Formalizing Property Constraints in Wikidata

Nicolas Ferranti[1], Axel Polleres[1,2], Jairo Francisco de Souza[3] and Shqiponja Ahmetaj[1]

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

Abstract

Constraints play an important role to ensure data integrity. While the Shapes Constraint Language (SHACL) provides a W3C recommendation for validating RDF Knowledge Graphs (KG) against such constraints, real-world KG have adopted their own constraint formalisms. Wikidata (WD), one of the largest collaboratively Open Data Knowledge Graphs available on the Web, represents property constraints through its own RDF data model, within its own authoritative namespaces, which might be an indication that the nature of WD property constraints is different from other Knowledge Graphs. In this paper we investigate the semantics of WD constraints, and unambiguously formalize all current constraints using SPARQL to retrieve violations; we also discuss the expressiveness of WD constraint language compared with SHACL core and discuss the evolution of constraint violations. We found that, while all current WD property constraint types can be expressed using SPARQL, only 86% (26 out of 30) can be expressed using SHACL core: the rest face issues related to using separator properties and arithmetic expressions.

1. Introduction

KG is a computational structure that uses a graph-based model to represent real-world entities, their attributes, and relationships [1]. Since its creation by Wikimedia Foundation in 2012, Wikidata (WD) is continuously growing, and has become one of the largest open KG available on the Web: with more than 13.7B triples[1]. WD has grown larger than DBpedia, one of the main and most central Linked Open Data (LOD) KGs that contains 9.5B triples[2]. One of the main reasons for this growth is WD's user community, with more than 24k active users, humans and bots with different purposes driving the KG's growth in several directions. The large user community is primarily motivated by Wikipedia, as the vast majority of Wikipedia pages incorporate content from WD [2].

There are several ways to create a KG [1]. They can be curated like Cyc [3], extracted from semi-structured web knowledge bases like DBpedia [4] and YAGO [5], collaboratively maintained by a community of users like WD, or extracted from unstructured/semi-structured

Wikidata'22: Wikidata workshop at ISWC 2022
nicolas.ferranti@wu.ac.at (N. Ferranti); axel.polleres@wu.ac.at (A. Polleres); jairo.souza@ice.ufjf.br (J. F. d. Souza); shqiponja.ahmetaj@wu.ac.at (S. Ahmetaj)
0000-0002-5574-1987 (N. Ferranti); 0000-0001-5670-1146 (A. Polleres); 0000-0002-0911-7980 (J. F. d. Souza); 0000-0003-3165-3568 (S. Ahmetaj)

[1] https://w.wiki/5myX, as from October 2022
[2] https://lod-cloud.net/dataset/dbpedia, as from March 2022

sources like NELL [6]. Usually, the organization responsible for the KG development has to choose a trade-off between accuracy and coverage, the more content they try to cover the greater the margin for including errors. Therefore, it is common to apply refinement techniques after the KG construction. The focus of graph refinement techniques are the data layer (instance level) or the terminology layer (concept level) [1].

The WD community approach is focused on the data layer with the terminology layer evolving as data evolves "on the side", whereas other KGs typically deploy separately defined ontologies or schemas. That is, WD does not have a predefined formal ontology [7]. Rather, in order to reinforce consistent usage of the community-developed terminology, separate WD projects have emerged to specify *constraints*, which serve as a means to identify errors in the data layer. However, none of these projects deploys the current W3C recommendation for validating RDF graphs against constraints, namely, the Shapes Constraint Language (SHACL): instead, Wikidata uses its own representation model for what they call *property constraints*. Indeed, WD's property constraint language could be argued to predate SHACL: the first property constraint was created on WD in 2015, when SHACL was still only a W3C working draft.

While SHACL relies on standardised validators to identify inconsistencies, WD violation reports are calculated within an ad-hoc extension of Wikibase [8], in particular, the *Property Constraints* project[3] which we focus on in this paper. Differences between representation models, as well as this informal/operational definition of WD constraints (with unfortunately even partially unavailable source code) obfuscate the formal semantics of WD's property constraints, and whether these actually differ from what is expressible on common RDF graphs using SHACL. To close this gap, we investigate the semantics of WD property constraints (WDPC), compare their representation and expressiveness with SHACL, and finally take a look at the extent of constraint violations (CVs) within WD. Our two main contributions are: (1) we argue whether and how WDPC could be expressed as SHACL constraints using SHACL-core language, and discuss expressiveness issues comparing SHACL's and Wikidata's approaches to define property constraints. With translating a large part of WD's property constraints to SHACL, we also contribute a large-scale testbed for SHACL. (2) We then unambiguously formalize *all* WDPC as SPARQL queries, which provide a declarative means to express constraints while being also operationalizable.

This paper is structured as follows. Section 2 discusses how to represent the semantics of WDPC in SHACL-core; since this is not possible in all cases, in Section 3 we present a complete mapping of property constraints to SPARQL instead. We briefly discuss how WDPC have evolved over time, arguing that our SPARQL formalization could be used for operationalizing such analyses. After discussing related works on constraint formalization and quality analysis for KGs in Section 4, we conclude in Section 5 with pointers to future research directions.

2. Expressing Wikidata Constraints with SHACL

A number of dedicated, authoritative [9] RDF namespaces are used to represent different aspects of the same property - and therefore also in the representation of property constraints - in WD, as illustrated in Fig. 1.

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

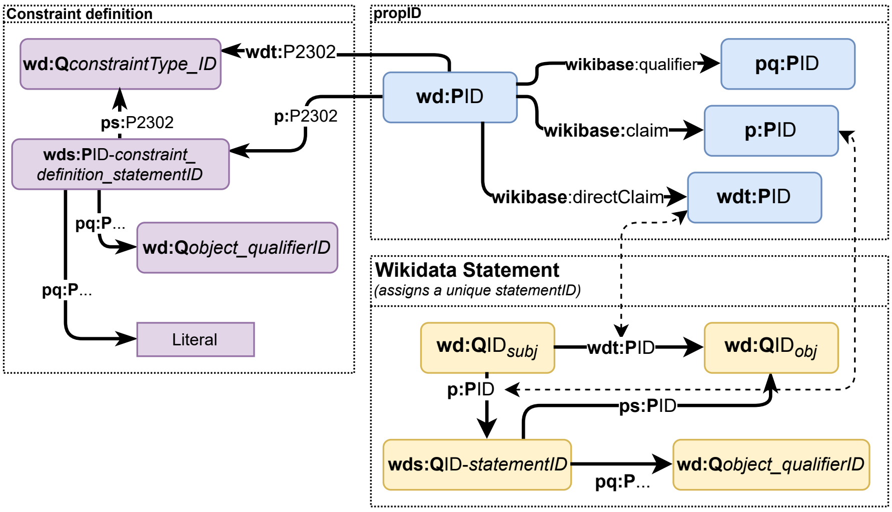
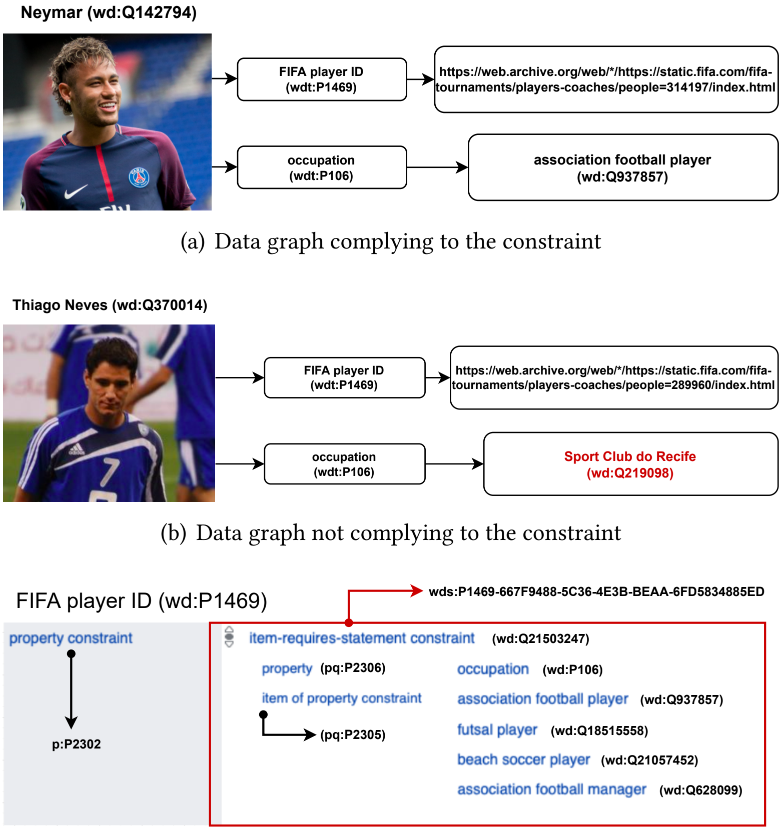


Figure 1: The Wikidata meta-model based on dedicated namespaces

Fig. 2 shows a concrete example of item-requires-statement constraint for the "FIFA player ID" property. WD concepts are specified using the prefix *wd*, used for **entities** like "Neymar" (Q142794) but also **properties** like "FIFA player ID" (P1469), i.e., both entities and properties can be "referred to as concepts".

Triples using (*wd*-prefixed) concepts in subject or object positions comprise facts about these concepts, where the *wdt* namespace (used for a property id in predicate position) is used to reference a direct relationship between concepts. The *wd* prefix is never used in the predicate position. In Fig. 2(a), the triple {`wd:Q142794 wdt:P106 wd:Q937857`} establishes a direct relation between two items: a football player and his occupation, P106 (occupation) which is referred to as a relation using the *wdt* prefix. Relationships can also be qualified, i.e. statements about a particular factual (*wdt*) relationship between concepts can be further described, introducing a reification (see also [10]) mechanism through the consistent use of the dedicated namespaces *p* and *wds*, which facilitates to refer to the statements made on a particular subject, as illustrated on an abstract level in Fig. 1, and again, in the concrete example of Fig. 2(c). WD also allows the use of qualifiers on this statement level, i.e., metadata can be added to describe statements more accurately. Qualifiers are represented with *pq* prefix, exclusively used on statements (i.e. for properties of subject URLS with the prefix *wds*).

WDPC refer to particular, community-defined constraint types, such as for instance an *item-requires-statement* constraint, where specific instantiations of this constraint type are defined as qualified statements on a particular property that should fulfill this constraint. For instance an item-requires constraint (Q21503247) for the property "FIFA player ID" (P1469) is illustrated in Fig. 2(c): the constraint states that if an item has a "FIFA player ID" (P1469), the same item should also have as occupation (P106) one of the following items: association football player (Q937857), futsal player (Q18515558), beach soccer player (Q21057452), or association football manager (Q628099). These concrete restrictions are defined through qualifiers specific to the particular constraint type and property; we discuss property constraint qualifiers in detail in


Neymar (wd:Q142794)
FIFA player ID (wdt:P1469)
https://web.archive.org/web/*/https://static.fifa.com/fifa-tournaments/players-coaches/people=314197/index.html
occupation (wdt:P106)
association football player (wd:Q937857)
Thiago Neves (wd:Q370014)
FIFA player ID (wdt:P1469)
https://web.archive.org/web/*/https://static.fifa.com/fifa-tournaments/players-coaches/people=289960/index.html
occupation (wdt:P106)
Sport Club do Recife (wd:Q219098)
FIFA player ID (wd:P1469)
wds:P1469-667F9488-5C36-4E3B-BEAA-6FD5834885ED
property constraint
p:P2302
item-requires-statement constraint (wd:Q21503247)
property (pq:P2306)
occupation (wd:P106)
item of property constraint
(pq:P2305)
association football player (wd:Q937857)
futsal player (wd:Q18515558)
beach soccer player (wd:Q21057452)
association football manager (wd:Q628099)

(a) Data graph complying to the constraint

(b) Data graph not complying to the constraint

(c) Wikidata property constraint representation format

Figure 2: Example of a WD constraint and data graphs with different behaviors (as in 2022-03-29)

Section 2.1.

To model our concrete constraint from Fig. 2, first the triple {`wd:P1469 p:P2302 wds:P1469-667F9488-5C36-4E3B-BEAA-6FD5834885ED`} connects the property “FIFA player ID” (wd:P1469) to a statement node that is the bridge to the qualifiers; here, the *p* prefix is used by property “property constraint” (P2302) to describe a relation between an entity (*wd*) and a statement (*wds*): this statement (consisting of several claims) then is used to specify concrete constraint requirements by means of particular qualifiers. Properties that implement a constraint can use different values in the qualifiers to customize the constraint type (in our case *item-requirements-statement* constraint) to the particular property (in our case the “FIFA player ID”): as shown in Fig. 2(c), here the *wds* statement has as “target” property (pq:P2306) the item “occupation” (wd:P106), and lists allowed values for this property as “item of property constraint” (pq:P2305).

Fig. 2(a) and 2(b) present two different data graphs: the first one complying with *item-requires-statement* constraint and the second one violating it. Both subjects have “FIFA player ID” but only “Neymar” (Q142794) has a valid “occupation”, whereas “Thiago Neves” (Q370014) does not.

The WD community has defined a wide range of such property constraint types, some of which resemble known RDFS axioms, for instance the *type constraint* (Q21503250) which is similar in spirit to the constraint reading of an *rdfs:domain* statement. Other property constraint types describe more complex relationships, such as for instance the *contemporary constraint* (Q25796498) which defines that if two entities are connected by a specific property, both of them must coexist at some point in time [11].

2.1. Property Constraint Types and Qualifiers

To date, WD defines 30 property constraint types represented as subclasses (P279) of property constraint (Q21502402). Table 1 gives an overview of all the constraint types. One can also find in this overview the number of different properties that use each constraint (from March 2022) and the list of all property qualifiers that can be used by each contraint type.

In the following we will present the results of a one-by-one analysis of these constraint types, discussing expressibility in terms of mappings to SHACL and SPARQL. Before we turn to the actual mappings, we need to discuss the common qualifiers used to define property constraints, which we will use for test compliance. For instance, recall from Fig. 2(c), where the *property* and *item of property constraint* qualifiers were used to express the property and allowed values that should present to fulfill the *item-requires-statement* constraint. Finding ways to represent the semantics of qualifiers will be fundamental to understand which particular constraint types can or cannot be expressed declaratively in SHACL-core. The set of qualifiers we will use to characterize the semantics of property constraints on a particular property (**P**ID) is as follows:

- **Format as a regular expression (P1793)**: used only by the *format constraint* to express that the value of **P**ID should comply with a predefined regular expression. In SHACL, it can be expressed through *sh:pattern*.
- **Property (P2306)**: used to check the availability of an (additional) property P' on the subjects of **P**ID; it is usually complemented by *Item of property constraint* which restricts the objects of P'. Property (P2306) can be represented through SHACL's (more generic) *sh:path* which represents the path to be taken until the node to be tested.
- **Item of property constraint (P2305)**: checks items expected as values of either **P**ID or the property P' indicated by P2306. SHACL has a set of components to restrict values that can be used to express the same meaning, for instance, *sh:hasvalue* and *sh:in*.
- **Separator (P4155)**: A qualifier used by constraints to express that multiple statements for **P**ID can exist as long as the values of the separator properties are distinct: the separator as such implements a "composite key" for constraint validation. To the best of our knowledge, there is no equivalent SHACL component to model composite keys.
- **Relation (P2309)** and **Class (P2308)**: Relation and Class are qualifiers used together. Relation represents the relationship expected between the subject or object and a set of predefined items described by Class (P2308). The possible relationships are: *instance of*, *subclass of*, and *instance or subclass of*.[4] SHACL component *sh:class* can be used to check the type of an item, and it also includes hierarchical reasoning to check subclasses.

[4] We note that in WD, it is not explicitly specified whether such subclass of relationships should be interpreted transitively, or whether *instance of* relationships should also affect instances of subclasses. In our encodings, we took a choice encoding these as property paths, similar to [12].

However, 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. sh:path* can also be used together with *sh:hasValue* or *sh:in* to combine the path expected and the object values expected at the end of this path.

- **Range checking qualifiers:** Minimum value (P2313), Maximum value (P2312), maximum date (P2311), and minimum date (P2310): these all describe ranges of values or dates. The most similar properties in SHACL to represent range restriction are *sh:minExclusive* and *sh:maxExclusive* for open intervals, and *sh:minInclusive* and *sh:maxInclusive* for closed intervals.
- **Exception to the constraint (P2303)**: is the set of **P**ID's subjects that should not be tested by the constraint. SHACL has no direct component for exceptions; however, it is possible to combine within *sh:or* component two acceptance clauses as follows: either the subject is within (sh:in) the listed exceptions *or* it must conform to the constraint (example in Section 2.2).
- **Constraint Scope (P4680)**: Defines the scope where the constraint should be checked. Scopes can be identified according to the namespace used by a property in a triple, such as "as main values" (*wdt*) or "as qualifiers" (*pq*). When creating a corresponding SHACL Shape for a constraint using this qualifier, we explicitly refer to the respective prefix(es).

There are six qualifiers for constraints not discussed in this paper because they do not represent information relevant for constraint checking, but supplementary constraint information: Syntax Clarification (P2916), Constraint Clarification (P6607), Replacement Property (P6824), Replacement Value (P9729), Constraint status (P2316), and Reason for deprecated rank (P2241).

2.2. Mapping WD Property constraints to SHACL

Fig. 3 exemplifies the translation of constraints into a SHACL shapes graph for our running example. The shape in Fig. 3(a) is applied to all nodes that are subjects of *wdt:FifaPlayerId* (line 8), there must be at least one path from these nodes using the property *wdt:Occupation* (lines 10 and 11) to one of the items listed in *sh:in* (line 12). This shape clarifies that it is possible to encode allowed values in SHACL, something considered uncertain in [8]. Shenoy et al. [8] also argue that it is unclear if SHACL can encode exceptions in property constraints. If we take our previous example in Fig. 2, suppose that Thiago Neves (Q370014) is an exception to the constraint. In this case, the graph complies when one of the two conditions is satisfied: either the subject is an exception – in our case only Thiago Neves (Q370014) – or the subject has one of the required Occupations (P106); Fig. 3(b) details the SHACL shape for this example.

Table 1 presents the entire set of analyzed constraint types, their WD IDs, as well as a column to state whether it was possible to map the constraint type to SHACL (and SPARQL, respectively, see Section 3 below). The particular SHACL encodings can be found in an online repository[5]. SHACL has many components to express constraints. Core components are recognisable by any SHACL validator, however there are additional components that are not necessarily recognized by validators (e.g. *sh:sparql*). In this work, the expressiveness of WD constraints in SHACL is discussed in terms of core components only.

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

Constraints requiring the existence of a specific statement, e.g. *item-requires-statement*, *required qualifier*, and *one-of* are naturally captured by SHACL Core constraint components due to the main forms of construction that are based on choosing a target node, verifying the existence of a path and, possibly, verifying the existence of a value. Table 1 shows that the vast majority of WD constraints can be rewritten in SHACL (86%), some could only be partially written (7%), and some can not be represented in SHACL (7%). The group of partially representable constraints consists of constraints that use *separator* (P4155) qualifiers: while it is straightforwardly possible to verify the uniqueness of a property value with respect to the claim subject, when a separator qualifier property is used, it could be understood as a "composite key"; however since it is not possible to compare the values of different paths that correspond to unique combinations of separators in SHACL, that is, to distinguish different nodes matching the same regular path expression, these cannot be expressed in SHACL core. Concerning the 7% that could not be represented, the *difference-within-range* constraint (Q21510854) requires the difference between two values to be calculated and compared to a predefined range: while SHACL core has components to check for equalities (*sh:equals*), inequality (*sh:disjoint*, and *sh:lessThan*), arithmetic operations are not included. Finally, *Single-best-value* (Q52060874) constraints could again not be expressed due to the absence of operators to check the existence of multiple equal values obtained by different paths following the same regular expression (similar to the problem with separator qualifiers mentioned above).

We note that beyond its core language, SHACL provides means to refine constraints in terms of full SPARQL queries through a SPARQL-based constraint component (*sh:sparql*); indeed using full SPARQL, all WDPC can be represented.

3. Operationalizing Wikidata Constraints with SPARQL

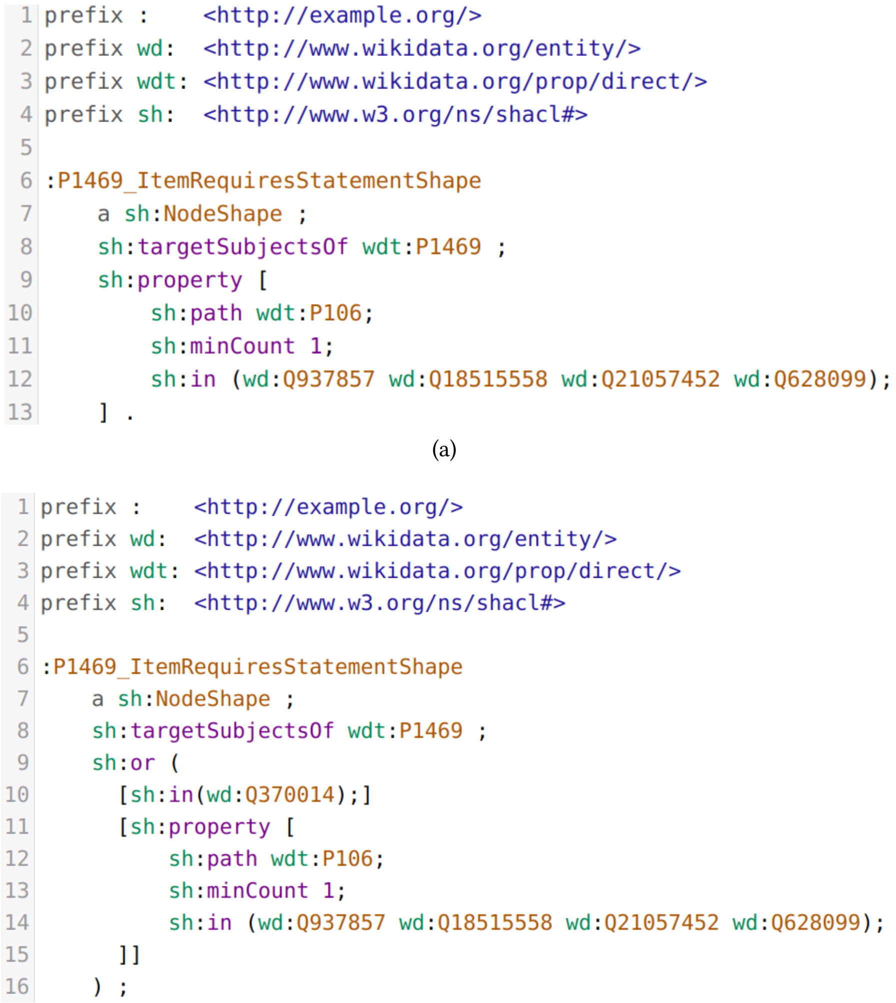
While the previous section showed partial expressibility of WDPC in SHACL, we still lack a fully *operationalizable* formalization: a constraint representation formalism needs, in our opinion, both to be (i) operationalizable – in the sense of being able to compute and report inconsistencies – as well as (ii) declarative – in the sense of an unambiguous, exchangeable formalization, capable of understanding the meaning of constraints.

The availability of WD's database reports web page[6] 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, and it also indicates that indeed there is an operationalized workflow to check these constraints.

The property pages can be accessed where one can take a detailed look at the inconsistent claims per violated property.[7] Unfortunately, the result of the operationalization on WD's database reports is only available in HTML format, and moreover, the code behind is not publicly available. To close this gap, we can use SPARQL: since WD as an RDF graph can be

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

[7] 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 **7 violations**, retrieved 05 May 2022.



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

:P1469_ItemRequiresStatementShape
    a sh:NodeShape ;
    sh:targetSubjectsOf wdt:P1469 ;
    sh:property [
        sh:path wdt:P106;
        sh:minCount 1;
        sh:in (wd:Q937857 wd:Q18515558 wd:Q21057452 wd:Q628099);
    ] .
```

(a)

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

:P1469_ItemRequiresStatementShape
    a sh:NodeShape ;
    sh:targetSubjectsOf wdt:P1469 ;
    sh:or (
      [sh:in(wd:Q370014);]
      [sh:property [
          sh:path wdt:P106;
          sh:minCount 1;
          sh:in (wd:Q937857 wd:Q18515558 wd:Q21057452 wd:Q628099);
      ]]
    ) ;
```

(b)

Figure 3: SHACL Shape for "FIFA Player ID" and item-requires-statement constraint. (a) is the original shape and (b) is the example enconding exceptions

queried through a SPARQL query service, if we manage to express CVs per constraint type as SPARQL queries, we can benefit from the query language's declarative nature both providing a declarative *and* operationalizable constraint specification.

3.1. Expressing and validating WD property constraints in SPARQL

Based on Fig. 2, we illustrate the structure of our constraint validation using SPARQL queries to capture the semantics of WDPC and provide a formal interpretation, and at the same time shall enable retrieval of inconsistent data. Fig. 4 represents the query for the *item-requires-statement* constraint: for this constraint type a required property and its required value(s) needs to be checked. The query structure of Fig. 4 is divided into groups as follows.

Group 1 represents data returned by the query, usually it is composed by the claim containing the property that is violating the constraint (line 2), and extra information about the claim or the constraint – in our case, the property missing for the subject – and, if given, the constraint status or reason for deprecation (line 3). **Group 2** is the beginning of *where* clause, matching properties (and associated statements) that use the particular constraint (lines 5 and 6) – in our case for the *item-requires-statement* (Q21503247): to retrieve properties using another specific property constraint type, it is only necessary to change the constraint id in line 6.

Group 3: statements from Group 2 are then used to retrieve relevant constraint qualifiers for the specific constraint type (lines 7 and 8), i.e., in our case the required value (line 7) and property P' (line 8), as explained in Section 2.1 above, plus optional qualifiers such as the constraint status or information about constraint deprecation (lines 9+10). **Group 4** matches the actual triples that will be checked for compliance (line 13): as we can see in this example, we need to navigate between the different property namespaces described in Fig. 1 above before matching subject and object (line 12). **Group 5** combines the statement qualifiers values from Group 3 and the triples from Group 4 to create the violation patterns, for *item-requires-statement* constraint the goal is to remove from the results all triples that have required property along with the required value (lines 15 and 16) and remove also the exceptions to the constraint (line 18).

```
 1. SELECT
 2.  ?s ?wdt_property ?o                                                      } Group 1
 3.  ?wdt_required_property ?constraint_status ?deprecated_reason
 4. WHERE {
 5.  ?wd_property p:P2302 ?statement.                                       } Group 2
 6.  ?statement ps:P2302 wd:Q21503247. # item-requires-statement
 7.  ?statement pq:P2305 ?has_required_value.
 8.  ?statement pq:P2306/wikibase:directClaim ?wdt_required_property.
 9.  OPTIONAL {?statement pq:P2316 ?constraint_status}.                      } Group 3
10.  OPTIONAL {?statement pq:P2241 ?deprecated_reason}.
11.
12.  ?wd_property wikibase:directClaim ?wdt_property.   } Group 4
13.  ?s ?wdt_property ?o.
14.  FILTER NOT EXISTS {
15.    ?s ?wdt_required_property ?any_required_value.
16.    ?statement pq:P2305 ?any_required_value.          } Group 5
17.  }
18.  FILTER NOT EXISTS {?statement pq:P2303 ?s.}
19. }
```

Figure 4: Query for *item-requires-statement* constraint with a required value.

We have encoded all 30 current constraint types (some of which in separate queries for different variations) in queries following similar patterns corresponding to **Group1–Group 5** in our example where 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 (containing examples for checking particular properties per constraint type) can be found – as shortcut-links to WD's query service – in Table 1, and also in the repository (cf. Footnote 5). For instance, our query https://w.wiki/58KM implementing the FIFA Player ID's item-requires-statement constraint returned **8 violations**, as opposed to the 7 on

WD's database report page (cf. Footnote 7) at the time of writing: our formalization captures a violation not reported on the DB page; upon checking back, the "new" violation was added after the batch report generation, but is immediately captured by our "realtime" query.

4. Related Work

In this paper, we target the formalization of WDPC using SPARQL and discuss expressiveness when compared to SHACL. Various other prior works have already sought solutions to improve the data quality of KGs, by defining constraints or by means of ontological reasoning.

Data restrictions within WD are also discussed by the community and implemented through other projects using some pre-established technologies. For instance, the *Wikidata Schemas* project[8] relies on the Shape Expressions language (ShEx) [13]. ShEx is a formal modeling and validation language for RDF data, which allows the declaration of expected properties, cardinalities, and the type and structure of their objects. As opposed to property constraints, the *Schemas Project* is focused on defining entity (Wikidata concepts) restrictions.

Erxleben et al. [2] exploit properties describing taxonomic relations in WD to extract an OWL ontology. They propose the extraction of schematic information from property constraints and discuss their expressibility in terms of OWL axioms. 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 WDPC.

Abián et al. [11] propose a definition of contemporary constraint that was later adopted by WDPC. Shenoy et al. [8] present a quality analysis of WD focusing on correctness, checking for weak statements under three main indicators: CV, 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. [14] present a tool for designing/editing shape constraints in SHACL and ShEx suggesting WD as a potential use case, but – to the best of our knowledge – without exhaustively covering or discussing the existing WDPC.

Apart from works specifically on WD, in [15] the authors try to identify systematic errors in the construction of DBpedia, their method uses the DOLCE ontology as background knowledge to find inconsistencies in the assertional axioms. First, the target information is extracted from DBpedia and linked to the DOLCE ontology, then a reasoner check if this new data creates inconsistency. Before, Bischof et al. [12] already highlighted logical inconsistencies in DBpedia which can be detected using OWL QL, rewritten to SPARQL1.1 property paths.

5. Conclusions

We have formalized the current 30 different property constraint types of WD using SPARQL and discussed ways to encode them with W3C's standard recommendation mechanism for formalizing constraints over RDF Knowledge Graphs, SHACL. This study made it possible to

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

Table 1
Wikidata property constraints set

#ID	#Name	#SHACL	#PropCount	#SPARQL	#Qualifiers
Q52004125	allowed entity types	Yes	8446	https://w.wiki/58K4 (Wikibase item Q29934200) https://w.wiki/58Mn (Wikibase property Q29934218) https://w.wiki/58Mq (Wikibase lexeme Q51885771) https://w.wiki/58Mr (Wikibase form Q54285143) https://w.wiki/58Mt (Wikibase sense Q54285715)	P2303;P2305;P2316;P4680;P6607
Q21510851	allowed qualifiers	Yes	725	https://w.wiki/58MR	P2241;P2303;P2304;P2306;P2316;P6607
Q21514353	allowed units	Yes	492	https://w.wiki/58Kk	P2303;P2305;P2316;P6607
Q54554025	citation needed	Yes	338	https://w.wiki/58MT	P2303;P2316;P6607
Q21510852	Commons link	Yes	78	https://w.wiki/58LW	P2307;P2316
Q21502838	conflicts-with	Yes	4096	https://w.wiki/58LU	P2303;P2304;P2305;P2306;P2316;P2916;P6607;P6824;P9729
Q25796498	contemporary	Yes	124	https://w.wiki/58ML	P2303;P2316;P6607
Q21510854	difference-within-range	No	10	https://api.triplydb.com/s/r8SK2sO8L	P2303;P2306;P2312;P2313;P2316;P4680;P6607
Q21502410	distinct-values	Partially	6601	https://w.wiki/58LT	P2303;P2304;P2316;P2916;P4155;P6607
Q21502404	format	Yes	7235	https://w.wiki/58LP	P1793;P2241;P2303;P2316;P2916;P4680;P6607
Q52848401	integer	Yes	165	https://w.wiki/58LN	P2303;P2316
Q21503247	item-requires-statement	Yes	5215	https://w.wiki/58LJ (only req. prop.) https://w.wiki/58LM (also req. val.)	P2241;P2303;P2304;P2305;P2306;P2316;P2916;P4680;P6607
Q108139345	label in language	Yes	129	https://w.wiki/58LG	P2316;P424
Q55819106	lexeme requires language	Yes	97	https://w.wiki/58LF	P2305;P6607
Q55819078	lexeme requires lexical category	Yes	9	https://w.wiki/58LZ	P2305
Q64006792	lexeme value requires lexical category	Yes	1	https://w.wiki/58Lb	P2305
Q21510857	multi-value	Yes	31	https://w.wiki/58Lf	P2304;P2316;P6607
Q51723761	no-bounds	Yes	74	https://w.wiki/58Le	P2303;P2316
Q52558054	none-of	Yes	39	https://w.wiki/58Lk	P2303;P2304;P2305;P2316;P2916;P6104;P6607;P6824;P9729
Q21510859	one-of	Yes	208	https://w.wiki/58Lm	P2241;P2303;P2305;P2316;P2916;P6607
Q52712340	one-of qualifier value property	Yes	5	https://w.wiki/58Ln	P2305;P2306
Q53869507	property scope	Yes	14703	https://w.wiki/58Mi (as main value) https://w.wiki/58Mk (as qualifier) https://w.wiki/58Mm (as reference)	P2303;P2304;P2316;P2916;P4680;P5314;P6607
Q21510860	range	Yes	345	https://w.wiki/58Mb (for values) https://w.wiki/58Mg (for dates)	P2303;P2310;P2311;P2312;P2313;P2316;P6607
Q21510856	required qualifier	Yes	382	https://w.wiki/58Lo	P2241;P2303;P2304;P2306;P2316;P4680;P6607
Q52060874	single-best-value	No	159	https://w.wiki/58Lz	P2303;P2316;P4155;P4680;P6607
Q19474404	single-value	Partially	6483	https://w.wiki/58M4	P2241;P2303;P2304;P2316;P2916;P4155;P4680;P6607
Q21510862	symmetric	Yes	45	https://w.wiki/58NA	P2303;P2316
Q21503250	type	Yes	6033	https://w.wiki/58MX (instanceOf) https://w.wiki/58MV (subclassOf) https://w.wiki/58MW (instanceOrSubclassOf)	P2241;P2303;P2304;P2308;P2309;P2316;P4680;P6607
Q21510864	value-requires-statement	Yes	370	https://w.wiki/58M9 (only req. prop.) https://w.wiki/58M7 (also req. val.)	P2241;P2303;P2304;P2305;P2306;P2316;P4680;P6607
Q21510865	value-type	Yes	1025	https://w.wiki/58ME (instanceOf) https://w.wiki/58MF (subclassOf) https://w.wiki/58MN (instanceOrSubclass)	P2303;P2304;P2308;P2309;P2316;P2916;P6607

clarify to which extent SHACL can represent constraints of a widely used real-world KG: one of our results is a collection of practical SHACL constraints that can be used in a large and growing real world dataset; indeed the non-avilability of practical SHACL performance benchmarks has already been emphasized by [16], where we believe our work could be a significant step forward. Other results we presented include clarifications of heretofore uncertain issues, such as the representability of permitted entities and exceptions in WDPC within SHACL [8]. We also could argue the non-expressibility of certain WD constraints, due to the impossibility to compare values obtained through different paths matching the same regular path expression within SHACL-core. These issues could be addressed when we used SPARQL to validate constraints, where indeed all 30 constraints could be formalized, including additional constraint types *Citation needed* (Q54554025) and *Single best value* (Q52060874) that are not even yet reported by WD's official constraint reporting tool, cf. Footnote 6 above.

The formalization and operationalization of property constraints establishes a mutual relationship with the WD community: analyzing the formalization helps to enrich the way constraints are modeled and vice versa. We hope that this article stimulates discussions in the community to enrich the representation of constraints that still might have subjective interpretations, as well as support the evolution of validation approaches in SHACL as WD can now be considered as a large benchmark dataset for SHACL validators.

As future work, we plan to use the results of this paper to systematically collect/analyse the kinds of CVs in WD and study their patterns. Understanding violations and their evolution is fundamental to identify modeling or other systematic data quality issues and propose further refinements, especially in collaboratively and dynamically created KGs such as WD. Proposing refinements is a process that can be envisioned when taking into account the repair information declaratively represented in and retrievable through operationalizable constraints.

Acknowledgments

Axel Polleres' work is supported by funding in the European Commission's Horizon 2020 Research Program under Grant Agreement Number 957402 (TEAMING.AI).

References

[1] H. Paulheim, Knowledge graph refinement: A survey of approaches and evaluation methods, Semantic web 8 (2017) 489–508.

[2] F. Erxleben, M. Günther, M. Krötzsch, J. Mendez, D. Vrandečić, Introducing wikidata to the linked data web, in: International semantic web conference, Springer, 2014, pp. 50–65.

[3] D. B. Lenat, Cyc: A large-scale investment in knowledge infrastructure, Communications of the ACM 38 (1995) 33–38.

[4] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, et al., Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia, Semantic web 6 (2015) 167–195.

[5] M. Fabian, K. Gjergji, W. Gerhard, et al., Yago: A core of semantic knowledge unifying

wordnet and wikipedia, in: 16th International world wide web conference, WWW, 2007, pp. 697–706.

[6] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka, T. M. Mitchell, Toward an architecture for never-ending language learning, in: Twenty-Fourth AAAI conference on artificial intelligence, 2010.

[7] A. Piscopo, E. Simperl, Who models the world? collaborative ontology creation and user roles in wikidata, Proceedings of the ACM on Human-Computer Interaction 2 (2018) 1–18.

[8] K. Shenoy, F. Ilievski, D. Garijo, D. Schwabe, P. Szekely, A study of the quality of wikidata, Journal of Web Semantics 72 (2022) 100679.

[9] A. Haller, A. Polleres, D. Dobriy, N. Ferranti, S. Rodríguez Méndez, An analysis of links in wikidata, in: ESWC 2022-19th European Semantic Web Conference, 2022.

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

[11] D. Abián, J. Bernad, R. Trillo-Lado, Using contemporary constraints to ensure data consistency, in: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, 2019, pp. 2303–2310.

[12] S. Bischof, M. Krötzsch, A. Polleres, S. Rudolph, Schema-agnostic query rewriting in sparql 1.1, in: International semantic web conference, Springer, 2014, pp. 584–600.

[13] S. Staworko, I. Boneva, J. E. L. Gayo, S. Hym, E. G. Prud'Hommeaux, H. Solbrig, Complexity and expressiveness of shex for rdf, in: 18th International Conference on Database Theory (ICDT 2015), 2015.

[14] I. Boneva, J. Dusart, D. F. Alvarez, J. E. L. Gayo, Shape designer for shex and shacl constraints, in: ISWC 2019-18th International Semantic Web Conference, 2019.

[15] H. Paulheim, A. Gangemi, Serving dbpedia with dolce–more than just adding a cherry on top, in: International semantic web conference, Springer, 2015, pp. 180–196.

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