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

Nicolas Ferranti [a,*], Jairo Francisco De Souza [b] and Axel Polleres [a]

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

Abstract. In this paper, we delve into the crucial role of constraints in maintaining data integrity in knowledge graphs with a specific focus on Wikidata, one of the largest collaboratively open data knowledge graphs on the web. Despite the availability of a W3C recommendation for validating RDF Knowledge Graphs against constraints via the Shapes Constraint Language (SHACL), however, Wikidata currently represents its property constraints through its own RDF data model, using proprietary authoritative namespaces, and – partially ambiguous – natural language definitions. In order to close this gap, we investigate the semantics of Wikidata property constraints, by formalizing them using SHACL and SPARQL. While SHACL Core's expressivity turns out to be insufficient for expressing all Wikidata property constraint types, we present SPARQL queries to identify violations for all current Wikidata constraint types. We compare the semantics of this unambiguous SPARQL formalisation with Wikidata's violation reporting system and discuss limitations in terms of evaluation via Wikidata's SPARQL query endpoint, due to its current scalability. Our study, on the one hand, sheds light on the unique characteristics of constraints in Wikidata that potentially have implications for future efforts to improve the quality and accuracy of data in collaborative knowledge graphs. On the other hand, as a "byproduct", our formalisation extends existing benchmarks for both SHACL and SPARQL with a challenging, large scale real-world use case.

Keywords: Wikidata, Data quality, Knowledge Graphs, Constraints, Shapes Constraint Language, SPARQL

1. Introduction

A Knowledge Graph (KG) uses a graph-based model to represent real-world entities, their attributes, and relationships [1]. Entities can be anything that can be uniquely identified and described, such as people, places, things, or concepts. Statement, representing relationships between entities are represented as edges in the graph, while the attributes of entities are again graph nodes. As such, KGs can be used to represent a wide range of information, including encyclopedic knowledge, scientific data, and corporate data, and, – along with meta-information attached to statements – also contextual information, such as who *provenance*, *preference* amongst statements, or temporal context (e.g. *when* a statement was added, so called transaction time, or was valid, called *validity time*, cf. e.g. [2, 3]. In order to process such KGs, the Semantic Web community has defined standards such as

- the Resource Description Framework (RDF) [4] to publish and interchange KGs,
- a Standard Protocol and RDF Query Language (SPARQL) [5] to query KGs,

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

- RDF Schema (RDFS) [6] and the Web Ontology Language (OWL) [7] to define and describe the schema of KGs in RDF itself,
- the Shapes Constraint Language (SHACL) [8, 9] to validate KGs.

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

Since its creation by Wikimedia Foundation in 2012, Wikidata (WD) has become one of the largest such KGs, publicly available on the Web, with more than 100M items[1] and 14B triples.[2] One of the main factors responsible for this growth is WD's user community, with more than 24k active users (humans and bots). The large user community is primarily motivated by Wikipedia, as the vast majority of Wikipedia pages incorporate content from WD [10].

In terms of supporting above-mentioned Semantic Web standards, the WD KG is available in standard RDF format and can be queried via a public SPARQL endpoint. Yet, WD does neither adhere to OWL/RDFS, nor SHACL: while other knowledge graphs often have predefined formal ontologies or schemas defined in RDFS and OWL, WD takes a different approach, with the community focusing on the development of the data layer (A-box) and the terminology layer (T-Box) evolving alongside it. This means that WD does not have a single, pre-defined formal ontology [11] adhering to RDFS/OWL's well-defined semantics. 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 wrt. vocabulary usage. However, again none of these projects deploys the current W3C recommendation for validating RDF graphs against constraints, namely, SHACL.

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

When it comes to how these constraints should be interpreted/checked, there is a description in natural language for each constraint type available in their pages (e.g. Single-value constraint[4]). While SHACL relies on standardised validators to identify inconsistencies, WD violation reports are calculated within an ad-hoc extension of Wikibase [12]. The violation reports are published in an HTML page[5], however, the approach behind the generation of the reports is not public, and there is a maximum limit of violations displayed by each constraint type in the property pages. For a community-based KG with billions of triples, projects like Property Constraints represent a key resource for creating tools to assist in the analysis and refinement of inconsistent data on WD, however the development of such tools is hampered by the way this data is currently collected and made available. Since the only official description about how to check property constraints is in natural language, the semantics of property constraints may be subject to subjective interpretations.

In this paper, we explore the use of SPARQL and SHACL as tools for formalizing constraints in WD. Previous approaches to constraint formalization in WD have relied on encoding constraints in a separated logical framework [13] and ad-hoc calculations, but the use of standardized tools like SPARQL and SHACL could provide more accurate, open, and efficient means of identifying and addressing inconsistencies in the data. Our main contributions are as follows:

- We study in how far the expressiveness of SHACL's core language is sufficient to express WD property constraints and come to the conclusion that SHACL's core language is not expressive enough to represent all WD

[1] https://www.wikidata.org/wiki/Wikidata:Statistics, as from January 2023
[2] https://short.wu.ac.at/7t66, last accessed 13 February 2013
[3] https://www.wikidata.org/wiki/Help:Property_constraints_portal
[4] https://www.wikidata.org/wiki/Help:Property_constraints_portal/Single_value
[5] https://www.wikidata.org/wiki/Wikidata:Database_reports/Constraint_violations/Summary

property constraints. Out of 32 property constraint types, SHACL's core language can not express two property constraints, one is not reasonably expressible, and another three can only be expressed partially.
– For WD property constraints expressible in SHACL, we present a tool to automatically translate such constraints to SHACL; the tool can benefit also other Wikibase KGs that import WD property constraints.
– We unambiguously formalize all 32 WD property constraint types as SPARQL queries which provide a declarative means to express constraints while being also operationalizable.[6] SPARQL queries offer the possibility of checking the violations in real time on the WD SPARQL endpoint, as well as the flexibility to collect useful information to analyse the usage of a constraint such as status, reasons for deprecation, and exceptions.
– We present a comparison of our SPARQL approach to the current WD violation reports, demonstrating the feasibility of using SPARQL to actually check constraints: particularly, we highlight potential ambiguities and reasons for deviations in violations found with our approach compared to the WD violation reports.

The remainder of this paper is structured as follows. Section 2 presents background on WD's meta-model and explains how property constraints are represented using this model.

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

Yet, as not all WD property constraints are expressible in SHACL, in Section 4 we instead present a complete mapping of all WD property constraints to SPARQL: after briefly discussing how WD property constraints have evolved over time, we argue that our SPARQL formalization could be used for unambiguously operationalizing such analyses continuously. As a demonstration of feasibility, we present a detailed analysis and experiments, comparing violations found by our approach with the officially reported constraint violations by WD itself in Sections 5+6; here, firstly, we particularly discuss and highlight differences and ambiguities that we believe are clarified by our approach in a reproducible manner. Secondly, we also discuss, that constraint checking via SPARQL over the WD endpoint partially leads to timeouts: as such, our constraint checking queries, which involve complex queries motivated by a clear, real use case, going beyond recent benchmarks such as WDbench [14], could be viewed as a novel SPARQL benchmark on their own. After discussing related works on constraint formalization and quality analysis for KGs in Section 7, we conclude in Section 8 with pointers to future research directions.

2. Background

Constraints play an important role in specifying rules for data, defining the requirements to prevent it from becoming corrupt, and ensuring integrity. In relational databases, standardized integrity constraints are typically associated with attributes and relations and enforced by the database management system (DBMS). Imposing such constraints in NoSQL DBMS however mostly shifts the responsibility to the application developer, and indeed KG are usually stored in NoSQL DBMS [15], which has driven the development of own bespoke constraint representations and checking techniques. Standardized ways to express and validate constraints in graph-based data models like RDF have only recently been defined: The W3C recommends using the Shapes Constraint Language[7] (SHACL), which defines integrity constraints in the form of graph patterns (called "shapes") to be checked against a data item (called shape "targets") in the graph. Although SHACL is a W3C recommendation, constraints modeling varies (and pre-dates SHACL) among Open Knowledge Graphs such as DBpedia and WD. DBpedia, like other open KGs (e.g. YAGO, GeoNames), makes use of an ontology to model data. An ontology (formalized in standards such as RDFS and OWL) provides an axiomatic specification of the KG's schema and has the advantage of being machine-readable, which enables reasoning over the data to find inconsistencies through (deductive) inference: indeed, several studies have shown ontological inconsistencies in DBpedia [16, 17]. Yet, while ontology languages allow the definition of a set of axioms that define sets of individuals and the relationship types allowed between them, they are

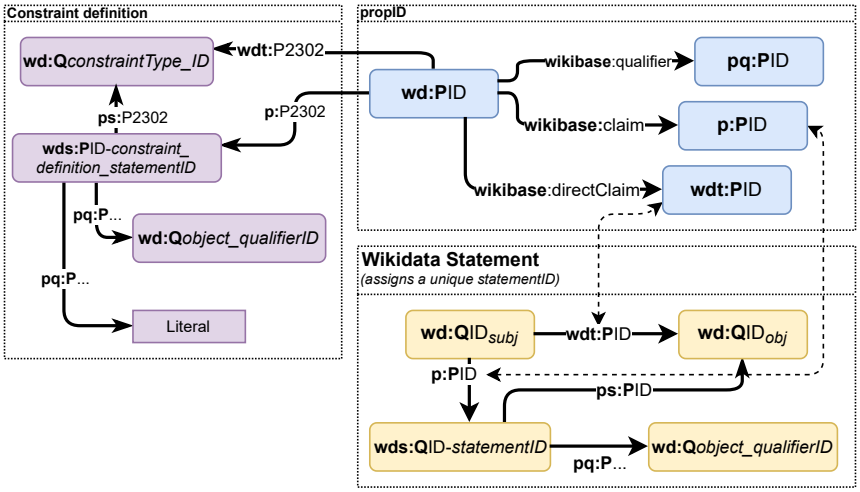
[6] Notably, as it turns out, some constraint types can only be partially evaluated online due to incomplete RDF representation of WD's native data model on WD SPARQL query endpoint.
[7] https://www.w3.org/TR/shacl/

most useful for deductive reasoning tasks such as node classification or overall satisfiability checking, rather than pinpointing single constraint violations.

Standard ontological inference as a means to detect inconsistencies is not directly applicable to the approach taken by WD: the WD community is primarily focused on the data layer, and the terminology layer evolves in the background as editors add/update new facts, potentially introducing new properties and classes. In fact, there is neither a strict distinction between the data and terminology layers, nor does the terminology layer rely on OWL/RDFS [18]. Rather, proprietary, likewise community-driven, adhoc processes have been set up within WD to define constraints on the terminology use: in particular, the *Property Constraints* project[8] which we will focus on in this paper, aims at defining restrictions applied to the usage of WD properties. In the following subsection, we introduce the data modeling adopted by WD, including details of how property constraints are represented.

2.1. Wikidata's data modeling

A number of dedicated, authoritative [18] 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.



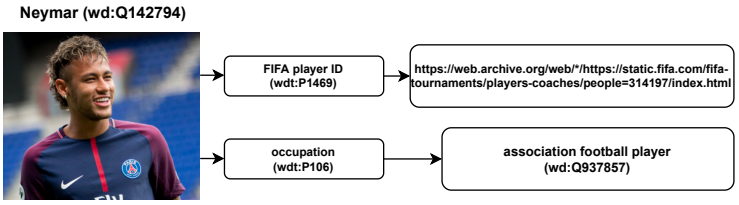
Prefix	Namespace	Purpose
wd	http://www.wikidata.org/entity/	used to identify WD concepts (entities or properties)
wdt	http://www.wikidata.org/prop/direct/	used for direct property relationships (claims) between (*wd*) concepts.
p	http://www.wikidata.org/prop/	relates a (*wd* concept to a *wds* statement node further describing the direct *wdt* claim.
wds	http://www.wikidata.org/entity/statement/	used for statement nodes, i.e., intermediate nodes that link claims to statement qualifiers.
pq	http://www.wikidata.org/prop/qualifier/	used for "qualifying", adding meta-data to (*wds*) statements.
ps	http://www.wikidata.org/prop/statement/	used to "re-connect" (*wds*) statements to the original claim's object value.

Fig. 1. The Wikidata meta-model based on dedicated namespaces

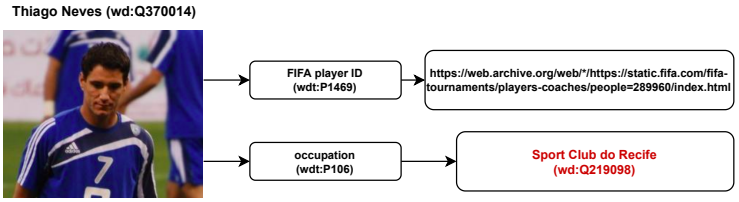
Fig. 2 shows a concrete example of the so called *item-requires-statement* constraint for the "FIFA player ID" (P1469) property. Wikidata concepts are specified using the prefix *wd*, used for **entities** like "Neymar" (Q142794) but also **properties** like "FIFA player ID" (P1469), i.e., both entities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

Triples using such (*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. That is, the *wd* prefix is never used in the predicate position. In Fig. 2(a), the triple

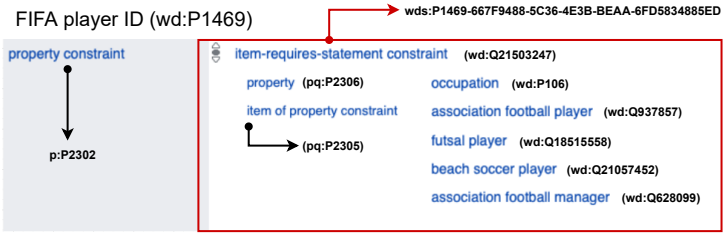
[8] https://www.wikidata.org/wiki/Wikidata:WikiProject_property_constraints



(a) Data graph complying to the constraint



(b) Data graph not complying to the constraint



(c) Wikidata property constraint representation format

Fig. 2. Example of a Wikidata constraint and data graphs with different behaviors (as in 2022-03-29)

`{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 [19]) 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). Wikidata also allows the use of qualifiers on this statement level, i.e., metadata can be added to describe statements more accurately. Qualifiers are represented through the *pq* prefix which is exclusively used on statements (i.e. for properties of subject URLS with the prefix *wds*).

WD property constraints 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), this very 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 will discuss these property constraint specific qualifiers in detail in Section 3.

To model our concrete constraint from Fig. 2(c), 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 or the *value-type constraint* (Q21510865) which resembles *rdfs:range*. 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 [20].

To date, WD defines 32 property constraint types represented as subclasses (P279) of property constraint (Q21502402). Table 6 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 February 2023) and the list of all property qualifiers that can be used by each constraint type.

3. Expressing Wikidata Constraints with SHACL

In the following we will present the results of a one-by-one analysis of WD's property constraint types, discussing expressibility in terms of mappings to SHACL and SPARQL. Before we turn to the actual mappings, we need to discuss the "language" Wikidata has introduced to define property constraints, including common qualifiers used to define such constraints, which we will use for testing 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 be 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. The set of qualifiers we will use to characterize the semantics of property constraints on a particular property (PID) 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 (or objects) of **P**ID; it is usually complemented by *Item of property constraint* which restricts the objects (or subjects) 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 qualifiers are 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*.[9] SHACL component

[9] 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 [17]. Find details in the supplementary material.

sh:class can be used to check the type of an item, and it also includes hierarchical reasoning to check subclasses. 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)**: represents the set of **P**ID's subjects that should not be tested by the constraint. SHACL has no direct component to express 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 (for an example cf. Section 3.1).
- **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).
- **Reason for deprecated rank (P2241)**: qualifier used to express that a constraint is deprecated and indicate the particular deprecation reason, for instance, "obsolete" (Q107356532), "constraint provides suggestions for manual input" (Q99460987), etc. SHACL has the boolean component *sh:deactivated* to disable a shape, however the values are boolean only and it is not possible to represent specific reasons with this SHACL property.
- **Group By (P2304)**: a qualifier used to group constraint violations in constraint violation reports. SHACL has component *sh:group* to indicate that the shape belongs to a group of related property shapes.
- **Constraint status (P2316)**: represent the severity degree of a constraint as mandatory or suggested constraint. The most similar SHACL component that can be used to express status is *sh:severity* with one of the three possible severity degrees, i.e, *sh:Info*, *sh:Warning*, or *sh:Violation*.
- **Syntax Clarification (P2916)** and **Constraint Clarification (P6607)**: Both represent text descriptions for constraints. *Constraint Clarification* was originally created to describe the purpose of the constraint for a property, it is also used to describe suggested repairs in textual form. *Syntax Clarification* provides a textual description of the regex syntax of a value. While this qualifier cannot be explicitly formalized, the SHACL component *sh:description* is used to provide descriptions of the property in a given context and it is also used to describe a meaning in natural language for the property shape.
- **replacement property (P6824) and replacement value (P9729)**: Both represent suggestions to fix inconsistencies by replacing properties or the value of a **P**ID claim. SHACL components are designed to capture inconsistencies but not to fix them, this kind of information present in WD property constraints can not be represented with SHACL explicitly; however, it could inform choosing/computing specific repairs (as, for instance, recently discussed for SHACL in [21, 22]).

3.1. 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* (lines 12 and 13). We note that this shape clarifies that it is possible to encode allowed values in SHACL, something that was considered uncertain in [12]. Shenoy et al. [12] 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 in the exceptions – 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.

```
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)

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

Table 6 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 4 below). The particular SHACL encodings can be found in an online repository of our paper.[10]

SHACL has many components to express constraints. The core constraint components are recognisable by any SHACL validator, however there are additional components that are not necessarily recognized by validators. In the scope of this work, the expressiveness of WD constraints in SHACL is discussed in terms of SHACL core components only.

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 6 shows that the vast majority of WD constraints can be rewritten in SHACL (81%), some could only be partially written (9%), some can not be represented in SHACL (6%), and one is not reasonably expressible (3%).

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. Another constraint type, so called allowed-entity-type constraints (Q52004125), are – while representable in SHACL[11] – only partially verifiable, due to an incomplete export of entity types in Wikidata's RDF graph; we will discuss details of this issue when discussing the SPARQL operationalisation in the Section 4.

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. Additionally, the *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). Finally, the *allowed qualifiers constraint* (Q21510851) is also not directly expressible in SHACL Core: this constraint limits the allowed qualifier properties with a specific property to a fixed set, meaning that the use of all *other* qualifier constraints needs to be limited: the problem herein lies in SHACL there is no way to list non-allowed paths implicitly (e.g. by referring

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

[11] https://github.com/nicolasferranti/wikidata-constraints-formalization/blob/main/constraints-formalization/allowed%20entity%20types/shacl_shape_P3831.ttl

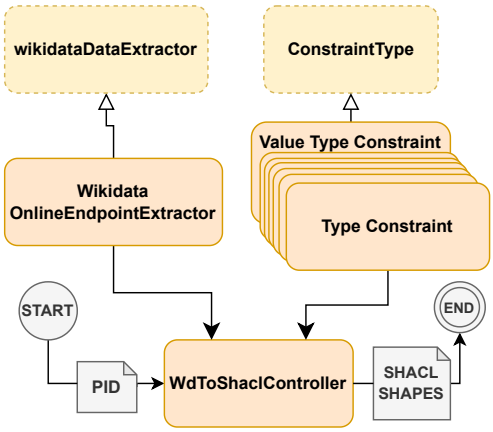


Fig. 4. Wikidata to SHACL architecture. Dashed lines represent abstract classes.

to a path/property via a specific type). We present an admittedly "clumsy" workaround, i.e., listing all non-allowed qualifiers *explicitly* in our github repository[12].

3.2. *wd2shacl - a tool to automatically convert Wikidata constraints to SHACL*

After having identified SHACL core expressible WD constraint types, we also have developed a tool prototype to provide automatic conversion from the WD property constraint model to the corresponding SHACL shape, which shall enable the creation of SHACL real-world benchmark from Wikidata: Wd2shacl enforces Wikidata property constraint testing with SHACL and makes mappings available for any Wikidata property. To this end, first, the example SHACL shapes (such as the ones in Fig. 3(a) and 3(b)) were generalized to become templates for specific constraint types, e.g. removing the specific qualifier values assigned to a single property. Then, our wd2shacl tool populates these templates according to the actual qualifier values that characterize a specific property.

The architecture of this tool is shown in Fig. 4. As input, it is provided the PID of the desired property. The controller (WdToShaclController) uses a data extractor to collect all constraint types from the property, as well as the qualifiers describing them. The data extractor (wikidataDataExtractor) directly queries the respective qualifier statements from a Wikibase instance via SPARQL, where our current implementation directly uses Wikidata's endpoint (WikidataOnlineEndpointExtractor); alternative inherited extractor classes can be created to return data in a predefined format if necessary (e.g. to query from HDT archived versions of Wikidata [13]), or also to query alternative Wikibase instances: indeed, other Wikibase instances, such as the EU Knowledge Graph[14] re-use WD's property constraint mechanism and could be likewise checked using the tool via alternative extractors.

After collecting constraint types and qualifiers for the input property, the controller instantiates the template to create respective SHACL constraints for the extracted applicable constraint types. A specific class (ConstraintType) is implemented for each SHACL expressible constraint type to create the SHACL shape by combining the template with the given qualifiers. The controller returns the populated SHACL templates as SHACL Turtle files, containing the required prefixes and a list of SHACL shapes, written using SHACL core language, for the constraint types associated with the PID. The tool is freely available online within our github repository[15].

3.3. *Limitations of SHACL for WD property constraint checking*

We note the following limitations for the implementation of WD property constraints via SHACL:

[12] https://github.com/nicolasferranti/wikidata-constraints-formalization/blob/main/constraints-formalization/allowed%20qualifiers/shacl_shape_P6.ttl

[13] Available at https://www.rdfhdt.org/datasets/

[14] https://linkedopendata.eu/wiki/The_EU_Knowledge_Graph

[15] https://github.com/nicolasferranti/wikidata-constraints-formalization/tree/main/shacl-generator

– firstly, we can cover only a subset of SHACL expressible WD property constraints;
– secondly, our approach instantiates separate SHACL shapes separately for each property and constraint type;
– thirdly, the capacity of checking these constraints against the whole WD graph – to the best of our knowledge – goes beyond the scalability (and feature coverage) of existing SHACL tools.

As for the first item, obviously the above mentioned issues regarding expressivity of WD property constraints within SHACL Core limit its applicability. Additionally, non-core features, unfortunately, are are not mandatorily (and thus rarely) implemented by SHACL validators so far. Yet, beyond its core language, SHACL provides means to refine constraints in terms of full SPARQL queries through a SPARQL-based constraint component (*sh:sparql*).

We should note at this point that all our derived SHACL constraints are non-recursive, but make extensive use of also SHACL Core features that have not been universally implemented yet in validators: for instance a popular approach to validate non-recursive (and partially even recursive) SHACL is translating shapes to single SPARQL queries that can be directly checked on a SPARQL endpoint, cf. [23, 24]. However, at a closer look, Corman et al.'s work, which is focused on theoretical expressiveness of recursive SHACL in SPARQL and presents the translation to SPARQL queries over an endpoint rather as a "side product", does not cover certain language features, or respectively suggests to implement them through normalization to a "core" of the SHACL Core language by pre-processing. While this is theoretically feasible, in practice it yields infeasible queries.

The approach by Corman et al. yield's SPARQL queries reporting violations by `NOT EXISTS` sub-queries expressing the conditions to be verified by the respective targets specified by the constraint. The construction of these subqueries is modularly defined via the SHACL grammar.

While a full account of such translations is out of the scope of our paper, as an example, let us demonstrate the principle with a straightforward SPARQL translation of a simple *conflicts-with* (Q21502838) constraint for the property *family name* (P734); according to the constraint property it should not be used together with the property P1560 (given name version for other gender), as expressible concisely in the following SHACL shape:

```
1   @prefix wdt:    <http://www.wikidata.org/prop/direct/> .
2   @prefix sh:     <http://www.w3.org/ns/shacl#> .
3
4   [
5    a sh:NodeShape ;
6    sh:targetSubjectsOf wdt:P734;
7    sh:property [
8        sh:path wdt:P1560 ;
9          sh:maxCount 0 ;
10    ]
11  ].
```

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

```
1  SELECT * WHERE {
2      ?T <http://www.wikidata.org/prop/direct/P734> [] .
3      FILTER NOT EXISTS { ?T <http://www.wikidata.org/prop/direct/P734> [] .
4            FILTER NOT EXISTS {  ?T <http://www.wikidata.org/prop/direct/P1560> [] . } } }
```

Unfortunately, this query currently times out on WD's SPARQL endpoint, which is in fact not surprising due to the "double-negation" yielding from a modular translation, as the NOT EXISTS operator is particularly hard to evaluate for SPARQL engines. Also note that our SPARQL formulation, to be executable at all, needs to "copy" the target definition within the verification part (line 3), due to the broken recursive correlation semantics of the (NOT) EXISTS operator in SPARQL, cf. [25, 26].

A more direct and crisp, and also executable formulation of this query can be easily constructed:

```
1   SELECT * WHERE {
2       ?T <http://www.wikidata.org/prop/direct/P734> [] .
3       ?T <http://www.wikidata.org/prop/direct/P1560> [] . }
```

In fact, we claim that violations of this particular constraint type, i.e. the *conflicts-with* constraint, could be checked more generally on *all* properties in one go, with a single SPARQL query[16]:

[16] https://short.wu.ac.at/8tb6

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

Indeed, this single query checks *all* violations of the conflicts-with constraint type and returns all violations of conflicts-with constraints at once: intuitively, line 2-3 looks up properties `?wdtprop` and the corresponding entity `?wdprop` (cf. the illustration in Figure 1) having a conflict-with constraint (line 3), and retrieving the respective conflicting property `?conflicting_wdtprop` (line 4); finally, line 5-6 checks the existence of a statement using both conflicting properties for the target `?T`. The query indeed is executable on the WD SPARQL endpoint, cf. Footnote 16, with the slight limitation that we need to LIMIT the results retrieved, as overall too many such violations exist to be retrieved via the UI; more details on that in our experiments section.

Overall, we hope this illustrative example has sufficiently motivated that a direct translation of property constraints to SPARQL has advantages over SHACL for various reasons. That is, while in this section we in principle have made a case for using a subset of WD's property constraints as a "playground" to automatically generate a large testbed for SHACL (Core) validators, we also hope to have convinced the reader that this approach is not (yet) practically feasible. Therefore, let us now turn to generalize this approach to *fully* operationalize WD constraint validation via SPARQL.

4. Operationalizing Wikidata Constraints with SPARQL

As opposed to the prototypical nature of the previous section, here we aim at a fully *operationalizable* formalization: we propose SPARQL as a constraint representation formalism that fulfills both the requirements to be (i) operationalizable – in the sense of being able to compute and report inconsistencies – 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[17], 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 already. The current reports provide access to a separate page for each property listed where one can take a detailed look at the inconsistent claims per violated property[18]. Unfortunately though, the result of the operationalization shown on WD's database reports pages is only available in HTML format, and moreover, the code behind is not publicly available. That is, the current operationalization is neither decalarative nor is the code openly available.

As a summary from WD's database reports, Fig. 5 shows the development of property constraints over time for the 10 most violated constraints, according to the WD database reports web page. We observed that since the introduction of WD property constraints in 2015, the total number of constraints has grown from 19 in 2015 to 32 in 2023; new constraints were created, evolved, or ceased to exist. Data in Fig. 5 point to an increase in the number of violations for the *one-of* (Q21510859) constraint type. *Required qualifier* (Q21510856) constraints were introduced and began to be analyzed in 2019 only and are already emerging among the main causes of violations. The modeling of constraints is constantly evolving, for instance, constraints *used for values only* (Q21528958), *used as reference* (Q21528959), and *used as qualifier* (Q21510863) no longer exist and were migrated to a single constraint type: *property scope* (Q53869507). As such, we emphasize that our endeavour to model and formalize constraint types as SPARQL queries should not be viewed as a once-of exercise: rather we aim at proposing to rethink the *process* of developing such constraints themselves in terms of such SPARQL queries to be included as declarative an unambiguous means for their definitions.

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

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

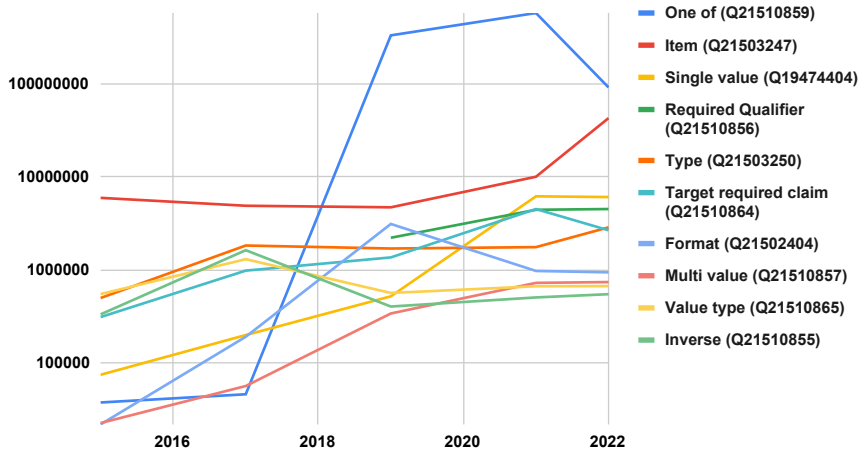


Fig. 5. #violations for Top 10 most violated constraint types (logarithmic scale)

We have already motivated SPARQL as a candidate to declaratively model constraint checks per constraint type in the previous section: since WD as an RDF graph can be queried through a SPARQL query service, if we manage to express constraint violations per constraint type as SPARQL queries, we can benefit from the query language's declarative nature both providing a declarative *and* operationalizable constraint specification.

4.1. Understanding the Semantics of Constraints

One of the biggest challenges in formalizing constraint types, that they are not yet represented using an unambiguous logical language, was the understanding of the semantics behind the constraint: that is, WD property constraints model defines how constraints are represented but not how they should be checked. To understand how to check constraints, a description property (*schema:description*) is provided along with the "described at URL" (P973) property, a link to a page describing the constraint in natural language. The vast majority of Wikidata constraint types have such a description page (28 out of 32), as our running example *item-requires-statement*[19]. The pages present descriptions in natural language and provide some statements as examples of compliance/violation, however we argue that this is not enough due to the interpretive subjectivity present in texts: as we will see in the later discussion, we could find and document several cases of textual ambiguity, leaving room for different interpretations and thus implementations of respective constraint checks.

An example of subjectivity can be seen in the use of the Wikidata class and property hierarchy. The type constraint (Q21503250) allows to restrict the type of the subject of a triple. In terms of first-order logic, if we consider the triple *head of government(United States, Joe Biden)*, the subject "United States" (Q30) of a "head of government" (P6) statement must be an "instance of" (P31) a specific class (e.g. "government" (Q7188) or "government agency" (Q327333)). The type constraint page states that subclass relationships represented by "subclass of" (P279) must be considered when testing the constraint. Therefore, "Prime Minister's Office" (Q7243282) can have a head of government because *instance of(Prime Minister's Office, cabinet department)* and *subclass of(cabinet department, government agency)* exist. Yet, the same is not automatically true for subproperties (P1647) of P6, i.e. property constraints do not propagate to subproperties in WD.

Along these lines, the *allowed qualifiers constraint* (Q21510851) states that when using a property, only a limited set of properties can be used *as qualifiers* through the reification mechanism. Using the same example, if *head of government(United States, Joe Biden)* exists, then it is possible to add qualifiers to specify the period of time when the information is valid (e.g. "start time" (P580) and "end time" (P582)). For instance, "Banská Bystrica" (Q144983),

[19]https://www.wikidata.org/wiki/Help:Property_constraints_portal/Item

a municipality in Slovakia, had until 2006 "Ján Králik" (Q13421338) as head of government; the qualifier "dissolved, abolished or demolished date" (P576) is used to determine the end of that period. Although *subproperty of("dissolved, abolished or demolished date", end time)* exists, *head of government(Banská Bystrica, Ján Králik)* is considered a violation by the system that reports violations on Wikidata.

Although "subproperty of" (P1647) is considered the pendant of "subclass of" (P279) when it comes to the hierarchy of properties, the page that describes allowed qualifiers constraint does not mention the use of the hierarchy of subproperties, neither allowing nor prohibiting its use.

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

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

Notably, though, neither the entity types which are part of the Wikibase base ontology (`wikibase:`-prefixed nor the `wd:`-prefixed types appear consistently in Wikidata's RDF export on the WD endpoint across entity types: that is, while for instance on the one hand the query

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

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

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

returns 0 results. While our SHACL formalisation (cf. Footnote 11) somewhat ignores this potential issue, a "best-effort" SPARQL query trying to approximate the respective constraint check is referenced in the first line of our summary Table 6.

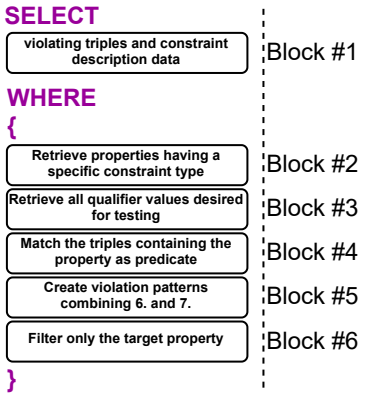
In summary, all of these examples and issues should motivate the following **disclaimer**: the SPARQL queries proposed in this paper were created from the available descriptions and aim to reduce the margin of interpretation in dealing with WD constraints. As such, all the SPARQL formalizations presented in the following and detailed in Table 6 reflect our best-effort *interpretation* of the respective natural language definitions. Yet, the goal and scope of our work is to contribute to these interpretations in an unambiguous, declarative manner.

4.2. Expressing and validating WD property constraints in SPARQL

We generalize the approach sketched in the end of Section 3.3: based on the more complex example in Fig. 2, let us illustrate the overall structure of our constraint validation approach using SPARQL queries, which were designed to capture the semantics of WD property constraint types. Recall that our goal is to both provide SPARQL queries as a formal interpretation and at the same time enabling retrieval of "whitnesses" for inconsistent data.

Fig. 6(a) presents a generic structure followed by each SPARQL query proposed in this paper. We generalize queries into different "blocks", such that each block can contain multiple triple patterns as exemplified in Fig. 6(b), which fulfill different functions. Fig. 6(b) represents the concrete query for the *item-requires-statement* constraint: for this constraint type a required property and its required value(s) needs to be checked. Each block of the query structure of Fig. 6(b) is detailed as follows.

Block #1, i.e., the SELECT clause, represents the information to be returned by the query describing the violation; usually it is composed by the claim containing the property that is violating the constraint (Fig. 6(b) line 2), plus extra information about the triple or about the constraint – in our case, the property missing for the subject – (line 3), and finally, we also return the constraint status or reason for deprecation (line 4), if given.



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

```
1  SELECT DISTINCT                                                                  Block #1
2  ?s ?wdt_property  ?o
3  ?wdt_required_property
4  ?constraint_status ?deprecated_reason
5  {                                                                                Block #2
6    ?property p:P2302 ?statement.
7    ?statement ps:P2302 wd:Q21503247. #item-requires-statement
8
9    ?statement pq:P2306 ?required_property.                                        Block #3
10   ?statement pq:P2305 ?has_required_value.
11   ?required_property wikibase:directClaim ?wdt_required_property.
12   OPTIONAL {?statement pq:P2316 ?constraint_status}
13   OPTIONAL {?statement pq:P2341 ?deprecated_reason}
14
15   ?s ?wdt_property ?o.                                                           Block #4
16   ?property wikibase:directClaim ?wdt_property.
17
18   FILTER NOT EXISTS {                                                            Block #5
19     ?s ?wdt_required_property ?any_required_value.
20     ?statement pq:P2305 ?any_required_value.
21   }
22   FILTER NOT EXISTS {?statement pq:P2303 ?s.}
23
24   FILTER (?wdt_property = wdt:P1469)                                             Block #6
25 }
```

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

Fig. 6. SPARQL queries general template with exemplification

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

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

Block #4 matches the actual statements that will need to be checked for constraint verification (line 15): as we can see in this example, we need to navigate between the different property namespaces described in Fig. 1 above before matching subject and object (line 16).

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

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

We have encoded all 32 constraint types (some of which in separate queries for different variations) in SPARQL queries, following similar patterns corresponding to **Block#1–Block#6** 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 (which contain examples for checking particular properties per constraint type) can be found in Table 6 – as shortcut-links to our GitHub repository (cf. Footnote 10), and executable on WD's query service. For instance, our query https://short.wu.ac.at/72ng implementing the FIFA Player ID's item-requires-statement constraint returned **190 violations**, as opposed to the 192 on WD's database report page (cf. Footnote 18) at the time of writing. In our formalization we divide the item-requires statement constraint checking into two queries, one including required properties and another including required properties and values. The second query https://short.wu.ac.at/rmba retrieves the remaining **2** violations. While both checks could be combined in one UNION query, we prefer this design with multiple queries in case of different possible constraint variations, cf. the respective lines of Table 6 pointing to multiple queries.

Again, we note that apart from WD itself, there are an increasing number of other Wikibase instances listed in the Wikibase Registry[20] that partially re-use WD's property constraints: obviously, for Wikibase instances that declare constraints using the WD property constraints model, it should likewise be possible to check constraint violations on Wikibase instances using our SPARQL templates. For instance, The EU Knowledge Graph[21] is a Wikibase knowledge graph containing information about institutions of the European Union, countries, projects financed by the EU and their beneficiaries [27], which imports WD's property constraints. As an example, we used our SPARQL template to search for *format constraint* violations on this KG, with our query https://short.wu.ac.at/xmsm returning 472 violations at the time of writing.

5. Experiments

In order to verify our approach against the WD database reports, we designed an experiment to evaluate the semantics of our SPARQL queries comparing the violations obtained by our queries with the violations published in the WD Database reports (c.f. Footnote 5). Unlike DBpedia, where a version of the KG is pragmatically generated and made available every three months[22], Wikidata's dynamic approach causes the KG to be constantly updated with new statements. This approach makes the comparison between strategies to capture violations difficult because of the uncertainty of the KG's state at the time violations are collected. This makes checking and comparing *all* constraint violations is infeasible, since collecting them via the SPARQL endpoint takes to long to keep in sync, even disregarding (unfortunately increasingly common) timeouts on WD's SPARQL query endpoint.

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

1. We identified the top 5 most violated constraint types from the Wikidata's violation statistics table on December 16, 2022: One-of (Q21510859), item-requires-statement (Q21503247), Single value constraints (Q19474404 and Q52060874), required qualifier (Q21510856), and value-requires-statement (Q21510864).
2. We ranked the associated properties in descending order of number of violations for each of these constraint types.
3. We executed our SPARQL queries to collect the violations of 5 different properties for the 5 constraint types, totaling 25 violation sets available in our github repository [23].
4. The ad-hoc violation checking system used in WD takes about a day to execute and publish results, thus our queries were executed one day before the data was available. Consequently, we extract the set of corresponding violations published by the WD portal referring to the same properties on the next day. For instance, the "FIFA player ID" (P1469) property specific violations are made available in the WD report page[24].
5. Finally, we structured and compared the violations reported by the WD Database reports with the violations retrieved by the SPARQL queries on the WD endpoint.

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

[20] https://wikibase-registry.wmflabs.org/wiki/Main_Page
[21] https://linkedopendata.eu/wiki/The_EU_Knowledge_Graph
[22] https://www.dbpedia.org/resources/snapshot-release/
[23] https://github.com/nicolasferranti/wikidata-constraints-formalization/tree/main/experiment_data
[24] https://www.wikidata.org/wiki/Wikidata:Database_reports/Constraint_violations/P1469

6. Results

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

6.1. One-of Constraint

The first results concern the **One-of constraint** and they are available in Table 1. Three properties were skipped for this constraint type due to timeout issues on the WD SPARQL endpoint.

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

Table 1
One of constraint violations

For **"genre" (P136)**, the reason why 11 violations were not found by our approach is that our queries (cf. explanation of **Block#4** above) are checking constraints on the exported *truthy* statements in the WD RDF dump,[25] i.e., the *non-deprecated* statements in the wdt: namespace, which include the *PreferredRank* values (when there is one) when there are multiple values, and the *NormalRank* values when there is no PreferredRank. Due to this, 8 out of 11 subjects were found by our approach but associated with different values, for instance, for "Johann Sebastian Bach" (Q1339) we found "Baroque music" (Q8361) as the PreferredRank and a violation, while for the WD Reports "western classical music" (Q9730), a NormalRank object is the violation. In **"applies to part" (P518)**, the 147 violations identified by both approaches contain the property P518 used as main property (wdt: prefix), and these violations are also highlighted by the WD page of each entity in the UI,[26] e.g. "elder abuse" (Q427883) has three different values for P518, all of them violations to the constraint. Notably, the remaining reported violations not identified by our query are also not considered violations in the WD pages in the UI, because the property P518 is used as a *qualifier* in these cases: for instance, "Catalan Countries" (Q234963) has as "country" (P17) "Italy" (Q38), however this "applies to part (P518)" "Alghero" (Q166282). Currently, this is not displayed as a violation by the WD pages, but the Database reports is testing the constraint also for qualifiers. In our case, it would be necessary to adapt the triple pattern "`?s wdt:P518 ?o`", to alternatively also test the qualifier namespace (pq:), if one wants to test the query also for qualifiers.

The 4 violations not captured by our approach for **"distribution format" (P437)** are due to the same reason highlighted for P136. We identified these 4 subjects but since the property has multiple values and there is one PreferredRank value, our SPARQL query computes the violation for the value in the PreferredRank. For instance, "The Simpsons" (Q886) has as "distribution format" (P437) "video on demand" (Q723685) and "terrestrial television"

[25] https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format#Truthy_statements
[26] Note: it is necessary to be logged in in Wikidata to see violations in the UI)

(Q175122) but "terrestrial television" (Q175122) is marked as the PreferredRank. While our query captures a violation for "terrestrial television" (Q175122), WD Reports captures a violation for "video on demand" (Q723685). Again, for **"sport (P641)"** the 4 violations are regarding multiple values with different Ranks. For instance, "Tove Alexandersson" (Q113200) has "sport" (P641) "orienteering" (Q29358), "ski orienteering" (Q428242), "skyrunning" (Q3962667), and "ski mountaineering" (Q1075998). "orienteering" (Q29358) is the PreferredRank. While our query captures "orienteering" (Q29358) as the violation, the WD reports captures the other values. The analysis of the one-of constraint for property P641 reveals that an improvement option would be: instead of listing every possible sports type, the constraint could refer to a superclass of sports and include hierarchical class inference when testing the constraint. Therefore, all the orienteering types would be under the same subclass of sport and it would not be necessary to add a new type of sport to the constraint once this type is connected to a superclass.

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

6.2. Item-requires-statement constraint

For *Item requires statement* constraints (IRS), which are very common, ten properties were skipped due to the timeout in the WD SPARQL endpoint. For this constraint type, the values displayed in the VA column of Table 2 can exceed 5001 because the same property can have multiple IRS instances; moreover, such instances can request the existence of only one property or a pair <property,value>. Therefore, the WD database reports presents a list of violations for each instance of IRS, which we summed up in the tables to report the respective **VA** numbers.

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

Table 2

Item requires statement constraint violations

In Table 2, note that for the top 3 properties (P1559, P1976, and P2539), our approach found all the available violations and some extra violations that unfortunately cannot be compared because the results available in the WD database reports are incomplete. For **"ResearcherID" (P1053)**, the 5 statements not captured by our queries were again due to deprecated values, i.e., again, WD does not match the pattern "?s wdt:P1053 ?o" when ?o is deprecated (i.e., non-truthy). Further on property P1053, WD database reports points to 4 violations on the IRS with the required property "instance of" (P31) and "value human" (Q5). Our approach identified these 4 violations and 2 more not reported by the database reports, they are "Milieu Intérieur Consortium" (Q86498220) and "Wolfgang Wagner" (Q73833983). The first one is an instance of "project" (Q170584) and the second one has "conflation" (Q14946528) as the PreferredRank value. In this case we can also notice a bad usage of the constraint, because the IRS constraint was used to restrict the type of the subject, simulating the behavior of what should actually be a *type constraint*.

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

6.3. Single-value constraint

The statistic table of the WD database reports points Single-value constraint as the third most violated constraint type, when analysing the details, we notice that this statistic takes into account Single-value (Q19474404) and Single-Best-value (Q52060874) constraints. Therefore, to perform the experiment it was necessary to use the queries designed for these two types of constraints. Eleven properties were skipped for presenting timeout in the SPARQL endpoint for at least one of the query types. The results are presented in Table 3, where unlike the previous tables, we include in columns VA and OV the total number of unique entities found, i.e., the total number without repeated entities.

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

Table 3

Single Value/Best Single Value constraints violations

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

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

The occurrence of properties from the astronomy domain, such as "type of variable star" (P881) and "surface gravity" (P7015), was expected, since the astronomy community in WD uses deprecation and different rankings to represent historical data, as also observed in [12]. Therefore, it is common to find statements with multiple values, where the higher ranked ones represent more accurate or currently accepted data by the community.

6.4. Required qualifier constraint

The required qualifier constraint has the same principle described for IRS, the same property can have multiple instances of the required qualifier constraint, each one of them requiring a different property to be used as a qualifier for a given statement. For this constraint type, which again is very common, three properties were skipped due to timeout on the WD SPARQL endpoint, where again we moved to the next best properties. The results are available in Table 4, showing that the whole set of available violations (VA) was found by our SPARQL approach (OV).

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

Table 4
Required Qualifiers constraint violations

6.5. Value-requires-statement constraint

Finally, *Value-requires statement* constraints (VRS) are again similar to IRS, but instead of requiring the existence of a statement in the subject, these quite common constraints require a statement in the object. Ten overly common properties were skipped due to timeouts in the SPARQL endpoint. Two different queries were used for each property, one checking required properties and another checking pairs of required properties and values. Main results are available in Table 5.

For **"molecular function" (P680)** and **"associated cadastral district" (P10254)** the intersection is equal to the number of violations published by the WD database reports. The 8 statements not found for **"has edition or translation" (P747)** and the 18 for **"consecrator" (P1598)** are due to the existence of one PreferredRank value among multiple values. Lastly, for the most violated property, **"heritage designation" (P1435)**, the 13 statements claimed as violations by database reports and not reported by our approach fall into the same category: there are multiple values and one of them is marked as PreferredRank value, therefore the pattern "`?s wdt:P1435 ?o`" does not capture the remaining values. For instance, "Vatican City" (Q237) has as "heritage designation" (P1435) the values "UNESCO World Heritage Site" (Q9259) and "Cultural property under special protection" (Q26086651), however "UNESCO World Heritage Site" (Q9259) is marked as the PreferredRank. Our query can be easily adapted to focus on the statement nodes instead of the direct value, as illustrated in query https://short.wu.ac.at/xasn. It is only necessary to replace "`?s wdt:P1435 ?o`" by "`?s p:P1435/ps:P1435 ?o`".

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

Table 5
Value requires statement constraint violations

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

#ID	#Name	#SHACL	#PropCount	#SPARQL	#Qualifiers
Q52004125	allowed entity types	Partially	9657	Partially (https://short.wu.ac.at/pdhs)	P2303;P2305;P2316;P4680;P6607
Q21510851	allowed qualifiers	Not reasonably expressible	819	https://short.wu.ac.at/6inv	P2241;P2303;P2304;P2306;P2316;P6607
Q21514353	allowed units	Yes	509	https://short.wu.ac.at/ntff	P2303;P2305;P2316;P6607
Q54554025	citation needed	Yes	370	https://short.wu.ac.at/fty7	P2303;P2316;P6607
Q21510852	Commons link	Yes	89	https://short.wu.ac.at/4c5k	P2307;P2316
Q21502838	conflicts-with	Yes	1181	https://short.wu.ac.at/6hsf	P2303;P2304;P2305;P2306;P2316;P2916;P6607;P6824;P9729
Q25796498	contemporary	Yes	128	https://short.wu.ac.at/53m9	P2303;P2316;P6607
Q111204896	description in language	Yes	7	https://short.wu.ac.at/z759	P424;P2316
Q21510854	difference-within-range	No	9	https://short.wu.ac.at/cqrs	P2303;P2306;P2312;P2313;P2316;P4680;P6607
Q21502410	distinct-values	Partially	7462	https://short.wu.ac.at/6u73	P2303;P2304;P2316;P2916;P4155;P6607
Q21502404	format	Yes	7690	https://short.wu.ac.at/vfwt	P1793;P2241;P2303;P2316;P2916;P4680;P6607
Q52848401	integer	Yes	183	https://short.wu.ac.at/8f39	P2303;P2316
Q21510855	inverse	Yes	125	https://short.wu.ac.at/wsz9	P2241;P2303;P2306;P2316;P4680;P6607
Q21503247	item-requires-statement	Yes	4171	https://short.wu.ac.at/rmba (only req. prop.) https://short.wu.ac.at/72ng (also req. val.)	P2241;P2303;P2304;P2305;P2306;P2316;P2916;P4680;P6607
Q108139345	label in language	Yes	762	https://short.wu.ac.at/cnfx	P2316;P424
Q55819106	lexeme requires language	Yes	172	https://short.wu.ac.at/68dk	P2305;P6607
Q55819078	lexeme requires lexical category	Yes	13	https://short.wu.ac.at/mne4	P2305
Q64006792	lexeme value requires lexical category	Yes	1	https://short.wu.ac.at/fcke	P2305
Q21510857	multi-value	Yes	36	https://short.wu.ac.at/8t2p	P2304;P2316;P6607
Q51723761	no-bounds	Yes	76	https://short.wu.ac.at/f7dw	P2303;P2316
Q52558054	none-of	Yes	115	https://short.wu.ac.at/48jf	P2303;P2304;P2305;P2316;P2916;P6104;P6607;P6824;P97
Q21510859	one-of	Yes	222	https://short.wu.ac.at/hejz	P2241;P2303;P2305;P2316;P2916;P6607
Q52712340	one-of qualifier value property	Yes	10	https://short.wu.ac.at/nj53	P2305;P2306
Q53869507	property scope	Yes	9912	https://short.wu.ac.at/pzs3 (as main value) https://short.wu.ac.at/wxth (as qualifier) https://short.wu.ac.at/scxh (as reference)	P2303;P2304;P2316;P2916;P4680;P5314;P6607
Q21510860	range	Yes	358	https://short.wu.ac.at/fyvx (for values) https://short.wu.ac.at/9ggg (for dates)	P2303;P2310;P2311;P2312;P2313;P2316;P6607
Q21510856	required qualifier	Yes	391	https://short.wu.ac.at/ym6e	P2241;P2303;P2304;P2306;P2316;P4680;P6607
Q52060874	single-best-value	No	207	https://short.wu.ac.at/ad6r	P2303;P2316;P4155;P4680;P6607
Q19474404	single-value	Partially	7254	https://short.wu.ac.at/8ds8	P2241;P2303;P2304;P2316;P2916;P4155;P4680;P6607
Q21510862	symmetric	Yes	48	https://short.wu.ac.at/7hpr	P2303;P2316
Q21503250	type	Yes	6860	https://short.wu.ac.at/v7uz (instanceOf) https://short.wu.ac.at/wvzt (subclassOf) https://short.wu.ac.at/eydg (instanceOrSubclassOf)	P2241;P2303;P2304;P2308;P2309;P2316;P4680;P6607
Q21510864	value-requires-statement	Yes	230	https://short.wu.ac.at/4nht (only req. prop.) https://short.wu.ac.at/aadh (also req. val.)	P2241;P2303;P2304;P2305;P2306;P2316;P4680;P6607
Q21510865	value-type	Yes	1077	https://short.wu.ac.at/3akc (instanceOf) https://short.wu.ac.at/xaed (subclassOf) https://short.wu.ac.at/7pv9 (instanceOrSubclass)	P2303;P2304;P2308;P2309;P2316;P2916;P6607

Table 6

Wikidata property constraints set

7. Related Work

In this paper, we have target the formalization of WD property constraints using SPARQL and SHACL. Various other prior works have already sought solutions to improve the data quality of KGs by defining constraints using these two formalisms or by means of ontological reasoning.

Data restrictions within WD are also discussed by the community and implemented through further projects using other pre-established technologies. For instance, the *Wikidata Schemas* project[27] relies on the Shape Expressions language (ShEx) [28]. 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. Unlike SHACL, which was designed to provide a constraint language for RDF, ShEx was intended to be a grammar or schema for RDF graphs [9]. As opposed to property constraints, the *Schemas Project* is focused on defining entity (Wikidata concepts) restrictions. At the time of writing, WD has more than 200k classes[28] and the *Schemas* project counts with 375 ShEx schemas[29]. This is a ratio of approximately only 0.2% of all classes having a defined ShEx schema. On the other hand, 99% of properties (10788[30] out of 10812[31]) are restricted by at least one property constraint type. These numbers illustrate that the impact of understanding the semantics of property constraints is – at the moment – more significant than ShEx schemas. A separate analysis would be required for analysing the *Schemas project* in more detail: taking for instance the entity schema *E10*[32] for the class *human* as an example, using ShEx as rather descriptive than prescriptive constraint language [29], some properties are highlighted as "desired" properties only in the schema. For instance in the mentioned schema E10, the property *mother* (P25) is defined with a Kleene star

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

meaning that the absence of a "mother" does not lead to inconsistencies, which indicates that the objective of such schema is rather to assist in the "design" of classes than constraint checking in the strict sense. Also, although there are some ShEx to SHACL conversion tools[33], additional challenges related to recursion impose further limitations on the process [9], as a lot of the theoretical work on SHACL either restricts or excludes recursive shapes (e.g. [23, 24]: the mentioned entity schema E10, for instance restricts fathers, mothers, sibling, etc. also (recursively) to be humans. The SHACL shapes we used in the present work are all non-recursive, and we did not encounter WD property constraints that would require recursive shapes at this point.

Erxleben et al. [10] exploit properties describing taxonomic relations in WD to extract an OWL ontology from WD. The authors also propose the extraction of schematic information from property constraints and discuss their expressibility in terms of OWL axioms. However, whereas we focus herein concretely on covering all property constraints as a means to find possible violations in the data, Erxleben and colleagues rather stress the value of their corresponding OWL ontology as a (declarative) high-level description of the data, without claiming complete coverage of all WD property constraints.

Martin and Patel-Schneider [13] discuss the representation of WD property constraints through multi-attributed relational structures (MARS), as a logical framework for WD. Constraints are represented in MARS using extended multi-attributed predicate logic (eMAPL), providing a logical characterization for constraints. Despite covering 26 different constraint types, to the best of our knowledge, the authors have not performed experiments to evaluate the accuracy of the proposed formalization, nor its efficiency, and do not discuss implementability: in fact the theoretical framework partially skips over subtleties of checking certain constraints in practice. As an example, the translation of *allowed entity types constraints* in the extended version of [13], assumes that entity types in Wikidata can be checked via simple instance-of type checking: our SPARQL query shows that this is not the case in practice for all

[27] https://www.wikidata.org/wiki/Wikidata:WikiProject_Schemas
[28] https://short.wu.ac.at/p2zn
[29] https://www.wikidata.org/wiki/Wikidata:Database_reports/EntitySchema_directory
[30] https://short.wu.ac.at/g2ya, last accessed 13 February 2023
[31] https://www.wikidata.org/wiki/Wikidata:List_of_properties, last accessed 13 February 2023
[32] https://www.wikidata.org/wiki/EntitySchema:E10
[33] https://rdfshape.weso.es/shexConvert

entity types as they are differently represented in the actual WD RDF dump.[34] Our work – focusing on the *practical* implementability of property constraints in SPARQL – complements such theoretical approaches.

Abián et al. [20] propose a definition of contemporary constraint that was indeed later adopted by WD property constraints. Shenoy et al. [12] present a quality analysis of WD focusing on correctness, checking for weak statements under three main indicators: constraint violation, community agreement, and deprecation. The premise is that a statement receives a low quality score when it violates some constraint, highlighting the importance of constraints for KG refinement. Boneva et al. [30] 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 WD property constraints.

Apart from works specifically on constraint for WD, in [16] the authors systematically identify errors in DBpedia, using the DOLCE ontology as background knowledge to find inconsistencies in the assertional axioms. They feed target information extracted from DBpedia and linked to the DOLCE ontology into a reasoner checking for inconsistencies. Before, Bischof et al. [17] already highlighted logical inconsistencies in DBpedia which can be detected using OWL QL, rewritten to SPARQL 1.1 property paths – not unlike our general approach.

7.1. SHACL and SPARQL Benchmarks

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

As for SHACL, real-world performance benchmarks still seem to be rare. Schaffenrath et al. [31] have presented a benchmark consisting of 58 SHACL shapes over a graph with 1M N-quads sample from a tourism knowledge graph, evaluated with different graph databases, emphasizing that "larger data exceeded [...] available resources". The shapes we present are on the one hand targeting an (orders of magnitude) larger dataset, but on the other hand can also be evaluated locally on a per entity level, thus providing a benchmark of quite different nature. Also, the evolving nature of WD makes a dynamic/evolving benchmark that can be evaluated/scaled along the natural evoluation and growth of WD itself. Next, [32] present a synthetic SHACL benchmark derived from the famous LUBM ontology benchmark, while also emphasizing the current lack of real-world benchmarks for SHACL.

Closest but also orthogonal in focus to our own work is a recent paper by Rabbani et al. [33], which focuses on the orthogonal problem of automatically *extracting* shapes (representable as SHACL) from large KGs such as WD in a data-driven manner, rather than on the formalisation of community-driven constraints as we do.

Finally, apart from serving as a basis for novel benchmarks for SHACL, particularly our SPARQL formalisation extends, and in our opinion complements, the existing landscape of real-world SPARQL benchmarks. Indeed, the – to the best of our knowledge – only benchmark for WD, WDbench [14] covers a significantly different kind of Wikidata queries than we do. WDbench is a benchmark extracted from WD query logs, focusing on queries that time out on the regular WD query endpoint, but restricted to queries on truthy statements only, that is for instance not covering queries on qualifiers. Our queries on the contrary, by definition all relate to querying qualifiers and, as such they require the whole WD graph and cannot be answered on the truthy statements alone. Yet, similar to the WDbench queries, many of the queries we present, particularly on very common properties, suffer from timeouts, as our experiments confirm. Thus, while the approach we present shows in principle feasible, it calls for novel more scalable approaches to efficiently solve such SPARQL queries that currently time out. We hope, as the queries we focus on typically only affect local contexts of entities and properties, they could hopefully be solved, e.g., by clever modularisation and partitioning techniques.

8. Conclusions and Future Work

In the present work, we have formalized all 32 different property constraint types of WD using SPARQL and discussed ways to encode them with W3C's standard recommendation mechanism for formalizing constraints over

[34] cf. https://short.wu.ac.at/pdhs

RDF Knowledge Graphs, SHACL. This study made it possible to clarify to which extent SHACL can represent community-defined 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-availability of practical SHACL performance benchmarks has already been emphasized by [32], where we believe our work could be a significant step forward towards leveraging WD as a large benchmark dataset for SHACL validators.

Other results we presented include clarifications of heretofore uncertain issues, such as the representability of permitted entities and exceptions in WD property constraints within SHACL [12]. 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).

As we could show, all these issues could be addressed when using SPARQL to formalize and validate constraints, where all 32 constraints could in principle be formalized, with the caveat that the *allowed entity types* constraint (Q52004125) does not permit to be entirely validated on the current, incomplete, Wikidata RDF dump. In this context, as a partially negative result, one of the main limitations of the work was the increasing performance limitations of WD's query endpoint, which call for more scalable query interfaces and bespoke evaluation mechanisms. On the positive side, this limitations gives rise to further research considering property constraint violation detection as a performance SPARQL benchmark as such. As a first next step in this directions, we aim at comparing our results from the WD SPARQL endpoint with a local installation, comparing different graph databases or lightweight query approaches such as HDT [34] to support a queryable version of WD constraints checks independent of the SPARQL endpoint, which – for reasons of immediate comparison with the current WD violation reports – was beyond our scope of the present paper.

WD property constraints are dynamically evolving and maintained by the community, as shown by new constraint types such as *Citation needed* (Q54554025), a constraint type not even yet reported by WD's official constraint reporting tool, cf. Footnote 17. We believe that our formalization and operationalization of property constraints in a declarative way, using SPARQL, establishes a mutual relationship with the WD community: analyzing the formalization helps to enrich the way constraints are modeled and vice versa clarifies their semantic interpretation, as opposed to the current, partially ambiguous natural language definitions, verifiable only through the partially disclosed WD database reports. We further hope that this article stimulates discussions in the community to enrich the representation of constraints that still might have subjective interpretations.

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

References

[1] H. Paulheim, Knowledge graph refinement: A survey of approaches and evaluation methods, *Semantic web* **8**(3) (2017), 489–508.

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

[3] A. Zimmermann, N. Lopes, A. Polleres and U. Straccia, A General Framework for Representing, Reasoning and Querying with Annotated Semantic Web Data, *Journal of Web Semantics* **12** (2012), 72–95. http://polleres.net/publications/zimm-etal-2012-JWS.pdf.

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

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

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

[7] P. Hitzler, M. Krötzsch, P.F.P.-S. Bijan Parsia and S. Rudolph, OWL 2 Web Ontology Language Primer (Second Edition), 2012, http://www.w3.org/TR/owl-primer/.

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

[9] J.E.L. Gayo, E. Prud'Hommeaux, I. Boneva and D. Kontokostas, Validating RDF data, *Synthesis Lectures on Semantic Web: Theory and Technology* **7**(1) (2017), 1–328.

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

[11] A. Piscopo and E. Simperl, Who models the world? Collaborative ontology creation and user roles in Wikidata, *Proceedings of the ACM on Human-Computer Interaction* **2**(CSCW) (2018), 1–18.
[12] K. Shenoy, F. Ilievski, D. Garijo, D. Schwabe and P. Szekely, A Study of the Quality of Wikidata, *Journal of Web Semantics* **72** (2022), 100679.
[13] D. Martin and P.F. Patel-Schneider, Wikidata Constraints on MARS., in: *Wikidata@ ISWC*, 2020.
[14] R. Angles, C.B. Aranda, A. Hogan, C. Rojas and D. Vrgoč, WDBench: A Wikidata Graph Query Benchmark, in: *The Semantic Web – ISWC 2022*, U. Sattler, A. Hogan, M. Keet, V. Presutti, J.P.A. Almeida, H. Takeda, P. Monnin, G. Pirrò and C. d'Amato, eds, Springer International Publishing, 2022, pp. 714–731. ISBN 978-3-031-19433-7.
[15] S. Tiwari, D. Gaurav, A. Srivastava, C. Rai and K. Abhishek, A preliminary study of knowledge graphs and their construction, in: *Emerging Technologies in Data Mining and Information Security*, Springer, 2021, pp. 11–20.
[16] H. Paulheim and A. Gangemi, Serving DBpedia with DOLCE–more than just adding a cherry on top, in: *International semantic web conference*, Springer, 2015, pp. 180–196.
[17] S. Bischof, M. Krötzsch, A. Polleres and S. Rudolph, Schema-agnostic query rewriting in SPARQL 1.1, in: *International semantic web conference*, Springer, 2014, pp. 584–600.
[18] A. Haller, A. Polleres, D. Dobriy, N. Ferranti and S. Rodríguez Méndez, An Analysis of Links in Wikidata, in: *ESWC 2022-19th European Semantic Web Conference*, 2022.
[19] D. Hernández, A. Hogan and M. Krötzsch, Reifying RDF: What Works Well With Wikidata?, in: *Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems*, CEUR Workshop Proceedings, Vol. 1457, CEUR-WS.org, 2015, pp. 32–47. http://ceur-ws.org/Vol-1457/SSWS2015_paper3.pdf.
[20] D. Abián, J. Bernad and 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.
[21] S. Ahmetaj, R. David, M. Ortiz, A. Polleres, B. Shehu and M. Šimkus, Reasoning about Explanations for Non-validation in SHACL, in: *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, Vol. 18, 2021, pp. 12–21.
[22] R. David, S. Ahmetaj, M. Šimkus and A. Polleres, Repairing SHACL Constraint Violations using Answer Set Programming, in: *Proceedings of the 21st International Semantic Web Conference (ISWC 2022)*, LNCS, Vol. 13489, Springer, 2022, pp. 375–391.
[23] J. Corman, J.L. Reutter and O. Savkovic, Semantics and Validation of Recursive SHACL, in: *The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part I*, D. Vrandecic, K. Bontcheva, M.C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L. Kaffee and E. Simperl, eds, Lecture Notes in Computer Science, Vol. 11136, Springer, 2018, pp. 318–336. doi:10.1007/978-3-030-00671-6_19.
[24] J. Corman, F. Florenzano, J.L. Reutter and O. Savkovic, SHACL2SPARQL: Validating a SPARQL Endpoint against Recursive SHACL Constraints, in: *Proceedings of the ISWC 2019 Satellite Tracks (Posters & Demonstrations, Industry, and Outrageous Ideas) co-located with 18th International Semantic Web Conference (ISWC 2019), Auckland, New Zealand, October 26-30, 2019*, M.C. Suárez-Figueroa, G. Cheng, A.L. Gentile, C. Guéret, C.M. Keet and A. Bernstein, eds, CEUR Workshop Proceedings, Vol. 2456, CEUR-WS.org, 2019, pp. 165–168. http://ceur-ws.org/Vol-2456/paper43.pdf.
[25] P.F. Patel-Schneider and D. Martin, EXISTStential Aspects of SPARQL, in: *Proceedings of the ISWC 2016 Posters & Demonstrations Track co-located with 15th International Semantic Web Conference (ISWC 2016), Kobe, Japan, October 19, 2016*, T. Kawamura and H. Paulheim, eds, CEUR Workshop Proceedings, Vol. 1690, CEUR-WS.org, 2016. http://ceur-ws.org/Vol-1690/paper72.pdf.
[26] D. Hernández, C. Gutierrez and R. Angles, The Problem of Correlation and Substitution in SPARQL - Extended Version, Technical Report, CoRR, 2018. http://arxiv.org/abs/1801.04387.
[27] D. Diefenbach, M.D. Wilde and S. Alipio, Wikibase as an infrastructure for knowledge graphs: The eu knowledge graph, in: *The Semantic Web–ISWC 2021: 20th International Semantic Web Conference, ISWC 2021, Virtual Event, October 24–28, 2021, Proceedings 20*, Springer, 2021, pp. 631–647.
[28] S. Staworko, I. Boneva, J.E.L. Gayo, S. Hym, E.G. Prud'Hommeaux and H. Solbrig, Complexity and Expressiveness of ShEx for RDF, in: *18th International Conference on Database Theory (ICDT 2015)*, 2015.
[29] A. Bonifati, P. Furniss, A. Green, R. Harmer, E. Oshurko and H. Voigt, Schema validation and evolution for graph databases, in: *Conceptual Modeling: 38th International Conference, ER 2019, Salvador, Brazil, November 4–7, 2019, Proceedings 38*, Springer, 2019, pp. 448–456.
[30] I. Boneva, J. Dusart, D.F. Alvarez and J.E.L. Gayo, Shape designer for ShEx and SHACL constraints, in: *ISWC 2019-18th International Semantic Web Conference*, 2019.
[31] R. Schaffenrath, D. Proksch, M. Kopp, I. Albasini, O. Panasiuk and A. Fensel, Benchmark for Performance Evaluation of SHACL Implementations in Graph Databases, in: *Rules and Reasoning*, V. Gutiérrez-Basulto, T. Kliegr, A. Soylu, M. Giese and D. Roman, eds, Springer International Publishing, Cham, 2020, pp. 82–96. ISBN 978-3-030-57977-7.
[32] M. Figuera, P.D. Rohde and M. Vidal, Trav-SHACL: Efficiently Validating Networks of SHACL Constraints, in: *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, J. Leskovec, M. Grobelnik, M. Najork, J. Tang and L. Zia, eds, ACM / IW3C2, 2021, pp. 3337–3348. doi:10.1145/3442381.3449877.
[33] K. Rabbani, M. Lissandrini and K. Hose, Extraction of Validating Shapes from very large Knowledge Graphs, *Proceedings of the VLDB Endowement* **16**(5) (2023).
[34] J.D. Fernández, M.A. Martınez-Prieto, C. Gutiérrez, A. Polleres and M. Arias, Binary RDF Representation for Publication and Exchange (HDT), *Journal of Web Semantics* **19**(2) (2013). doi:https://dl.acm.org/doi/10.1016/j.websem.2013.01.002.