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**.

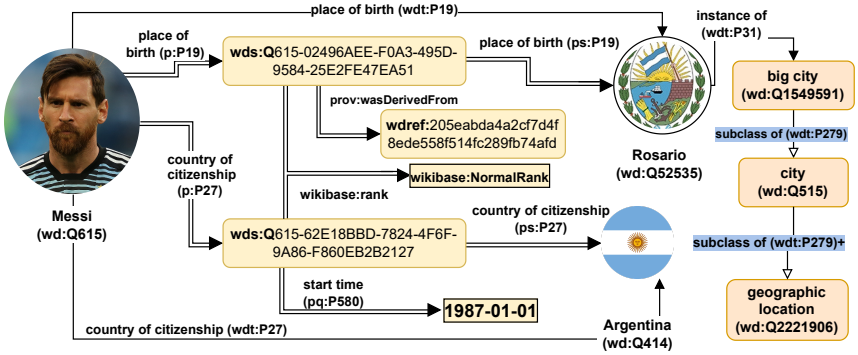
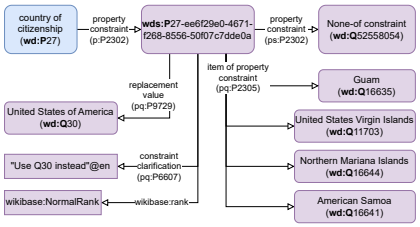


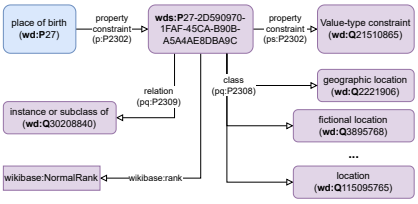
Fig. 1: Wikidata data model: single-line arrows indicate direct A-box (▲-tips) and T-box (△-tips) claims, respectively. Double line arrows denote A-box statement qualifiers, i.e., contextual information.

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 $\mathsf{place\_of\_birth}(\mathsf{Messi}, \mathsf{Rosario})$ shown in Figure 1 conforms to the *completeness constraint* in Fig. 2b since the additional claim $\mathsf{instance\_of}(\mathsf{Rosario}, \mathsf{big\ city})$ *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)	$\underline{p(s,v)} \wedge \mathsf{pc}(p, \text{One-of})@CQ \wedge(\text{iopc} : v) \notin CQ$	[wikibase:directClaim ?P; p:P2302 ?CQ]. ?S ?P ?V. ?CQ ps:P2302 wd:Q21510859. FILTER NOT EXISTS {?CQ pq:P2305 ?V.}
None-of (Q52558054)	$\underline{p(s,v)} \wedge \mathsf{pc}(p, \text{None-of})@CQ \wedge(\text{iopc} : v) \in CQ$	[wikibase:directClaim ?P; p:P2302 ?CQ]. ?S ?P ?V. ?CQ ps:P2302 wd:Q52558054. ?CQ pq:P2305 ?V.
IRS (Q21503247) *(no value)*	$\mathsf{pc}(p, \text{IRS})@CQ \wedge (\text{property} : p_c) \in CQ \wedge\underline{p(s,v)} \wedge \neg\exists v_c.\underline{p_c(s,v_c)} \wedge\neg\exists v_{cq}.(\text{iopc} : v_{cq}) \in CQ$	[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)*	$\mathsf{pc}(p, \text{IRS})@CQ \wedge (\text{property} : p_c) \in CQ \wedge\underline{p(s,v)} \wedge \neg\exists v_c.(\underline{p_c(s,v_c)} \wedge(\text{iopc} : v_c) \in CQ)$	[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)*	$\mathsf{pc}(p, \text{VRS})@CQ \wedge (\text{property} : p_c) \in CQ \wedge\underline{p(s,v)} \wedge \neg\exists v_c.\underline{p_c(v,v_c)} \wedge\neg\exists v_{cq}.(\text{iopc} : v_{cq}) \in CQ$	[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)*	$\mathsf{pc}(p, \text{VRS})@CQ \wedge (\text{property} : p_c) \in CQ \wedge\underline{p(s,v)} \wedge \neg\exists v_c.(\underline{p_c(v,v_c)} \wedge(\text{iopc} : v_c) \in CQ)$	[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)*	$\mathsf{pc}(p, \text{Conflicts-with})@CQ \wedge (\text{property} : p_c) \in CQ \wedge\neg\exists v_{cq}.(\text{iopc} : v_{cq}) \in CQ \wedge\underline{p(s,v)} \wedge \underline{p_c(s,v_c)}$	[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)*	$\mathsf{pc}(p, \text{Conflicts-with})@CQ \wedge (\text{property} : p_c) \in CQ \wedge(\text{iopc} : v_c) \in CQ \wedge\underline{p(s,v)} \wedge \underline{p_c(s,v_c)}$	[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)*	$\mathsf{pc}(p, \text{Single-value})@CQ \wedge (\text{separator} : q) \in CQ \wedge\underline{p(s,v)}@SQ \wedge \underline{p(s,v_c)}@SQ_C \wedge (v \neq v_c) \wedge\neg\exists v_q, v_{qc}.(\underline{\underline{(q : v_q) \in SQ}} \wedge \underline{(q : v_{qc}) \in SQ_C} \wedge v_q \neq v_{qc})$	[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	$\underline{\mathsf{P31}}(v, \underline{v_c}) \wedge \mathsf{P279} * (v_c, c)$	(wdt:P31/wdt:P279*)	Q21503252
subclass_of	$\mathsf{P279} * (v, c)$	(wdt:P279*)	Q21514624
instance_or_subclass_of	$(\underline{\mathsf{P31}}(v, \underline{v_c}) \wedge \mathsf{P279} * (v_c, c)) \vee (\mathsf{P279} * (v, c))$	(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 $\underline{p_c(s,v_c)}$ 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.

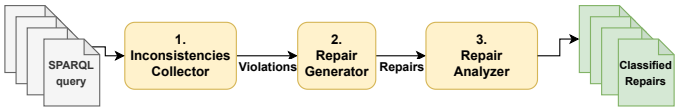


Fig. 3: Three steps experiment pipeline.

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)