy, or the constraint definition itself. In a collaborative KG, where all such changes can happen in parallel, by edits of different users, it is therefore important to track such user changes in order to analyse and hopefully support convergence and coherence at the scale of Wikidata.

Studying the evolution of Wikidata statements at all three levels is thus relevant for several reasons. It can reveal how collaborative communities organize maintenance work, how different semantic "subareas" within the graph change over time, and how schema-level structures such as classes, properties, and constraints are curated [6]. More broadly, methods developed for Wikidata can inform the analysis of other Wikibase KGs, or, more generally evolving KGs, especially when these graphs rely on similar mechanisms for properties, constraints, and community-driven editing.

However, as the literature shows, historical analysis of Wikidata remains difficult. Static dumps expose the state of the graph at particular points in time, while the raw edit history is organized around page revisions, not explicitly expressing statement lifecycles. Systems for querying historical graph states have been explored [7], but providing scalable access to the full history of a graph of Wikidata's size remains technically challenging, especially as Wikidata grows further. As a result, many empirical studies either focus on (current) snapshots [8,5], restrict themselves to samples [9], or require substantial custom preprocessing before temporal questions can be asked. Recent work has begun to construct temporal KG resources from Wikipedia-derived data [10]. However, comparable statement-lifecycle machinery for Wikidata itself, that also relates changes to the evolution of the editors' community, remains limited, despite Wikidata's unique role within open KGs.

In this paper, we address this gap by introducing a practical machinery for extracting statement-level history from Wikidata and transforming it into temporal artifacts on all three mentioned levels, that can support downstream, user-community related analyses: statement lifecycles, (schema) hierarchy lifecycles, and constraint lifecycles. We use this machinery to study semantic specialization in Wikidata editing behavior. Rather than merely quantifying users' edits, we ask whether their activity concentrates around particular semantic dimensions of the graph. We consider three complementary views: property-centered activity, class-context activity based on temporally valid instance of (P31) statements, and constraint-maintenance activity involving property constraint (P2302) statements. We demonstrate that a bespoke statement lifecycle extraction and aggregation pipeline can support all these views.

[1] similar in spirit to RDFS/OWL, cf. [4]

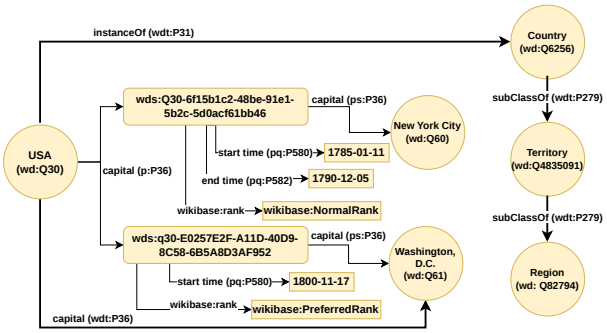


Figure 1. Simplified Wikidata RDF representation of two statements for capital (P36) of the United States, adapted from [5]. Full statements are represented through statement nodes in the `wds:` namespace, connected through `p:` and `ps:` predicates and enriched with qualifiers such as `start time` (P580), `end time` (P582), and a `wikibase:rank`. Direct claim `wdt:` triples expose truthy values, here the preferred-rank statement for Washington, D.C., as well as the class membership (P31) of the US, plus the (relevant part of the) class hierarchy (P279).

Our experimental results from these analyses serve as a demonstration of the broader value of such historical statement lifecycle extraction. Beyond the study of editors' behaviors, the same machinery can support temporal graph slicing, the selection of time-appropriate schema information, and the construction of evaluation settings in which graph development predictions are compared against statements that are actually added later in Wikidata's history.

Overall, the contributions of this paper are as follows: **(1)** We introduce the concept of statement lifecycles and extraction machinery for Wikidata edit history, enabling temporal analyses without a full historical query service. All materials are available at [2]; **(2)** We derive temporal artifacts for three semantic dimensions of Wikidata maintenance: properties, classes, and property constraints; **(3)** We empirically analyse semantic specialization in Wikidata editing behavior, using properties, temporally valid (direct) class memberships, and property constraints as complementary semantic views of maintenance activity; **(4)** We discuss how the same machinery can be reused for broader temporal KG tasks, including temporal link prediction evaluation and time-aware input selection for repair models.

The remainder of this paper is organized as follows. Section 2 reviews Wikidata-related works and concepts, Section 3 defines the lifecycle concepts, and Section 4 describes the historical data and extraction pipeline. Section 5 presents the semantic specialization analyses, followed by broader uses of the machinery (Section 6) with Section 7 concluding and pointing future directions.

2. Background and Related Work

Wikidata is a collaboratively edited KG built on Wikibase. Its basic entities are items, identified by Q-identifiers, and properties, identified by P-identifiers. Statements connect

[2] `https://github.com/nicolasferranti/wikidata_statements_lifecycle_machine/`

such entities to a value through a property, and may be enriched with qualifiers, references, and ranks. Figure 1 illustrates this model for the property capital (P36). The full RDF representation uses reified "statement nodes", where Wikidata's reification model relies on separate, bespoke namespaces: the subject is linked to a statement node through the `p:` predicate, the statement node points to the value through a `ps:` predicate, and additional information can be attached through qualifiers `pq:`, such as start time (P580), end time (P582). Finally, statement ranks (`wikibase:rank`) are attached to the statement node. Wikidata RDF dumps also provide a so-called "truthy" direct claims view using `wdt:` predicates. These truthy `wdt:` triples denote a graph of the current best-ranked statement values, commonly used in downstream queries and analyses.

Yet, this RDF representation is in several respects incomplete with regard to information necessary for analyses regarding historical editing: both (i) *transaction time*[3] of when statements were edited, changed or removed, as well as (ii) provenance, i.e. who made the respective changes is missing.

Unlike KGs organized around a predefined formal ontology, Wikidata's schema-like structures emerge from community editing. Relations such as instance of (P31), subclass of (P279), and subproperty of (P1647) are commonly used to approximate its evolving terminology layer [11]. This makes Wikidata an important case for studying community-driven ontology engineering, but also a difficult one: both factual statements and their organizational structures change over time. Baroncini et al. [12] build on this view by proposing a methodology for analysing the evolution of sub-schemas within Wikidata, focusing on how domain-specific communities use and develop classes and properties.

The historical dimension of Wikidata has also motivated work on temporal access to graph states. Pellissier Tanon and Suchanek [7] proposed a historical query infrastructure exposing both revision-level diffs and graph states after revisions, demonstrating the value of historical querying while also highlighting the difficulty of supporting it at Wikidata scale; indeed, the historic endpoint provided in the paper is no longer maintained. Generally, given the scale issues and timeouts that even the current Wikidata endpoints or replications thereof are facing [13], the provision of public SPARQL endpoints as a general tool for such more complicated analyses does not seem to be the most promising route to take, even with new engines and benchmarks suggesting progress towards more scalable solutions for specific types of queries [14].

More broadly, work on evolving KGs distinguishes between dynamic, versioned, and temporal KGs, and emphasizes the challenges of observing, storing, querying, and analysing change in large open KGs [6]. Recent dataset work has taken a complementary route: Hofer et al. [10] construct a temporal extension of DBpedia by processing Wikipedia meta-history dumps and recording triple lifespans as transaction-time intervals. Our work follows this motivation, but targets Wikidata directly, and extends the idea of lifespan-annotation of triples with additional edit provenance annotations. Instead of extracting DBpedia triples from Wikipedia pages or providing a general historical query endpoint, we reconstruct the *lifecycles* (lifespan+edit provenance) of Wikidata statements and derive task-specific artifacts for an evolutionary analysis.

A second line of work studies the people and communities behind Wikidata. Piscopo and Simperl [11] analysed user roles and ontology quality over time, identifying broad roles such as contributors and leaders. Koutsiana et al. [15] complement this perspective

[3] while statement *valid* time is possible to be added through qualifiers, as shown in the example of Figure 1.

by studying communication patterns in Wikidata discussions and their relation to community engagement. Both these works show that Wikidata maintenance is socially structured, but do not yet enable a general machinery for relating specific schema-related specializations to users or communities. Our paper does exactly that and provides the means to analyse whether the social editing structure is also visible semantically: whether editors concentrate their activity around particular properties, class-contexts, or constraint types, on a granular level.

Property constraints provide a further schema-like layer of Wikidata. They are represented through property constraint (P2302) statements and are used to express expectations about how properties should be used. Ferranti et al. [5] formalize Wikidata's property constraints using SHACL and SPARQL, showing both their importance for quality control and the difficulty of interpreting and validating them at scale. Recent work on user recommendation for KG repair similarly treats past user activity as evidence of expertise in particular regions of the graph [16]. Our work is complementary: rather than validating/repairing constraints or recommending users, we reconstruct statement lifecycles and use them to analyse how Wikidata maintenance is distributed over time.

3. Statement, Type, (Class-)Hierarchy and Constraint Lifecycles in Evolving KGs

We denote as "statement lifecycle" the combined lifespan+editor-annotated history of statements within an evolving, collaboratively edited KG, in the sense of an arbitrary property graph. Without loss of generality, and abstracting from Wikidata's concrete RDF representation, we herein use MAPL [17,18] style notation that compactly represents qualified statements $s@Q$, where $s = (entity, property, value)$ is a statement, along with its respective context set Q (i.e., attribute-value pairs, including the statement-related qualifiers, references and ranks).

Drawing from the ideas of Annotated RDF [19,20], the full Wikidata edit history can be represented compactly as a sequence of *statement lifecycles* of the form $sl = s@Q[I, u_c, u_d]$, which comprise a qualified statement $s@Q$ annotated with a validity interval $I = [t_c, t_d)$, as well as the users u_c, u_d who created or, resp. deleted the statement. As a shortcut, when the same user created and deleted $s@Q$, we compactly write $s@Q[I,u]$ instead of $s@Q[I,u,u]$.

The overall history of an evolving KG can then be understood as $SL = \langle sl_1, ..., sl_n \rangle$ where for any $sl_i = s@Q_i[I_i, \ldots]$, $sl_j = s@Q_j[I_j, \ldots]$ referring to identical statements, the lifespan intervals I_i, I_j are non-overlapping. While we therefore could consider SL as a set, mainly for ease of processing,[4] we may also assume SL to be a *sequence ordered* by entities and starting time, i.e. by $entity, t_c$. For illustration, this means we can view SL_{WD} for the case of Wikidata as a sequence:

[4] Also, this resembles the order in the actual WD history dumps, for details cf. Section 4.

$$SL_{WD} = \langle\ (\text{wd:Q1}, p_{1,1}, v_{1,1})@Q_{1,1}[I_{1,1}, u_{c,1,1}, u_{d,1,1}]$$
$$(\text{wd:Q1}, p_{1,2}, v_{1,2})@Q_{1,2}[I_{1,2}, u_{c,1,2}, u_{d,1,2}], \ldots,$$
$$(\text{wd:Q1}, p_{1,m_1}, v_{1,m_1})@Q_{1,m_1}[I_{1,m_1}, u_{c,1,m_1}, u_{d,1,m_1}],$$
$$(\text{wd:Q2}, p_{2,1}, v_{2,1})@Q_{2,1}[I_{2,1}, u_{c,2,1}, u_{d,2,1}], \ldots,$$
$$(\text{wd:Qn}, p_{n,m_n}, v_{n,m})@Q_{n,m_n}[I_{n,m_n}, u_{c,n,m_n}, u_{d,n,m_n}]]\rangle$$

For convenience of analyses, we define further operators on statement lifecycle sequences, essentially inspired by traditional relational operators. Firstly, we use the *selection operator* $TL = \sigma_{p=\text{type}}(SL)$ for filtering SL by the type (instance-of) predicate: for Wikidata, we assume type=wdt:P31 (instanceOf), but for other KGs, using e.g. rdf:type for typing, this could be modified straightforwardly.

Next, we introduce a "lifecycle join" operator $SL \bowtie^L_{condition,sq_r} SL'$ which we define tuplewise as

$$(e,p,v)@Q[I,u_c,u_d] \bowtie^L_{condition,sq_r} (e,p',v')@Q'[I',u'_s,u'_e] = sq_r[I \cap I', u_c, u_d] \quad (1)$$

if the *condition* applies, and $\emptyset$ otherwise. Here sq_r is the "output" qualified statement.

We can then use this to derive the class-edits lifecycle CL, as follows:

$$CL = SL \bowtie^L_{e=e' \wedge t_c > t'_c, (e',p',v')@Q} TL$$

CL will produce statement lifecycles of the form $(e, \text{wdt:P31}, C)[I, u_c, u_d]$ for each original statement edit on e in SL; $t_c > t'_c$ in the condition ensures that only those edits are counted, when e already was a member of C. We note that the base assumption of non-overlapping statements in SL no longer holds after applying $\bowtie^L$. In fact, we have defined CL such that overlaps naturally occur by various entities sharing the same class memberships in parallel. Yet, this operator is very convenient for aggregation, along the evolution analysis use cases we have in mind, for instance counting edits per class instances.

3.1. Counting user edits per class:

For instance, if we count tuples in CL grouping by user u_c, we can conveniently count the edits on instances of a particular class, i.e., slightly abusing SQL syntax, the result of

```
SELECT u_c, C, COUNT(*) FROM CL
```

can provide a count of user edits per class, that is, all edits on instances of particular classes; it is important to note here that by design of $\bowtie_L$ only the overlapping edits during class membership shall be counted. As we will see in the next section, rather than using SQL, this can be computed conveniently, even for a large KG like Wikidata, by exploiting the sort order of SL and indexing CL.

3.2. Counting user edits per class incl. the class hierarchy:

We note that the previous method can only account for user edits per explicitly declared instances (via type, or resp. P31 statements), as the subclass hierarchy, which however is also subject to potential changes, is not taken into account. To this end, we can conveniently leverage the above-defined operators.

Let $HL = \sigma_{p=\mathsf{subclassOf}}(SL)$, where for Wikidata, we assume subclassOf=wdt:P279. The computation of all edits per class can be done, theoretically in two steps: **(1)** compute (by exhaustive self-joins, which we do not detail here) the temporal transitive closure over HL in overlapping intervals, which we call HL^*; and **(2)** compute $CL^* = CL \cup CL \bowtie^L_{C=entity' \wedge t_c \geq t'_c, (entity,p,C')} HL^*$.

$CL*$ results in all the transitive $(e, \mathsf{wdt{:}P31}, C)[I, u_c, u_d]$ statements per user edit in SL, respecting the type hierarchy. In practice this can be further optimised, but aggregating counts per user in CL first, and instead of materialising CL^*, combining step 2 into aggregating these partial counts over HL^*, again exploiting sort order and hashing for HL^*.

3.3. Computing constraint lifecycles:

As per [8], property constraint definitions in Wikidata are also statements of the form (e,wdt:P2302,v)@Q such that their lifecycles can be tracked in two steps. First, we identify the historical constraint *anchors* $C_A = \sigma_{p=\mathsf{p{:}P2302}}(SL)$, where each triple links a Wikidata property to a constraint statement node. This selection alone does not contain the full constraint definition. We therefore extend it with the statements in SL attached to each constraint node, i.e.:

$$CoL = C_A \cup C_D, \textit{ where } C_D = C_A \bowtie^L_{C_A.v=SL.e,\ (SL.e,SL.p,SL.v)@Q_{SL}} SL$$

The resulting artifact, which we call constraint lifecycles (CoL) contains both the declaration of a constraint on a property, $(e, \mathsf{p{:}P2302}, v)$, and the associated constraint-definition triples (i.e. constraint qualifiers). It should be intuitive to see that CoL can – in similar ways as above – be joined again with actual factual edits in the KGs to check temporal validity, violations, but also historical repairs, exploiting similar temporal joins, as we have abstractly introduced in the prior subsections.

Yet, before we go into further detail on possible analyses on the constraint level, it makes sense though, to dive a bit deeper into the actual construction of SL from Wikidata's historical dump. In fact, we have abstracted away so far from various details, such as the actual provision of these log files, or the fact that these dumps do not actually contain the history of truthy statements directly. In the next section we will thus need to go to the "lower decks" of the machinery we use (i) to process log dumps in order to produce abstractions and (ii) optimize the computation of aggregations and per-user counts such as the ones introduced above.

4. Historical Data and Extraction Pipeline

Figure 2 shows the pipeline used to instantiate the lifecycle machinery introduced in Section 3. The Wikidata page meta-history dump is organised in files that record successive

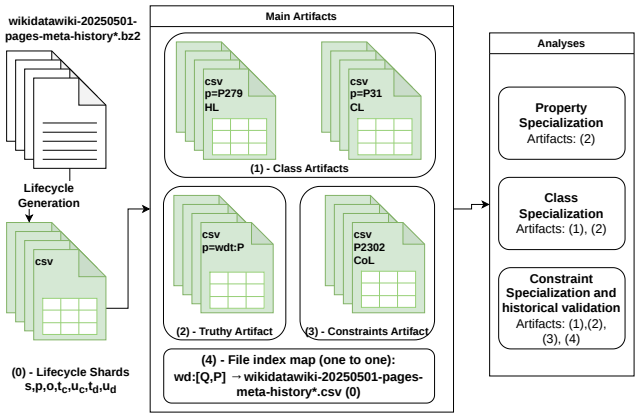


Figure 2. Overview of the historical extraction pipeline. Starting from the full Wikidata meta-history dump files, the lifecycle generation produces statement lifecycle "shards" in separate CSV files. These are then used to derive task-specific artifacts.

entity revisions (i.e. edit events on statements, ordered by $(entity, revisiontimestamp)$) rather than statement lifecycles. We therefore first reconstruct partial statement lifecycle "shards" by comparing consecutive revisions and recording for each observed statement per file, considering subject, predicate, object, creation timestamp and user (editor), and, when applicable, deletion timestamp and user. The resulting CSV *lifecycle shards* use the schema of *SL*, i.e.,$\mathtt{s}$, $\mathtt{p}$, $\mathtt{o}$, $\mathtt{t_c}$, $\mathtt{u_c}$, $\mathtt{t_d}$, and $\mathtt{u_d}$.

From the lifecycle shards per history dump file we materialize other main artifacts used in our experiments. The *truthy artifact* (direct claims) corresponds to $\sigma_{\text{prefix}(p)=\mathtt{wdt:}}(SL)$. Notably, `wdt:` statements are not directly stored in the meta-history dumps, but need to be recreated from scanning ranked statement changes in the entity revisions, following Wikidata's truthy semantics: preferred-rank statements are selected when present, otherwise normal-rank statements are considered; thus, also re-ranking in the revision history may result in "recording" creations/deletions of a wdt: statement within the resulting truthy artifact.

The *class artifacts* contain direct historical instances, $TL = \sigma_{p=\mathtt{wdt:P31}}(SL)$, and extracted subclass statements, $HL = \sigma_{p=\mathtt{wdt:P279}}(SL)$, with the P31 artifact being the main artifact used in the class-specialization analysis. The constraint artifact materializes *CoL* and is used to study both constraint specialization and historical validation against a dataset of repaired statements.

Finally, since we keep separate files for the other artifacts per the original history dump "shards", we build a useful entity-to-file index mapping Wikidata entity and property identifiers `wd:[Q,P]` to the lifecycle CSV shards where they occur. This index is used during historical constraint validation to retrieve the relevant lifecycle records efficiently, without scanning all generated shards.

5. Semantic Specialization Analyses

In order to demonstrate the utility of the artifacts defined in Section 4 we present downstream tasks related to *semantic specialization*, i.e., editing activity concentrated in a se-

mantically identifiable "regions" of Wikidata, such as (i) (sets of) properties, (ii) a class context, or (iii) a family of property constraints: property specialization uses the truthy direct-claim artifact; class specialization combines the truthy artifact with the P31 artifact (to directly produce aggregAted counts instead of materializing *CL*); constraint specialization and validation use the truthy artifact, *CoL*, as well as the entity-to-file index.

More concretely, our analyses address three orthogonal research questions. **RQ1:** To what extent do editors specialize in individual properties (or small portfolios of related properties)? **RQ2:** To what extent do editors specialize in items that belonged to particular class contexts at the time of editing? **RQ3:** How is property-constraint work distributed across users, and are reported constraint-based repairs historically consistent with the constraints valid at the time?

5.1. Property-Centered Specialization

RQ1 uses an editor–property matrix built from the truthy artifact (c.f. Section 4) to answer the question whether Wikidata editors specialize in particular properties, either as single-property specialists or as maintainers of small property portfolios. From the truthy artifact, we construct an editor–property matrix in which each row records the number of edit events by user u involving property p. To reduce noise, we exclude anonymous IP editors, users identified in our bot list, properties edited by too few distinct users, users with too few total edits, and users whose activity involves only a single property. After filtering, the analysis contains 104.4K users (12.2% of the original), 9.7K properties (75.9%), 2.6M user-property rows (60.0%), and 623.6M edits (37.7%).

We finally identify property specialization through a dominance criterion. A user u is said to dominate a property p when three conditions hold: (i) p has at least 50 edits globally; (ii) u contributed at least 75% of all edits involving p; and (iii) edits involving p account for at least 5% of u's own activity.

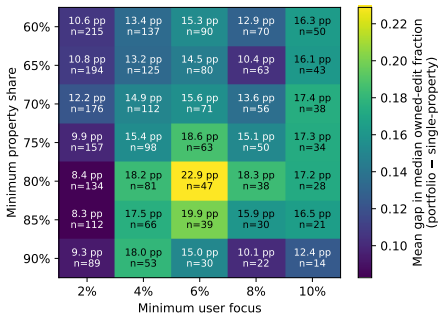
For a dominant user–property pair, we use four derived quantities. The *user–property edit count* is the number of edits made by user u involving property p. The *property share* is the fraction of all edits involving p that were made by u. The *user focus* is the fraction of u's edits that involve p. Finally, for each specialist, the *dominant-edit fraction* is the fraction of the user's edits that fall on all properties dominated by that user.

As for the last metric we note the following observation: while our definition of property specialization does primarily focus on *single* property specialisations – and a more detailed considerations of "vocabulary specialization" (i.e. *groups* of properties that co-occur in a user's edits) remains on our agenda – interestingly we also found users that dominate two or more properties, even under this strict definition. We call such users "portfolio specialists" and can observe interesting patterns in the identified portfolios.

The dominance criterion identifies 528 dominant user–property pairs, involving 412 distinct editors. Of these editors, 333 dominate exactly one property, while 79 dominate two or more properties ("portfolio specialists"). Figure 3(a) summarizes the two groups. While portfolio specialists are fewer, their dominated properties account for a larger share of their editing activity and for a much larger number of edits in absolute terms: the median single-property specialist has 4,059 dominant edits, and the dominated property accounts for 17.8% of their activity. By contrast, the median portfolio specialist has 37,787 dominant edits, and the dominated properties account for 34.7% of their activity. In both groups, the median property share is above 91%, meaning that the dominated properties are strongly concentrated around a corresponding user.

Group	Users	Dom. frac.	Dom. edits	Prop. share
Single-property	333	0.178	4,059	0.914
Portfolio	79	0.347	37,787	0.911

(a)



(b)

Figure 3. Property-centered specialization summary and threshold sensitivity. **(a)** Summary of specialist groups; *Dom. frac.* is the median share of a user's edits on dominated properties, *Dom. edits* is the median number of such edits, and *Prop. share* is the median user share of the total edit volume of the dominated properties. **(b)** Threshold sensitivity; rows vary minimum property share and columns vary minimum user focus, with cells averaged over count thresholds. Darker cells indicate small portfolio–single-property gaps, and *n* reports the mean number of portfolio specialists.

To test robustness of our notion of dominance/specialization, we repeat the analysis over a grid of alternative values for the four above-mentioned thresholds: Figure 3(b) shows that the robustness result is not driven by a single threshold choice. Across the tested threshold space, portfolio specialists consistently have a higher median dominant-edit fraction than single-property specialists, with gaps ranging from about 8 to 23 percentage points. As expected, stricter share thresholds reduce the number of detected portfolio specialists, but they do not remove the overall pattern.

A qualitative inspection of the highest-volume portfolios suggests that many portfolios are semantically coherent rather than arbitrary collections of properties. Among the 79 portfolio specialists, most dominate two properties, but the largest portfolios contain up to eight dominated properties. One eight-property portfolio concentrates on astronomical metadata, including catalog code (P528), apparent magnitude (P1215), SIMBAD ID (P3083), right ascension (P6257), declination (P6258), epoch (P6259), and constellation (P59). Another eight-property portfolio contains identifiers for successive editions of the Dictionnaire del'Académie française (such as P11284 and P11291). A six-property portfolio focuses on anime and manga external identifiers, including aniSearch anime ID (P12477), Kitsu anime ID (P11495), notify.moe anime ID (P12427), Anime-Planet anime ID (P11349), MangaUpdates manga ID (P11149), and livechart.me anime ID (P12489). Other coherent portfolios include clinical-trial metadata, ice-hockey career statistics, CJK character metadata, macroeconomic indicators, film and television identifiers, video-game identifiers, coin metadata, architectural identifiers, and chemical or mineral identifiers. In the inspected high-volume portfolios, external identifiers are especially visible: 30 of the 56 labeled properties are identifier properties.

Overall, our results provide evidence for property-centered specialization in Wikidata maintenance: the strongest pattern is not simply that some users edit a single property frequently, but that some users account for a large share of highly specialised activity on particular properties while also organizing a substantial part of their own activity around them. The distinction between single-property and portfolio specialists is important: single-property specialists may reflect narrow maintenance tasks, whereas portfo-

lio specialists suggest broader maintenance roles around related sets of properties, for instance, domain-wise related sets of (external) identifiers.

5.2. Class-Context Specialization

The second analysis asks whether editors specialize in items that belonged to particular class contexts at the time of editing. We aggregate the class-edits lifecycle CL from Section 3: an edit on an item is counted for each direct P31 class that was active when the edit occurred. We deliberately do not materialize the subclass hierarchy HL^* or CL^* here, since the goal is to capture the explicit class contexts in which users edited (rather than the ones implicit by subclassing); while we deem counting classes including transitive subclass edits potentially interesting for future work, we will show another example of using HL^* for constraint-checking in Section 5.3 below.

For analyzing class edits, we again remove anonymous IP editors and users identified as bots, excluding 650,156 of 820,341 users (79.3%) and 1,410,783 of 4,992,300 user–class rows (28.3%). The cleaned table contains 170,185 users and 3,581,517 user–class rows. We retain users with at least 50 class-context edits across at least two classes, yielding 29,058 multi-class users and avoiding degenerate single-class focus cases.

For each user u and class c, let $n_{u,c}$ be the number of edits assigned to c through CL, $N_u = \sum_c n_{u,c}$, and $p_{u,c} = n_{u,c}/N_u$. We measure concentration with the top-class share $\max_c p_{u,c}$ and with normalized entropy $H(u)/\log_2 k_u$, where $H(u) = -\sum_c p_{u,c} \log_2 p_{u,c}$ [21] and k_u is the number of active classes.

Concentration alone does not show whether a user's class profile is distinctive. We therefore compare p_u with an instance-based reference distribution over classes. For each class c, the reference mass m_c is the total number of times items were added as instances of c in CL, regardless of whether those memberships were later deleted. This reference distribution captures class prevalence in TL, not edit volume on items of that class. This matters because edit volume would make maintenance-intensive classes appear common simply because they receive many revisions.

We smooth these reference masses with $\alpha = 0.5$, defining $q_c = (m_c + \alpha)/(\sum_j m_j + \alpha K)$, where K is the number of classes in the combined user and reference universe. We then compute $D_{\mathrm{KL}}(p_u \parallel q) = \sum_c p_{u,c} \log_2(p_{u,c}/q_c)$ [22]. A high KL means that some classes account for a larger share of the user's edits than they account for in the historical CL reference distribution. Thus, top-class share and entropy measure how focused the user is, while KL divergence measures whether that focus is unusual relative to class prevalence in TL. For robustness, we compare this instance-based reference distribution with an edit-count reference and convex mixtures of the two, varying $\alpha \in \{0.1, 0.5, 1.0\}$ and the minimum-edit threshold. The main specification uses the instance-based reference, $\alpha = 0.5$, a minimum of 50 edits, and the multi-class user filter above.

Under the main specification, Table 1 shows the five users with the highest smoothed KL divergence. The ranking captures baseline-relative distinctiveness rather than only single-class concentration. Mrnmnl and Masquernya are narrow specialists, assigning 99.2% and 96.9% of their class-context edits, respectively, to a single top class. By contrast, Laurenownerscounsel has the highest KL score despite a top-class share of only 35.1%, with edits spread across three related legal classes. Jose Félix Diaz and Priyasekar1 likewise show broader but still distinctive profiles, with nine and six active classes, respectively. Thus, the top-ranked users illustrate both narrow class concentration and broader domain-context distinctiveness.

Table 1. Top five users by smoothed KL divergence under the main class-context specification. KL is $D_{\mathrm{KL}}(p_u \,\|\, q)$, comparing the user's observed class-context edit distribution p_u with the instance-based reference distribution q. Edits is the user's total number of class-context edits after filtering. Classes is the number of distinct direct P31 classes edited by the user. Top share is the fraction of the user's class-context edits assigned to their most-edited class. Top class reports that most-edited direct P31 class, with its Wikidata identifier.

User	KL	Edits	Classes	Top share	Top class
Laurenownerscounsel	23.67	57	3	0.351	eminent domain (wd:Q166332)
Masquernya	23.24	64	2	0.969	wd:Q11511388
Mrnmnl	23.12	119	2	0.992	UML tool (wd:Q2467310)
Jose Félix Diaz	22.62	93	9	0.204	Christian (wd:Q16865575)
Priyasekar1	22.34	82	6	0.427	software (wd:Q445019)

Across all 29,058 eligible users, we combine concentration and breadth into descriptive profile categories. Clear narrow specialists have top-class share at or above the empirical 90th percentile and at most three active classes, while structured multi-class users have top-class share between the median and the 90th percentile and four to fifteen active classes. Under the main specification, these groups contain 1,224 users (4.2%) and 5,282 users (18.2%), respectively, together accounting for 6,506 users (22.4%). A further 12,315 users (42.4%) are broad users with at least sixteen active classes and top-class share below 0.60, while 10,237 users (35.2%) fall outside these definitions.

Robustness checks support the presence of a class-specialization signal but caution against over-interpreting the exact ranking. Instance-based and edit-count reference distributions yield Spearman correlations of approximately 0.987 across $\alpha \in \{0.1, 0.5, 1.0\}$, although only five of the top ten users overlap. We therefore treat Table 1 as illustrative and emphasize the aggregate profile counts under the main specification.

5.3. Constraint-Maintenance

The third analysis uses the constraint lifecycle artifact in two ways. First, we briefly characterize who maintains Wikidata constraints and whether this work is concentrated around particular constraint types or target entities. Second, and more centrally, we test whether reported historical repairs can be validated against the constraint definitions that were actually available when the repaired triples were alive.

For the user-specialization view, we extract human-created, curated constraint additions from *CoL* and build user–constraint-type and user–target-entity matrices, where the target entity is usually the property to which the constraint is attached. As before, we exclude anonymous IP editors and identified bots. For human users, we compute the share of additions belonging to their most frequent constraint type and target entity, and classify users with at least 20 constraint additions using a 70% concentration threshold.

This analysis suggests that sustained human *CoL* statement creation is broad rather than narrowly concentrated. After filtering, the data contains 62,317 human-created, *CoL* statements additions by 835 users, covering 32 constraint types and 9,864 target entities. Among users with at least 20 additions, 150 are classified as generalists, 12 as constraint-type specialists, and only one as a target-entity specialist. Recurring constraint creators therefore usually work across several constraint types and properties rather than focusing on a single constrained entity.

The lifecycle view reinforces the importance of historical reconstruction. Most curated constraints involve limited named-human participation: 70.0% involve exactly one

Table 2. Historical availability of constraint definitions for reported repairs. "Reported id" tests only the constraint id reported by [9]; "any id" tests against all reconstructed historical constraints of the same type and property. The top-property share and pre-constraint count are computed within the no-intersection cases under the any-id setting. Abbreviations: IRS=item-requires-statement, VRS= value-requires-statement.

Type	ID	Repairs	Rep. id no int.	Any id no int.	Top property	Share	Pre-constraint
conflicts-with	Q21502838	496k	179k (36%)	112k (23%)	country (P17)	9%	10.5k
inverse	Q21510855	2.18M	178k (8%)	169k (8%)	category's main topic (P301)	21%	34.8k
IRS	Q21503247	16.2M	628k (4%)	613k (4%)	Ensembl transcript ID (P704)	33%	202k
one-of	Q21510859	15.1k	8.26k (55%)	8.26k (55%)	sex or gender (P21)	62%	5.08k
symmetric	Q21510862	928k	36.8k (4%)	36.0k (4%)	said to be the same as (P460)	43%	15.6k
VRS	Q21510864	15.1M	1.27M (8%)	1.24M (8%)	subclass of (P279)	59%	733k

named human user, 13.5% none, 11.1% two users, and 5.3% involve three or more. However, among recurring lifecycle participants, qualifier maintenance is prominent. This matters because the operational meaning of a constraint is often determined not only by the P2302 statement itself, but also by its ranks and qualifiers over time.

We therefore use the historical constraint lifecycles for a validation experiment. Starting from the raw historical constraint files, we export simplified constraint files for individual constraint types. These contain the property to which the constraint applies, the constraint id, its validity interval, and the testing-relevant qualifiers. Active constraints are selected according to rank, following Wikidata's truthy semantics. The export is constraint-type specific. For simple constraints such as one-of, the validator only needs historical allowed values and exceptions. When multiple qualifier combinations are needed, the export groups the relevant qualifiers into testing units. For example, an item-requires-statement constraint may combine a required property (P2306) with an optional item of property constraint (P2305). If these qualifiers were introduced at different times, the export splits the history into the corresponding validity intervals.

We evaluate this machinery on historical repairs released by [9]. Since historical repairs are used to mine correction rules and train GNN-based repair models [23], accurate temporal validation is important. We restrict the experiment to constraint types whose validation does not require class-hierarchy reasoning or complex qualifier handling such as separators (P4155) [5]. The selected types are shown in Table 2.

The validation pipeline has three steps. First, we normalize the repair triples from [9]. Second, we retrieve the lifecycle of each flagged triple through the entity-to-file index, avoiding a full scan of the historical lifecycle files. Third, we compare the repaired triple lifecycle with the simplified historical constraints. We report two temporal-availability checks: one using only the constraint id reported in the repair dataset, and one testing whether any reconstructed historical constraint of the same type and property intersects with the repaired triple lifecycle.

Table 2 reports the results. The number of reported repairs varies substantially by constraint type, with item-requires-statement and value-requires-statement accounting for most cases. Across all selected constraint types, a non-negligible share of reported repairs has no lifecycle intersection with the relevant historical constraint definition.

The two no-intersection measures are close for most constraint types, but differ more strongly for conflicts-with: 36% of reported repairs do not intersect with the reported constraint id, while 23% do not intersect with any reconstructed constraint of the same type and property. This indicates that some repairs cannot be reproduced with the exact reported constraint id, but can still be temporally grounded by another historical constraint instance of the same type and property.

The pre-constraint counts show that many no-intersection cases concern deletions predating the first reconstructed related constraint, ranging from 5.08k cases for sex or gender (P21) to 733k for subclass of (P279). These cases should therefore not be interpreted simply as errors in the repair dataset. Rather, they identify repairs for which the temporal evidence needed for validation is missing under the historical context.

This distinction matters for downstream repair-learning approaches. If historical repairs are used as supervision while constraint definitions are taken from a later graph state, models may learn from temporally misaligned examples. Historical constraint validation separates repairs grounded in an available constraint definition from repairs that can only be explained retrospectively.

Last, but not least, we mention that the constraint violations in [9], that we compare against here, did not consider type-checking with respect to the P279 class hierarchy (and dynamic changes thereof over time), which are strictly speaking relevant for constraint types such as subject-type constraints (Q21503250). In our supplemental materials, cf. Footnote 2, we provide a description how $HL*$ and the P279 class artifact can be used to check historic constraint validity under consideration of the class hierarchy, illustrated by the example of Figure 1[5].

6. Discussion and Broader Uses

Historical accuracy is crucial for reliable semantic systems, especially those built on temporal, evolutionary, or community-based KGs. In such settings, temporal matching is a sensitive but often overlooked requirement, as discussed in this paper. While our pipeline was evaluated for semantic specialization and historical constraint validation, the reconstructed statement lifecycles enable several additional real-world studies.

First, link prediction approaches are commonly evaluated on benchmarks where existing triples are artificially removed from static subgraphs. A temporally accurate KG instead allows such methods to be tested under a more realistic setting, where new triples naturally emerge through community editing. Second, studies on the evolution of KG embeddings could benefit from selecting temporally valid statements for different training cycles. Although our current lifecycle shards are organized by subject entity rather than chronological edit order, they could be straightforwardly exported into chronological files at a chosen temporal granularity.

More broadly, our approach can support the identification of static, temporal, or disputed facts, along with vandalism detection through user activity analysis. Finally, our use case suggests that machine learning-based repair methods may benefit from more accurate historical information when learning how inconsistencies emerge and how they can be resolved according with the contextual knowledge available at the time.

[5] https://github.com/nicolasferranti/wikidata_statements_lifecycle_machine/tree/main/h3_constraint_editors/h3b_constraints_checking_with_HL

7. Conclusions, Limitations and Future work

We have presented an analysis of semantic specialization in collaborative KG editing, focusing on how users specialize in edits involving properties and classes. Our results indicate the feasibility of such analyses when supported by infrastruture capable of reconstructing statement lifecycles. By tracing how triples are created and edited over time, our approach provides a historically grounded basis for studying user behavior, semantic change, and KG content evolution.

A natural next step is to move from individual to *communities* of users and aggregating on particular sub-vocabularies (groups of co-occurring properties and classes) that communities specialize on. This would make it possible to identify communities that repeatedly contribute to particular domains, modeling patterns, or regions of the ontology. We believe that the lifecycle-based infrastructure introduced in this paper provides a solid foundation for such extensions, since it already enables the aggregation of historically situated edits along semantic dimensions.

At the same time, our current analysis has several limitations. First, our class specialization analysis is performed at the level of concrete edits rather than over the complete historical hierarchy. This choice avoids the substantial cost of materializing the full hierarchy at each historical point, especially given that hierarchies in community-curated KGs are themselves evolving, incomplete, and may contain cycles. However, this also means that our current approach does not fully capture specialization patterns that depend on hierarchical entailment, such as edits involving subclasses, superclass propagation, or constraints whose validation requires reasoning over the class hierarchy.

Addressing this limitation is an important direction for future work. Materializing or approximating the historical hierarchy would enable richer analyses of semantic specialization and would support historical validation of constraint types that require hierarchical reasoning. This is particularly relevant for constraints whose satisfaction depends not only on explicitly edited statements, but also on the state of the surrounding schema at a given point in time. Developing scalable methods for representing and querying such evolving hierarchies is therefore part of our future agenda.

More broadly, the proposed statement lifecycle infrastructure opens several directions beyond the use cases explored in this paper. It could support analyses of community-level specialization, temporal constraint validation, distinction between static and temporal facts, detection of disputed or vandalized statements, and the construction of temporally accurate benchmarks for downstream machine learning tasks. In this sense, our work provides not only an empirical analysis of semantic specialization, but also a reusable basis for studying how collaborative KGs evolve over time.

Declaration on Generative AI

During the preparation of this work, the author(s) used GPT 5.4 in order to: Grammar and spelling checks, wording, and to explore alternative phrasing. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] Erxleben F, Günther M, Krötzsch M, Mendez J, Vrandecic D. Introducing Wikidata to the Linked Data Web. In: Mika P, Tudorache T, Bernstein A, Welty C, Knoblock CA, Vrandecic D, et al., editors. The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I. Lecture Notes in Computer Science. Springer; 2014. p. 50-65. Available from: https://doi.org/10.1007/978-3-319-11964-9_4.

[2] Hogan A, Blomqvist E, Cochez M, d'Amato C, de Melo G, Gutierrez C, et al. Knowledge Graphs. CoRR. 2020;abs/2003.02320. Available from: https://arxiv.org/abs/2003.02320.

[3] Auer S, Bizer C, Kobilarov G, Lehmann J, Cyganiak R, Ives Z. Dbpedia: A nucleus for a web of open data. In: international semantic web conference. Springer; 2007. p. 722-35.

[4] Haller A, Polleres A, Dobriy D, Ferranti N, Méndez SJR. An Analysis of Links in Wikidata. In: 19th European Semantic Web Conference, ESWC 2022. Springer; 2022. Available from: http://polleres.net/publications/halll-etal-2022ESWC.pdf.

[5] Ferranti N, De Souza JF, Ahmetaj S, Polleres A. Formalizing and validating Wikidata's property constraints using SHACL and SPARQL. Semantic Web. 2024;15(6):2333-80.

[6] Polleres A, Pernisch R, Bonifati A, Dell'Aglio D, Dobriy D, Dumbrava S, et al. How does knowledge evolve in open knowledge graphs? Transactions on Graph Data and Knowledge. 2023;1(1):11-1.

[7] Pellissier Tanon T, Suchanek F. Querying the edit history of wikidata. In: European Semantic Web Conference. Springer; 2019. p. 161-6.

[8] Ferranti N, Guimarães D, de Souza JF, Polleres A. Formalizing Repairs for Wikidata Constraint Violations: A Taxonomy and Empirical Analysis. In: International Semantic Web Conference. Springer; 2025. p. 369-87.

[9] Pellissier Tanon T, Bourgaux C, Suchanek F. Learning how to correct a knowledge base from the edit history. In: The World Wide Web Conference; 2019. p. 1465-75.

[10] Hofer M, Töpfer MM, Rost C, Rahm E. DBpedia-TKG: Capturing Wikipedia's Evolution as Temporal Knowledge Graphs. In: European Semantic Web Conference. Springer; 2025. p. 262-79.

[11] Piscopo A, Simperl E. Who models the world? Collaborative ontology creation and user roles in Wikidata. Proceedings of the ACM on Human-Computer Interaction. 2018;2(CSCW):1-18.

[12] Baroncini S, Martorana M, Scrocca M, Smiech Z, Polleres A. Analysing the Evolution of Community-Driven (Sub-) Schemas within Wikidata. In: Wikidata@ ISWC; 2022. .

[13] Angles R, Aranda CB, Hogan A, Rojas C, Vrgoč D. WDBench: A Wikidata Graph Query Benchmark. In: The Semantic Web – ISWC 2022: 21st International Semantic Web Conference, Virtual Event, October 23–27, 2022, Proceedings. Berlin, Heidelberg: Springer-Verlag; 2022. p. 714–731. Available from: https://doi.org/10.1007/978-3-031-19433-7_41.

[14] Bast H, Kalmbach J, Textor-Falconi R, Ullinger C. Sparqloscope: A Generic Benchmark for the Comprehensive and Concise Performance Evaluation of SPARQL Engines. In: The Semantic Web – ISWC 2025: 24th International Semantic Web Conference, Nara, Japan, November 2–6, 2025, Proceedings, Part II. Berlin, Heidelberg: Springer-Verlag; 2025. p. 22–40. Available from: https://doi.org/10.1007/978-3-032-09530-5_2.

[15] Koutsiana E, Reklos I, Alghamdi KS, Jain N, Meroño-Peñuela A, Simperl E. Talking Wikidata: Communication patterns and their impact on community engagement in collaborative knowledge graphs. arXiv preprint arXiv:240718278. 2024.

[16] Ferranti N, Guimaraes D, Souza J, Polleres A. Who Can Fix This? User Recommendation for Knowledge Graph Repair via Embedding-Based Clustering. In: Proceedings of the 13th Knowledge Capture Conference 2025; 2025. p. 170-3.

[17] Marx M, Krötzsch M, Thost V. Logic on MARS: Ontologies for Generalised Property Graphs. In: Sierra C, editor. Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. ijcai.org; 2017. p. 1188-94.

[18] Martin DL, Patel-Schneider PF. Wikidata Constraints on MARS. In: Kaffee L, Tifrea-Marciuska O, Simperl E, Vrandecic D, editors. Proceedings of the 1st Wikidata Workshop (Wikidata 2020) co-located with 19th International Semantic Web Conference(OPub 2020), Virtual Conference, November 2-6, 2020. vol. 2773 of CEUR Workshop Proceedings. CEUR-WS.org; 2020. Available from: https://ceur-ws.org/Vol-2773/paper-12.pdf.

[19] Udrea O, Recupero DR, Subrahmanian VS. Annotated RDF. ACM Trans Comput Logic. 2010 Jan;11(2). Available from: https://doi.org/10.1145/1656242.1656245.

[20] Lopes N, Polleres A, Straccia U, Zimmermann A. AnQL: SPARQLing up annotated RDFS. In: Proceedings of the 9th International Semantic Web Conference (ISWC 2010). vol. 6496 of LNCS. Shanghai, China: Springer; 2010. .

[21] Shannon CE. A mathematical theory of communication. The Bell system technical journal. 1948;27(3):379-423.

[22] Kullback S, Leibler RA. On information and sufficiency. The annals of mathematical statistics. 1951;22(1):79-86.

[23] Vázquez M, Innerebner K, Prock A, Klambauer G, Lex E, Schimunek J, et al. Structure is the Signal: Graph Encodings and GNNs for Constraint Repair in Collaborative KGs. In: 23rd European Semantic Web Conference (ESWC). Dubrovnik, Croatia; 2026. To appear.

Chapter 5

Conclusions and the Way Ahead

This thesis investigated how the maintenance of evolving, collaboratively developed knowledge graphs can be understood and supported when data, schema-related structures, constraints, and the communities maintaining them change over time. Using Wikidata as the main case study, the work examined four connected aspects of this problem: the formal interpretation and validation of community-defined constraints, the repair of detected violations, the identification of contributors with relevant expertise, and the reconstruction of statement histories for fine-grained temporal analysis.

The findings of this thesis indicate that knowledge graph maintenance cannot be reduced to validating a static graph against a fixed set of constraints. The semantics of the constraints themselves need to be made explicit, repairs may involve changes to either data or constraint definitions, maintenance activities are distributed among contributors with different areas of expertise, and the interpretation of changes may depend on the historical state in which statements and constraints coexisted. Taken together, these findings support a view of knowledge graph maintenance as an evolving process in which data quality, modeling decisions, community activity, and temporal context are closely connected.

The remainder of this chapter revisits the four research questions introduced in the Preface. For each research question, the main findings are summarized as a direct answer, related to the publications through which they were developed, and positioned with respect to the related literature. The subsequent sections synthesize these findings across the four research questions, discuss the limitations of the thesis, and identify directions for future work.

5.1 Main Findings and Answers to the Research Questions

This section revisits the proposed research questions, providing direct answers and summarizing key findings.

5.1.1 RQ1: Declarative Formalization and Validation of Property Constraints

RQ1: *How can Wikidata property constraints be formally represented and validated using standard, declarative, Semantic Web technologies?*

Wikidata property constraints can be given declarative representations by mapping their community-defined constraint types and parameters to standard Semantic Web technologies. However, the results show that SHACL-Core alone is not expressive enough to capture the complete property constraint mechanism. Of the 32 constraint types analyzed in **FPCWi22** [8] and **FVPCSWJ24** [5], 26 could be represented in SHACL-Core, while the remaining types require features that are only partially or not reasonably expressible in the core language. SHACL-SPARQL can address these expressivity limitations by using SPARQL to encode the additional validation semantics. In practice, however, once SPARQL is required, directly representing the constraints as SPARQL queries can provide a simpler alternative, since the queries can be evaluated against an RDF export of Wikidata served by a SPARQL engine without requiring a SHACL validator that supports SHACL-SPARQL. In contrast to SHACL-Core, SPARQL therefore provides sufficient flexibility to formulate validation queries for all 32 constraint types considered in the study, although the evaluation of some constraints remains affected by limitations of Wikidata's RDF serialization and the scalability of its public query endpoint [5].

FPCWi22 [8] established the initial mapping between Wikidata property constraints, their constraint-specific qualifiers, SHACL, and reusable SPARQL query patterns. **FVPCSWJ24** [5] extended this work by systematically analyzing the expressivity of SHACL-Core and SHACL-SPARQL, implementing the `wd2shacl` prototype for automatically generating SHACL shapes for expressible constraint instances, and providing declarative SPARQL formalizations for the complete set of constraint types considered in the study. The SPARQL formalizations were further evaluated against Wikidata's violation database reports[1]. The comparison showed broad agreement while also revealing differences related to aspects such as statement ranks, qualifier usage, empty values, and alternative interpretations of the natural-language constraint descriptions. These differences are themselves an important finding, since they demonstrate that declarative formalizations can make otherwise implicit choices in constraint validation explicit and reproducible.

These results complement previous approaches to the formal representation of Wikidata and Wikibase constraints. MARS provides a logical representation of Wikidata property constraints [20], while WShEx proposes a language for describing and validating entities directly in the Wikibase data model [9]. FPCWi22 and FVPCSWJ24 instead focus on operationalizing the property constraint mechanism over Wikidata's RDF representation using standard Semantic Web technologies. The results identify the boundary of SHACL-Core for this use case and show that SHACL-SPARQL can bridge this gap. At the same time, they show that direct SPARQL queries offer a pragmatic alternative when SPARQL is already needed to capture the missing seman-

[1] `https://www.wikidata.org/wiki/Wikidata:Database_reports/Constraint_violations`

tics, particularly when constraint violations are to be retrieved directly from an RDF representation through a SPARQL engine.

The answer to RQ1 therefore goes beyond determining whether Wikidata property constraints can be expressed in SHACL or SPARQL. Community-defined constraint semantics can be made explicit and reproducible using declarative Semantic Web technologies, while the appropriate representation depends on both expressivity and the intended validation setting. SHACL-Core provides a standard constraint representation for a substantial subset of Wikidata property constraint types, whereas direct SPARQL provides greater flexibility for the complete set considered in this thesis and a practical mechanism for retrieving violations from RDF data. These formalizations provide the basis for systematically retrieving constraint violations and, consequently, for the repair analysis developed in **FRTXISWC25** [7] and addressed by RQ2.

5.1.2 RQ2: Repairing Constraint Violations

RQ2: *How can constraint violations in Wikidata be systematically categorized and analyzed in terms of possible repair strategies based on static snapshots of the graph?*

Constraint violations can be systematically characterized through repair patterns derived from the structure of their violation witnesses. In **FRTXISWC25** [7], the semantics of selected Wikidata property constraints are represented using MAPL [20], building on earlier formalizations of Wikidata constraints. By negating the corresponding constraint conditions, the work derives *witness patterns* that identify a conjunction of data statements, statement qualifiers, constraint definitions, and constraint-specific qualifiers that constitute a violation. These witness patterns provide a common basis for describing how a violation can cease to hold and, consequently, for systematically categorizing possible repairs.

The resulting taxonomy distinguishes between changes to the A-box and changes to the T-box. A-box repairs modify the data involved in a violation, including the base statement, additional context statements, and their qualifiers. Depending on the witness pattern, a repair may therefore involve deleting conflicting information, adding missing contextual information, or modifying qualifiers. T-box repairs instead modify the structures against which the data is validated, including property constraint definitions, their constraint-specific qualifiers, and, where relevant, Wikidata's class hierarchy. The same framework also accounts for indirect changes such as re-ranking, constraint exceptions, and value replacements. This distinction is important because a violation does not necessarily imply that the data statement itself is incorrect. It may instead indicate that the corresponding constraint definition or schema-related information has changed.

FRTXISWC25 [7] applied these repair patterns to violations detected in historically aligned Wikidata snapshots from 2019 and 2023. Violations present in the earlier snapshot but absent from the later one were analyzed against the corresponding A-box and T-box changes occurring between the two states. The results show that both categories are relevant in practice. Some constraint types were predominantly associated with changes to data, while others showed substantial T-box activity.

The T-box results are particularly informative because they show that schema-level repair is not limited to deleting problematic constraints. For required qualifier constraints, T-box repairs were dominated by changes to the qualifier set while complete constraint deletions were rare. In particular, 375k repairs, corresponding to 93% of the analyzed repairs for this constraint type, were associated with removing point in time (P585) from the set of required qualifiers for position held (P39). Conversely, adding language of work or name (P407) to newly introduced required qualifier constraints for official website (P856) and described at URL (P973) was associated with more than 1.8 million new violations. These cases illustrate how modifying the parameters of a constraint can simultaneously change the validation status of large numbers of statements without removing the constraint itself.

A similar effect is visible for value-requires-statement constraints. For cast member (P161), changes to the set of accepted occupations included 14 additions and two removals and were associated with approximately 396k repairs as well as 863k new violations. For director (P57), eight occupations were added and one removed, corresponding to approximately 288k repairs and 236k new violations. These examples show that changes to constraint qualifiers can have bidirectional effects: they may resolve existing violations while simultaneously creating new ones.

These results extend prior work on Wikidata repair. Tanon et al. [26] considered data-level corrections and formalized A-box repair operations, but did not account for changes to constraint definitions or their qualifiers. **FRTXISWC25** [7] extends this perspective by using MAPL and the corresponding SPARQL witness patterns to distinguish the components participating in a violation and to derive repair operations over both the A-box data and the T-box.

The answer to RQ2 is therefore that Wikidata constraint repair cannot be adequately characterized as correction of individual data statements alone. Violation witness patterns provide a systematic way to identify which parts of the data or constraint definition may change in order for a violation to disappear, while the empirical analysis shows that both A-box and T-box changes occur in practice. In particular, the large effects produced by modifications to constraint qualifier sets demonstrate that constraint definitions should not be treated as a fixed ground truth against which an evolving knowledge graph is repaired. Rather, the data and the validation criteria evolve together. At the same time, because **FRTXISWC25** infers these repair patterns by comparing two static snapshots, it cannot always establish which intermediate edit actually resolved a violation. This limitation motivates the more fine-grained historical reconstruction developed in **STLC26** and addressed by RQ4.

5.1.3 RQ3: Recommending Contributors for Repair

RQ3: *How can knowledge graph repair processes be supported by recommending users with expertise to resolve detected constraint violations?*

Historical repair activity can provide a practical signal for identifying contributors with prior experience. In **URKCAP25** [6], this idea is investigated by linking historical constraint repairs to the users who performed them and organizing the repaired

entities into semantically coherent regions of Wikidata. Textual descriptions of the entities are represented using sentence embeddings, reduced in dimensionality with PCA/UMAP, and grouped using unsupervised clustering. A newly detected violation can then be associated with one of these topical clusters, allowing users with previous repair activity in the same region to be considered as candidates for resolving it.

The empirical results provide evidence that this approach captures meaningful patterns of specialization. For the *one-of* constraint repair dataset considered in **URKCAP25**, approximately 76% of users concentrated their repair activity within a single topical cluster, while the entropy of users' cluster distributions remained comparatively low. This suggests that, at least for the repair activity studied, contributors frequently operate within relatively narrow semantic regions rather than uniformly across the knowledge graph. The clustering experiments further showed that HDBSCAN produced the strongest cluster separation, reaching a silhouette score of 0.58, while frequently assigning the historically observed repairer to the same cluster as the repaired entity. These findings support the use of semantic clustering as an intermediate representation connecting detected violations with contributors who have worked on related entities in the past.

URKCAP25 also compared two strategies for ranking candidate users within a predicted cluster. The first ranks users according to the frequency of their historical repairs in that cluster, while the second ranks them according to the cosine similarity between the new entity and entities previously repaired by each user. Frequency-based ranking consistently outperformed the similarity-based strategy. In the best configurations, the historically observed repairer was retrieved among the top three candidates in approximately 60% of the test cases, while the difference at *Hits@1* was particularly pronounced, with values around 0.44 for frequency-based ranking compared with 0.10 for cosine-based ranking. This indicates that repeated repair activity within a semantic region is a stronger signal for user recommendation than the fine-grained semantic proximity of an individual entity to previously repaired entities.

These results complement previous studies that characterize contributor roles and editing behavior in Wikidata [21, 22], as well as studies concerned with Wikidata quality and collaborative maintenance [25]. Such work provides evidence that editing and maintenance activities are distributed unevenly across the community, but does not directly address the problem of assigning a newly detected constraint violation to contributors with expertise. Likewise, the historical repair dataset introduced by Tanon et al. [26] provides examples of how inconsistencies were corrected but does not use the identities and prior activity of repairers as a basis for user recommendation. **URKCAP25** extends these lines of work by connecting semantic characteristics of repaired entities with historical contributor activity and evaluating whether this information can support the ranking of candidate repairers.

The answer to RQ3 is therefore that collaborative knowledge graph repair can be supported by exploiting the combination of semantic information about the affected entities and historical information about who previously repaired similar regions of the graph. The results indicate that contributor specialization is sufficiently pronounced in the dataset studied to provide a useful recommendation signal, and that the frequency

of prior repair activity within a semantic region can serve as a simple but effective indicator for ranking candidate users. This should not be interpreted as establishing past repair frequency as a direct measure of expertise, but rather as evidence that historical maintenance behavior contains signals that can be used to support the assignment of future repair tasks. Since **URKCAP25** is exploratory and evaluates a single constraint type, broader conclusions require extending the analysis to additional constraint types and forms of maintenance activity.

5.1.4 RQ4: Statement Lifecycles and Temporal Analysis

RQ4: *How can reconstructing Wikidata statement lifecycles enable temporal analyses of knowledge graph evolution and maintenance on a fine-grained level, beyond static snapshots?*

Fine-grained temporal analyses of Wikidata can be enabled by transforming its page-oriented revision history into statement-level lifecycles that record when statements become valid, when they cease to be valid, and which contributors are associated with these changes. In **STLC26**, the Wikidata page-history dumps are processed to reconstruct the lifecycles of truthy statements rather than treating each historical dump as an independent graph state. The resulting representation associates statements with creation and deletion times and editor provenance, allowing the state of individual statements to be evaluated with respect to a selected point in time. Additional artifacts are derived for class and hierarchy statements, and property constraints, providing the temporal context required by downstream analyses.

This representation addresses a limitation of snapshot-based analysis. Snapshots can establish whether a statement or constraint is present at particular points in time, but they do not expose the sequence of intermediate changes or the provenance of those changes. The original Wikidata history provides this information at the level of page revisions, but does not directly represent the lifecycle of individual statements. **STLC26** bridges these two views by reconstructing statement-level temporal intervals from the revision stream. As a result, statements from different parts of the graph can be compared according to the periods in which they actually coexisted.

The analyses presented in **STLC26** illustrate the relevance of this temporal alignment for studying collaborative maintenance. Property-level editing histories enable the identification of contributors whose activity is concentrated on individual properties or small property portfolios, while class-context histories make it possible to relate an edit to the classes that were valid for the edited entity at the time of the edit. Constraint lifecycles provide a further temporal dimension by allowing constraint maintenance activity to be associated with the contributors and graph states in which it occurred. Together, these analyses show that specialization and maintenance behavior can be studied not only from aggregated edit counts, but also with respect to the semantic context that was valid when an edit was performed.

Historical constraint validation provides a particularly important use case. A statement change should only be interpreted as repairing a constraint violation if the relevant constraint definition and the statements required for its validation were valid at the

same time. Applying a constraint obtained from a later snapshot to an earlier data state can otherwise produce a historically inconsistent interpretation of the repair. By reconstructing statement and constraint lifecycles side by side, **STLC26** provides the machinery needed to determine whether a violation witness existed at a particular time and to relate its disappearance to subsequent changes. This directly addresses the temporal granularity limitation of the snapshot-based repair analysis in **FRTXISWC25**. Moreover, this more fine-grained analysis revealed that this aspect might have been overlooked in earlier works [26], indicating that repair benchmarks might need another look.

These results complement existing approaches to the analysis of evolving knowledge graphs. Snapshot-based studies provide materialized views of graph states and are well suited to comparisons between selected points in time [22, 7]. The Wikidata History Query Service instead provided query access to historical states and revision-level changes [27], while DBpedia-TKG demonstrates how Wikipedia evolution can be represented as a temporal knowledge graph [15]. **STLC26** takes a complementary approach centered on reconstructing reusable statement-level lifecycle artifacts from Wikidata. This makes transaction times and editor provenance available for analyses that require the historical alignment of data statements, schema-related structures, and property constraints.

The answer to RQ4 is therefore that statement lifecycle reconstruction provides a bridge between Wikidata's revision-oriented history and fine-grained temporal analysis at the level at which knowledge graph statements are created, changed, and removed. It enables historically consistent views of statements, constraints, and their surrounding context, and makes it possible to study maintenance activity with respect to the graph state in which it actually occurred.

5.2 Limitations and Future Work

Although the contributions of this thesis provide reusable methods and artifacts for the analysis and maintenance of evolving knowledge graphs, each line of work also has limitations that indicate opportunities for further research.

5.2.1 Formalization and Validation of Property Constraints

The formalization of Wikidata property constraints using SHACL and SPARQL is limited, first, by the expressivity of SHACL-Core. Some Wikidata property constraints cannot be fully represented using SHACL-Core alone. These expressivity limitations can be addressed through SHACL-SPARQL, which allows SPARQL syntax to encode validation semantics that are not available in the core language. In the setting considered in this thesis, however, this introduces an additional validation layer, since execution then depends on a SHACL validator capable of executing SHACL-SPARQL over Wikidata-scale data. Direct SPARQL formalizations provide an alternative in which the same validation logic can be evaluated directly against an RDF export of Wikidata served through a SPARQL engine. For this reason, the thesis favors explicit

SPARQL queries over embedding SPARQL queries inside SHACL shapes. Moreover, one could likewise argue that the practical constraints from a real-world use case like Wikidata can be used to motivate and inform future extensions of new SHACL features.

A second limitation of the SHACL-based approach developed in this thesis is that each shape must be materialized for each individual constraint instance. This can become difficult to manage in a setting such as Wikidata, where many properties are associated with multiple constraint statements and qualifiers. Moreover, changes to constraint definitions would require generating a new set of materialized shapes, which would also need to be maintained historically for analyses over past graph states. By contrast, the direct SPARQL approach is more generic: the same query pattern can be parameterized or filtered by constraint type, target property, or constraint-specific qualifiers. This makes the SPARQL formalization more suitable for retrieving batches of violations using current constraint instances and for conducting empirical analyses across large numbers of constraints, beyond existing benchmarks such as WDBench [2], that does not capture features we require.

5.2.2 Snapshot-based Repair Analysis

The work on proposing a taxonomy of repairs for Wikidata constraint violations is limited by its reliance on fixed snapshot analysis. Although the study selects two snapshots of the Wikidata knowledge graph and extracts both data and constraints according to the state represented in each snapshot, the delta between the two points in time captures only the overall set of changes that occurred between them. It does not preserve the ordering of the individual changes within that interval. Consequently, the approach can identify data and constraint changes that are consistent with the disappearance of a violation witness pattern, but due to the lack of temporal granularity, it cannot always pinpoint which specific change resolved the violation. This limitation motivated the development of **STLC26**, where the extraction of statement lifecycles aims to enable more granular repair analysis, together with historical constraint validation. Fully integrating both works is still open and on our agenda for future work.

5.2.3 Dataset Scope for User Recommendation

The work on recommending users suitable for addressing constraint violations is limited by the scope of the dataset. The study was designed as an exploratory investigation whose main goal was to test an augmentation pipeline for generating embedding-based entity clusters and to evaluate different cluster-quality and recommendation metrics. This required testing a range of configurations, but the experimental scope was restricted to a relatively small dataset focused on a single constraint type from a historical repair dataset. Future work should extend this approach to multiple constraint types, evaluate whether the observed specialization patterns generalize across different kinds of violations, and compare alternative embedding methods for representing entities. A broader evaluation could also investigate whether expertise signals derived from one

type of maintenance activity transfer to other constraint types or topical areas. In the line of the bigger picture painted in Polleres et al. [22], investigating community behaviors and repair quality over time in an evolving manner seems another natural extension of this work.

5.2.4 Statement Lifecycle Analysis and Applications

Finally, the statement lifecycle machinery only provides a first step of investigating the evolution of repairs and behaviors on a more fine-grained level, i.e., the presented work is herein still only at its beginning. In particular, the class-specialization analysis is performed at the level of concrete edits rather than over the complete historical class hierarchy. This design choice avoids the substantial computational cost of materializing the full hierarchy at each historical point, especially since hierarchies in community-curated knowledge graphs are themselves evolving and incomplete, and may contain cycles. However, this also means that the current approach does not fully capture specialization patterns that depend on hierarchical entailment, such as edits involving subclasses, superclass propagation, or constraints whose validation requires reasoning over the class hierarchy. Future work could therefore investigate efficient ways to integrate temporally valid hierarchical information into lifecycle-based analyses. The (non-)scalability of earlier attempts to capture fine-grained evolution analysis in SPARQL itself and running respective temporal endpoints over the whole evolution history also needs further investigation: while we provide a solution to work around these limitations by using bespoke scripts and extraction pipelines, monitoring the evolution of SPARQL engines, their update capabilities and investigate to which exact sizes and evolution rates they actually can scale is a separate, open research question.

More broadly, the statement lifecycle machinery introduced in this thesis opens several further directions for future research. One immediate application concerns the construction of more realistic benchmarks for temporal link prediction. Instead of removing existing triples from a static snapshot and treating them as future links, lifecycle artifacts enable reconstruction of the graph as it existed at a selected timestamp and separation of statements added during a subsequent period. A single traversal of the lifecycle files could therefore produce both a historically accurate training graph and an evaluation set containing the links that actually appeared during the following day, week, or month. Such datasets would reflect the real evolution of Wikidata more closely than artificially generated train-test splits and would allow link prediction methods to be evaluated using only the information available at prediction time.

The same machinery potentially also supports longitudinal studies of knowledge graph embeddings. Embedding models are trained from graph structures, textual descriptions, or combinations of both, and these inputs change as the knowledge graph evolves. It is therefore relevant to investigate whether and how the resulting entity representations also change over time.

The lifecycle artifacts make such studies possible by enabling the reconstruction of temporally accurate graph slices around a selected set of entities. The lookup index created in **STLC26** can first identify the files in which those entities occur as sub-

jects, after which the statements valid at each selected timestamp can be extracted in accordance with the input required by the embedding method. Ordering these slices chronologically would provide a consistent basis for retraining embeddings and for comparing how the representations of the same entities evolve alongside their surrounding subgraphs.

A further direction which we have only exemplified, but not yet systematically analyzed on Chapter 4, is fine-grained analysis of historical constraint violation. Its difficulty depends strongly on the evolving semantics of the constraint type being evaluated. Constraints that can be checked from a single base statement, such as one-of and none-of constraints, can be validated relatively directly once the historically valid constraint definition has been retrieved. Constraints such as item-requires-statement and value-requires-statement are more demanding because their validation depends on whether an additional context statement existed at the same time. The treatment of qualifiers also remains essential. As shown in **STLC26**, qualifier values must be grouped according to the semantics of the corresponding constraint type, since the correct validation condition may involve specific combinations of conjunction and disjunction. Constraints that depend on class hierarchies are even more challenging, because the relevant path must have existed during the period under analysis rather than only in the current graph. Future work should therefore incrementally extend the lifecycle-based validation procedure to scalably monitor constraints that require hierarchy reconstruction, and consider further optimizations (e.g., bespoke indexes) to do so in a scalable manner.

Finally, the lifecycle approach can improve dataset preparation for machine-learning-based knowledge graph repair, such as [26, 28]. A repair example should be derived from the graph and constraint state that existed immediately before the repair was performed. Validating an old statement against a constraint definition taken from a later snapshot can incorrectly classify an edit as a repair of a violation that did not yet exist. By reconstructing the statement, its relevant context, and the corresponding constraint definition at the same point in time, the lifecycle machinery enables the extraction of temporally consistent repair examples. These examples could then be organized into chronological training and evaluation sets, ensuring that a model does not learn from historically inaccurate information. In this way, lifecycle-based reconstruction provides not only a means of analyzing past changes, but also a foundation for more realistic and reliable evaluations of methods for link prediction, representation learning, constraint validation, and knowledge graph repair. Last, but not least, fine-grained edit-level information could also potentially be considered and used for training models that learn repair and maintenance tasks from such fine-grained evolution data: our groundwork could provide the necessary training data.

5.3 Final Remarks

The central conclusion of this thesis is that the maintenance of an evolving knowledge graph cannot be understood solely as the detection and correction of errors in a static collection of statements. In collaboratively maintained knowledge graphs such

as Wikidata, the criteria used to identify potential quality problems are themselves subject to change, the appropriate repair may involve either data or the structures used to validate it, maintenance work is distributed across contributors with different areas of experience, and the interpretation of an edit depends on the historical context in which it was performed. Knowledge graph maintenance is therefore an evolving socio-technical process in which data, validation criteria, contributors, and time need to be considered together.

The contributions of the thesis address these dimensions progressively. Declarative formalizations make community-defined property constraint semantics explicit and reproducible. By expressing Wikidata property constraints through standard Semantic Web technologies such as SHACL and SPARQL, this work also helps bridge the Wikidata and Semantic Web communities: Wikidata's validation practices can be represented using technologies familiar to the Semantic Web community, while Wikidata provides a large-scale, collaboratively maintained setting in which these technologies can be evaluated and applied. Violation witness patterns make it possible to reason systematically about repairs affecting both data and constraint definitions. Historical repair activity provides signals that can support the identification of contributors with relevant prior experience. Statement lifecycle reconstruction adds the temporal dimension required to place these statements, constraints, and maintenance activities in the graph states in which they actually occurred. Together, these contributions move from determining whether a graph state satisfies a constraint toward understanding how quality problems emerge, how they are resolved, who participates in their resolution, and how these processes develop over time.

Although Wikidata provides the primary setting for the investigations in this thesis, the underlying challenges are not specific to a single knowledge graph. Wikidata itself is built using Wikibase, and other knowledge graphs developed with the same framework share important aspects of its data model and infrastructure. Consequently, several of the methods and artifacts developed in this thesis, particularly those concerned with property constraints and statement representation can provide a basis for similar validation and maintenance in other Wikibase-based knowledge graphs. More broadly, any long-lived knowledge graph whose data, schema-related structures, validation rules, and contributor community evolve faces similar questions about the meaning of quality at a particular point in time. The work presented in this thesis therefore contributes both methods tailored to the Wikidata and Wikibase ecosystem and a broader perspective in which knowledge graph quality is understood not only as a property of a graph state, but as part of the continuing process through which structured knowledge is maintained and revised.

Bibliography

[1] Shqiponja Ahmetaj, Robert David, Axel Polleres, and Mantas Simkus. Repairing SHACL constraint violations using answer set programming. In Ulrike Sattler, Aidan Hogan, C. Maria Keet, Valentina Presutti, João Paulo A. Almeida, Hideaki Takeda, Pierre Monnin, Giuseppe Pirrò, and Claudia d'Amato, editors, *The Semantic Web - ISWC 2022 - 21st International Semantic Web Conference, Virtual Event, October 23-27, 2022, Proceedings*, volume 13489 of *Lecture Notes in Computer Science*, pages 375–391. Springer, 2022.

[2] Renzo Angles, Carlos Buil Aranda, Aidan Hogan, Carlos Rojas, and Domagoj Vrgoč. Wdbench: A wikidata graph query benchmark. In Ulrike Sattler, Aidan Hogan, Maria Keet, Valentina Presutti, João Paulo A. Almeida, Hideaki Takeda, Pierre Monnin, Giuseppe Pirrò, and Claudia d'Amato, editors, *The Semantic Web – ISWC 2022*, pages 714–731. Springer International Publishing, 2022.

[3] Dan Brickley and Ramanathan V. Guha. RDF schema 1.1. Technical report, The World Wide Web Consortium (W3C), February 2014.

[4] Fredo Erxleben, Michael Günther, Markus Krötzsch, Julian Mendez, and Denny Vrandecic. Introducing wikidata to the linked data web. In Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig A. Knoblock, Denny Vrandecic, Paul Groth, Natasha F. Noy, Krzysztof Janowicz, and Carole A. Goble, editors, *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, volume 8796 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 2014.

[5] Nicolas Ferranti, Jairo Francisco De Souza, Shqiponja Ahmetaj, and Axel Polleres. Formalizing and validating Wikidata's property constraints using SHACL and SPARQL. *Semantic Web*, 15(6):2333–2380, 2024.

[6] Nicolas Ferranti, Dayane Guimarães, Jairo de Souza, and Axel Polleres. Who can fix this? user recommendation for knowledge graph repair via embedding-based clustering. In Cogan Shimizu, Sebastián Ferrada, and Lalana Kagal, editors, *Proceedings of the 13th Knowledge Capture Conference 2025, K-CAP 2025, Dayton, Ohio, USA, December 10-12, 2025*, pages 170–173. ACM, 2025.

[7] Nicolas Ferranti, Dayane Guimarães, Jairo Francisco de Souza, and Axel Polleres. Formalizing repairs for wikidata constraint violations: A taxonomy and empirical

analysis. In Daniel Garijo, Sabrina Kirrane, Angelo A. Salatino, Cogan Shimizu, Maribel Acosta, Andrea Giovanni Nuzzolese, Sebastián Ferrada, Thibaut Soulard, Kouji Kozaki, Hideaki Takeda, and Anna Lisa Gentile, editors, *The Semantic Web - ISWC 2025 - 24th International Semantic Web Conference, Nara, Japan, November 2-6, 2025, Proceedings, Part I*, volume 16140 of *Lecture Notes in Computer Science*, pages 369–387. Springer, 2025.

[8] Nicolas Ferranti, Axel Polleres, Jairo Francisco de Souza, and Shqiponja Ahmetaj. Formalizing property constraints in wikidata. In Lucie-Aimée Kaffee, Simon Razniewski, Gabriel Amaral, and Kholoud Saad Alghamdi, editors, *Proceedings of the 3rd Wikidata Workshop 2022 co-located with the 21st International Semantic Web Conference (ISWC2022), Virtual Event, Hanghzou, China, October 2022*, volume 3262 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2022.

[9] José Emilio Labra Gayo. Wshex: A language to describe and validate wikibase entities. In Lucie-Aimée Kaffee, Simon Razniewski, Gabriel Amaral, and Kholoud Saad Alghamdi, editors, *Proceedings of the 3rd Wikidata Workshop 2022 co-located with the 21st International Semantic Web Conference (ISWC2022), Virtual Event, Hanghzou, China, October 2022*, volume 3262 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2022.

[10] OWL Working group. OWL 2 web ontology language overview (second edition). Technical report, The World Wide Web Consortium (W3C), December 2012.

[11] Armin Haller, Axel Polleres, Daniil Dobriy, Nicolas Ferranti, and Sergio José Rodríguez Méndez. An analysis of links in wikidata. In Paul Groth, Maria-Esther Vidal, Fabian M. Suchanek, Pedro A. Szekely, Pavan Kapanipathi, Catia Pesquita, Hala Skaf-Molli, and Minna Tamper, editors, *The Semantic Web - 19th International Conference, ESWC 2022, Hersonissos, Crete, Greece, May 29 - June 2, 2022, Proceedings*, volume 13261 of *Lecture Notes in Computer Science*, pages 21–38. Springer, 2022.

[12] Steve Harris and Andy Seaborne. SPARQL 1.1 query language. Technical report, The World Wide Web Consortium (W3C), March 2013.

[13] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *Management Information Systems Quarterly*, 28(1):6, 2008.

[14] Marvin Hofer, Sebastian Hellmann, Milan Dojchinovski, and Johannes Frey. The new dbpedia release cycle: Increasing agility and efficiency in knowledge extraction workflows. In Eva Blomqvist, Paul Groth, Victor de Boer, Tassilo Pellegrini, Mehwish Alam, Tobias Käfer, Peter Kieseberg, Sabrina Kirrane, Albert Meroño-Peñuela, and Harshvardhan J. Pandit, editors, *Semantic Systems. In the Era of Knowledge Graphs - 16th International Conference on Semantic Systems, SEMANTiCS 2020, Amsterdam, The Netherlands, September 7-10, 2020, Proceedings*, Lecture Notes in Computer Science, pages 1–18. Springer, 2020.

[15] Marvin Hofer, Maximilian Mario Töpfer, Christopher Rost, and Erhard Rahm. Dbpedia-tkg: Capturing wikipedia's evolution as temporal knowledge graphs. In Edward Curry, Maribel Acosta, Maria Poveda-Villalón, Marieke van Erp, Adegboyega Ojo, Katja Hose, Cogan Shimizu, and Pasquale Lisena, editors, *The Semantic Web*, pages 262–279, Cham, 2025. Springer Nature Switzerland.

[16] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs. *ACM Comput. Surv.*, 54(4):71:1–71:37, 2022.

[17] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Trans. Neural Networks Learn. Syst.*, 33(2):494–514, 2022.

[18] Holger Knublauch and Dimitris Kontokostas. Shapes constraint language (shacl). Technical report, The World Wide Web Consortium (W3C), July 2017.

[19] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. Dbpedia - A large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[20] David L. Martin and Peter F. Patel-Schneider. Wikidata constraints on MARS. In Lucie-Aimée Kaffee, Oana Tifrea-Marciuska, Elena Simperl, and Denny Vrandecic, editors, *Proceedings of the 1st Wikidata Workshop (Wikidata 2020) co-located with 19th International Semantic Web Conference(OPub 2020), Virtual Conference, November 2-6, 2020*, volume 2773 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020.

[21] 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.

[22] Axel Polleres, Romana Pernisch, Angela Bonifati, Daniele Dell'Aglio, 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, December 2023.

[23] Eric Prud'hommeaux, Iovka Boneva, Jose Emilio Labra Gayo, and Gregg Kellogg. Shape expressions language 2.1. Technical report, W3C Shape Expressions Community Group, October 2019.

[24] Guus Schreiber and Yves Raimond. RDF 1.1 primer. Technical report, The World Wide Web Consortium (W3C), June 2014.

[25] Kartik Shenoy, Filip Ilievski, Daniel Garijo, Daniel Schwabe, and Pedro A. Szekely. A study of the quality of wikidata. *J. Web Semant.*, 72:100679, 2022.

[26] Thomas Pellissier Tanon, Camille Bourgaux, and Fabian M. Suchanek. Learning how to correct a knowledge base from the edit history. In Ling Liu, Ryen W. White, Amin Mantrach, Fabrizio Silvestri, Julian J. McAuley, Ricardo Baeza-Yates, and Leila Zia, editors, *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, pages 1465–1475. ACM, 2019.

[27] Thomas Pellissier Tanon and Fabian M. Suchanek. Querying the edit history of wikidata. In Pascal Hitzler, Sabrina Kirrane, Olaf Hartig, Victor de Boer, Maria-Esther Vidal, Maria Maleshkova, Stefan Schlobach, Karl Hammar, Nelia Lasierra, Steffen Stadtmüller, Katja Hose, and Ruben Verborgh, editors, *The Semantic Web: ESWC 2019 Satellite Events - ESWC 2019 Satellite Events, Portorož, Slovenia, June 2-6, 2019, Revised Selected Papers*, Lecture Notes in Computer Science, pages 161–166. Springer, 2019.

[28] Miguel Vázquez, Kevin Innerebner, Alexander Prock, Günter Klambauer, Elisabeth Lex, Johannes Schimunek, and Axel Polleres. Structure is the signal: Graph encodings and gnns for constraint repair in collaborative kgs. In *European Semantic Web Conference*, pages 278–296. Springer, 2026.