Semantic Web -1 (2024) 1–48
DOI 10.3233/SW-243611
IOS Press
CORRECTED PROOF

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

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

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

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

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

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

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

1. Introduction

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

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

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

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

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

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

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

[1] https://www.wikidata.org/wiki/Wikidata:Statistics, as from January 2023.

[2] https://short.wu.ac.at/7t66, last accessed 13 February 2023.

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

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

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

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

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

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

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

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

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

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

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

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

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

2. Background

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

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

2.1. Data modeling in Wikidata

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

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

$$(subject, predicate, object)$$

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

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

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

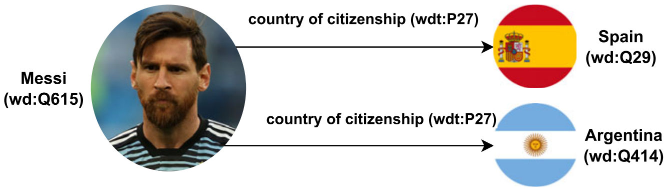

Messi
(wd:Q615)
country of citizenship (wdt:P27)
Spain
(wd:Q29)
country of citizenship (wdt:P27)
Argentina
(wd:Q414)

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

Table 1

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

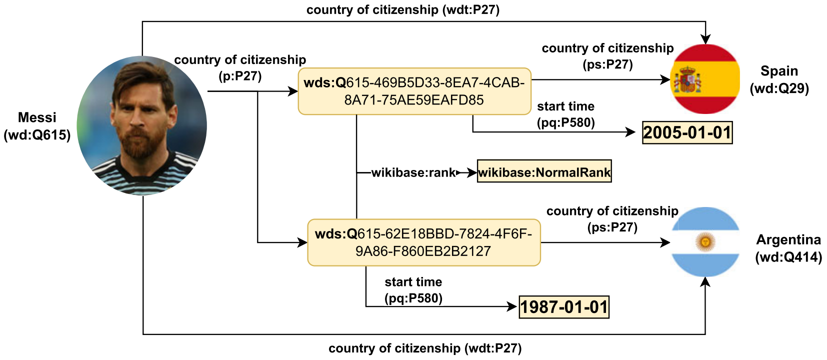

country of citizenship (wdt:P27)
country of citizenship (p:P27)
Messi (wd:Q615)
wds:Q615-469B5D33-8EA7-4CAB-8A71-75AE59EAFD85
country of citizenship (ps:P27)
Spain (wd:Q29)
start time (pq:P580)
2005-01-01
wikibase:rank
wikibase:NormalRank
wds:Q615-62E18BBD-7824-4F6F-9A86-F860EB2B2127
country of citizenship (ps:P27)
Argentina (wd:Q414)
start time (pq:P580)
1987-01-01
country of citizenship (wdt:P27)

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

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

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

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

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

`wikibase:rank`, `wikibase:NormalRank`),

. . .}

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

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

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

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

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

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

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

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

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

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

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

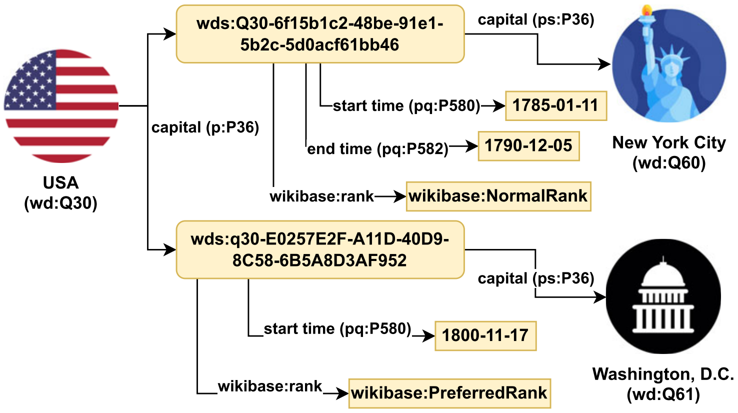


Fig. 3. A subgraph showing claims about two *capitals* (P36) of the *USA* (Q30) and their start and end times. According to a *single-value* constraint on P36, multiple values are allowed as long as they have different values for *start time* (P580) (and other so-called "separators", not shown here).

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

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

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

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

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

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

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

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

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

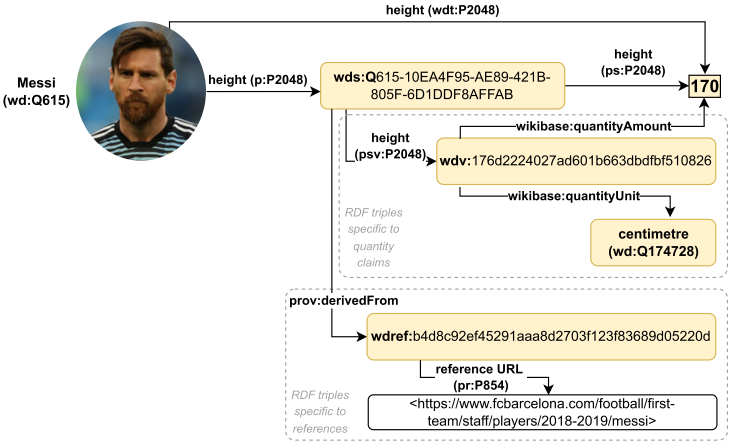

height (wdt:P2048)
Messi
(wd:Q615)
height (p:P2048)
wds:Q615-10EA4F95-AE89-421B-805F-6D1DDF8AFFAB
height
(ps:P2048)
170
wikibase:quantityAmount
height
(psv:P2048)
wdv:176d2224027ad601b663dbdfbf510826
wikibase:quantityUnit
RDF triples specific to quantity claims
centimetre
(wd:Q174728)
prov:derivedFrom
wdref:b4d8c92ef45291aaa8d2703f123f83689d05220d
reference URL
(pr:P854)
<https://www.fcbarcelona.com/football/first-team/staff/players/2018-2019/messi>
RDF triples specific to references

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

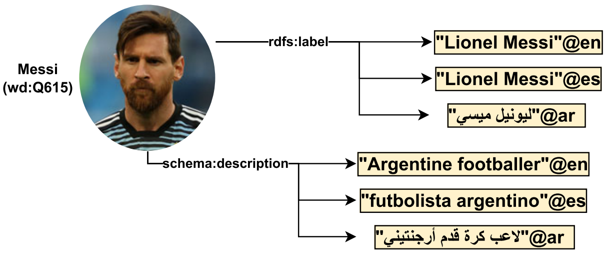


Fig. 5. A subgraph illustrating additional RDF triples for representing (multi-lingual) labels and descriptions of Wikidata entities, leveraging RDF's language-tagged literals.

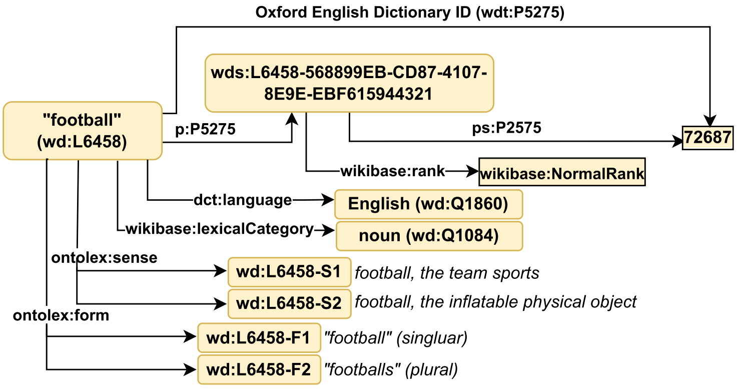


Fig. 6. A subgraph about the English noun "football", including normal claims, but also Wikidata-specific additional vocabulary to talk about languages.

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

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

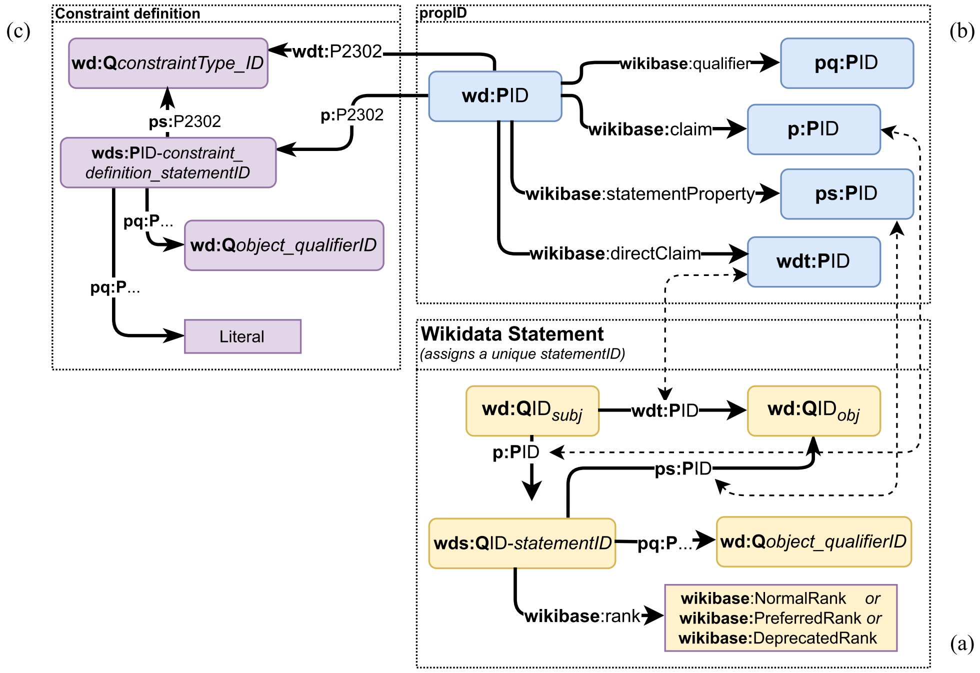


Fig. 7. The Wikidata meta-model in RDF and its namespaces usage illustrated for (a) statements and claims, (b) properties, and (c) property constraint definitions. Dashed lines represent equivalent entities. Figures 2 and 3 illustrate concrete instantiations of the "Wikidata Statement" block (a), while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

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

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

2.2. *Property constraints modeling by example*

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

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

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

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

Table 2

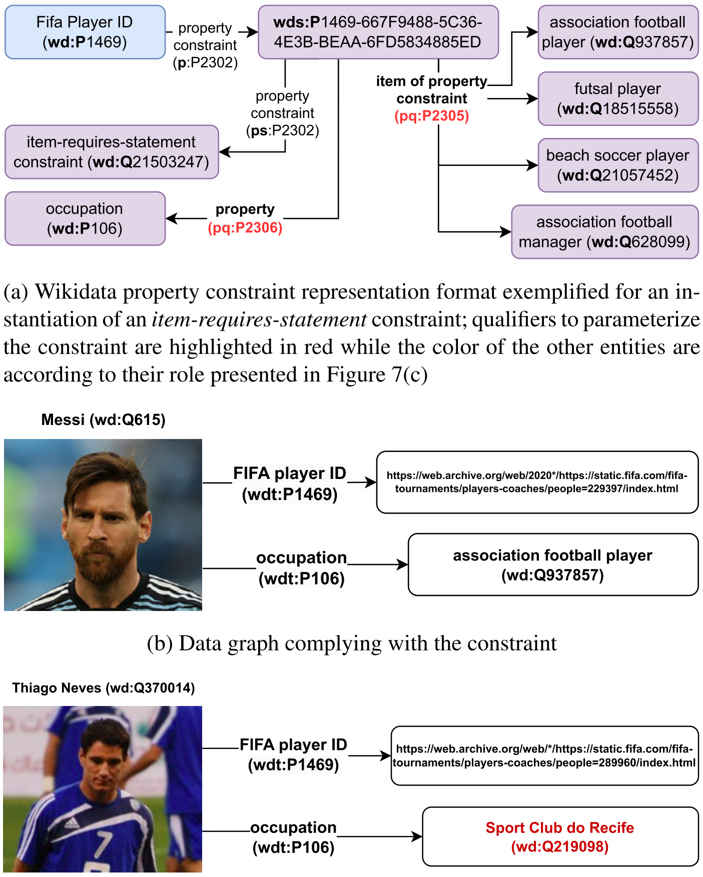
Wikidata property constraints types: incl. information about their usage in constraint definitions, information about whether and how we could express them in SHACL-Core and SPARQL, as well as which qualifiers they use (verified on the status at the writing of the paper using a variation of this query: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

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

Table 2

(Continued)

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


Fifa Player ID (wd:P1469)
property constraint (p:P2302)
wds:P1469-667F9488-5C36-4E3B-BEAA-6FD5834885ED
association football player (wd:Q937857)
futsal player (wd:Q18515558)
beach soccer player (wd:Q21057452)
association football manager (wd:Q628099)
item of property constraint (pq:P2305)
property constraint (ps:P2302)
item-requires-statement constraint (wd:Q21503247)
occupation (wd:P106)
property (pq:P2306)
Messi (wd:Q615)
FIFA player ID (wdt:P1469)
https://web.archive.org/web/2020*/https://static.fifa.com/fifa-tournaments/players-coaches/people=229397/index.html
occupation (wdt:P106)
association football player (wd:Q937857)
Thiago Neves (wd:Q370014)
FIFA player ID (wdt:P1469)
https://web.archive.org/web/*/https://static.fifa.com/fifa-tournaments/players-coaches/people=289960/index.html
occupation (wdt:P106)
Sport Club do Recife (wd:Q219098)

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

(b) Data graph complying with the constraint

(c) Data graph not complying with the constraint

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

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

Example 6. Fig. 8a illustrates how these qualifiers are concretely instantiated for an IRS constraint the property $P =$ P1469, *FIFA player ID*: this instance of IRS constraint states that if an item has a *FIFA player ID* (P1469), this very same item should also (i) have an *occupation* (P106), (ii) with one of the following four items: *association football player* (Q937857), *futsal player* (Q18515558), *beach soccer player* (Q21057452), or *association football manager* (Q628099); (i)+(ii) are defined through the resp. qualifiers P2306+P2305 specific to the particular constraint type. That is, firstly, the triple

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

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

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

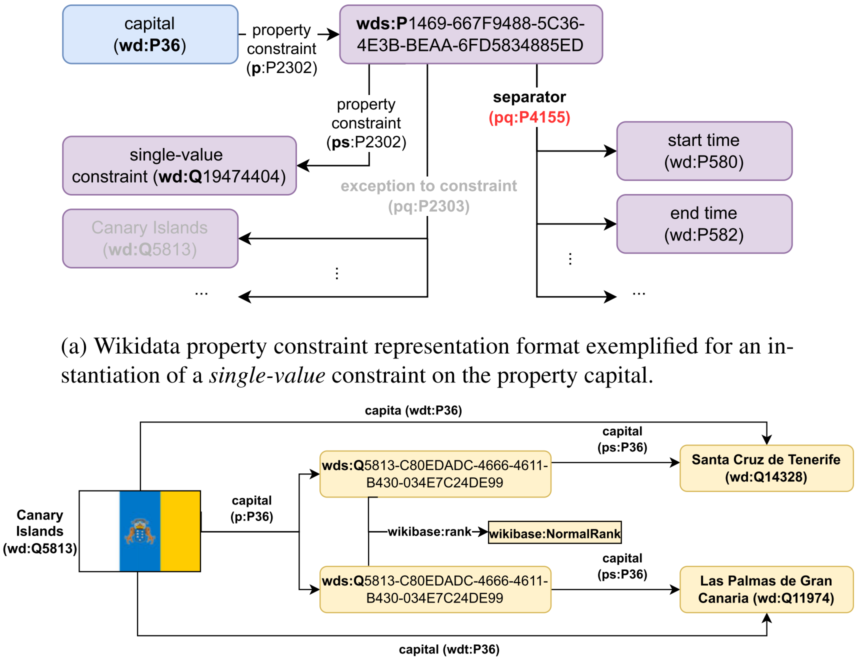
```
(wds:P1469-667F9488-5C36-4E3B-BEAA-6FD5834885ED,pq:P2306,wd:P106)
```

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

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

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

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



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

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

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

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

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

2.3. *Constraint qualifiers*

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

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

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

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

2.3.1. *Core constraint qualifiers*

- ***item of property constraint* (P2305)**: Lists items expected as values, depending on the constraint type, of either

  ∗ Case 1: P itself, or
  ∗ Case 2: another path P' from the subject of P

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

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

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

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

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

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

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

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

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

2.3.2. *Constraint exception qualifiers*

- ***exception to the constraint* (P2303)**: this qualifier is used to mark a set of Wikidata entities as an exception to the constraint, i.e., that should not be tested for violation, despite using predicate P.
  Usage Example: cf. Fig. 9 above; In principle this qualifier is applicable to all constraint types, i.e., all constraints could list such exceptions, as we can see in Table 2 (last column) though, not all qualifiers have exceptions (i.e., we did not find usage of this qualifier in all constraint types).
- ***reason for deprecated rank* (P2241)**: typically in combination with a *wikibase:DeprecatedRank* on the P2302-linked constraint-definition-statement,[20] this qualifier used to express, that a constraint is deprecated and indicates the particular deprecation reason. Possible reasons include *obsolete* (Q107356532) and *constraint provides suggestions for manual input* (Q99460987). Similar to P2303 for noting exceptions, this qualifier is – in principle – applicable to any constraints, and we indeed see it used in Table 2 with many constraint types.
  Usage Examples: Previously, entities having *place of birth* (P19) should also have *sex or gender*, but currently, this restriction is deprecated, with reason Q99460987 ("constraint provides suggestions for manual input"): constraints of this nature should not be checked or enforced, since the constraint should be rather interpreted as a suggestion than enforcing/restricting certain values.

2.3.3. *Descriptive constraint qualifiers*

- ***group by* (P2304)**: a qualifier used to group violations in specific groups of Wikidata's reports.
  Usage Example: for instance, for the property *population* (P1082) a group by qualifier specifies that violations for the *allowed qualifier* (Q21510851) constraints should be grouped by *country* (P17), as visible in in Wikidata's database reports.[21]
- ***constraint status* (P2316)**: represent the severity degree of a constraint as *mandatory* (Q21502408), *suggested* (Q62026391), or normal (no declared constraint status).[22]
  Usage Example: For example, the *subject type* constraint on property *academic degree* (P512) states that it is *mandatory*, that all the subjects be an *instance of human*, *fictional character*, or *person*. On the contrary, the above-mentioned *subject type* constraint on *date of birth* (P569), does not have an explicit status (and therefore should be considered a "normal" constraint).
- ***constraint clarification* (P6607)** and ***syntax clarification* (P2916)**: both represent text descriptions for constraints. *Constraint Clarification* was originally created to describe the purpose of the constraint for a property. We note that, while we see it used in the context of most constraint *types* (cf. Table 2) – at the time of writing –

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

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

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

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

only 865 out of 72,339 constraint definitions in total actually had such a clarification.[23] It is also used to describe suggested repairs in textual form. *Syntax Clarification* provides a textual description of the regex syntax of a value and is to be used in combination with the above-mentioned *format as a regular expression* (P1793) qualifier as documentation.

Usage Examples: For instance, a *none-of constraint* (Q52558054) for *country of citizenship* (P27) lists several "Pokemón regions", such as *Kanto* (Q1657833), that should not be used, where the additional P6607 clarifies that "It's not always clear in Pokémon canon if regions are part of a larger country or are countries in and of themselves."

– ***replacement property* (P6824) and *replacement value* (P9729)**: Both these qualifiers represent suggestions to fix inconsistencies, rather than to test/verify them; while *replacement property* suggests the replacement of constrained property P by another property, *replacement value* recommends using a specific value (a Wikidata concept or literal) instead of the current one.

  Usage Example: for instance, the property *country* (P17) uses a *conflicts-with constraint*, in combination with *item of property constraint* (P2306) qualifiers listing values

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

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

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

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

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

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

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

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

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

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

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

2.4.1. (Non-)consideration of subclasses and subproperties

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

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

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

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

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

This potential issue is, by the way, not restricted to IRS constraints, cf. Footnote 25 on p. 17, which illustrates a similar questionable example for the potential non-consideration of subclass-inferencing in the context of a *conflicts-with* constraint.

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

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

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

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

2.4.2. Interpreting separators

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

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

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

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

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

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

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

2.4.3. Handling exceptions, deprecated, or suggested constraints

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

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

2.4.4. Differences in Wikidata RDF serialization

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

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

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

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

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

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

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

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

returns 0 results on the Wikidata SPARQL endpoint.

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

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

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

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

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

3. Expressing Wikidata constraints with SHACL

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

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

3.1. SHACL validation

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

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

[31] For instance, the SPARQL endpoint returned over 13M instances of `ontolex:Form` at the time of writing, cf. https://w.wiki/8hku.

[32] https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format#WDQS_data_differences

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

(a)

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

(b)

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

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

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

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

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

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

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

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

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

3.2. Mapping Wikidata constraints to SHACL-Core

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

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

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

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

```
:P734_ConflictsWithShape
    a sh:NodeShape ;
    sh:targetSubjectsOf wdt:P734;
    sh:property [
        sh:path wdt:P1560 ;
        sh:maxCount 0 ].
```

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

- ***item-of-property-constraint* (P2305) and *property* (P2305)**: We note that Example 9 clarifies that it is possible to encode allowed values in SHACL, something that was considered uncertain in [55]: indeed, amongst the qualifier parameters in constraints, discussed in all values of the *item-of-property-constraint* (P2305) qualifier, can be turned into an `sh:in` expression (lines 7–8 in Fig. 10a); the respective *property* (P2306) qualifier is captured by an `sh:path`, where the respective path existence requirement for $P' =$ implied by the IRS constraint can be implemented by means of the `sh:qualifiedMinCount 1` restriction on this path (lines 5 and 10 in Fig. 10a). Obviously, arbitrary instantiations of IRS constraints can be turned into SHACL-Core shapes analogously, by collecting the resp. P2305 and P2306 qualifiers as parameters.

  As noted in Section 2.3.1, the P2306 qualifier may also in the context of other constraint types, refer to more generic paths P', likewise expressible through `sh:path` which indeed can represent arbitrary regular paths, for instance, the following shape for representing the afore-mentioned *allowed units* constraint on *height* (P2048), illustrates this in line 5:

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

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

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

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

```
:P11625_FormatConstraintShape a sh:NodeShape;
    sh:targetSubjectsOf wdt:P11625;
    sh:property [
        sh:path wdt:P11625 ;
        sh:pattern "[a-z\d_.-]+" ;
    ].
```

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

```
:P244_LabelInLanguageShape
        a sh:NodeShape ;
        sh:targetSubjectsOf wdt:P244 ;
        sh:property [
            sh:path rdfs:label ;
            sh:qualifiedValueShape [
                sh:languageIn ( "en" ) ;
            ] ;
            sh:qualifiedMinCount 1 ;
        ] .
```

- ***separator* (P4155)**: To the best of our knowledge, there is no direct equivalent SHACL-Core component to model composite keys, i.e., we cannot directly express respective constraint types such as *single-value* constraints, in case they use the separators; while a simplified version of a SHACL-Core shape to express a *single-value constraint* without separators is illustrated in Fig. 13a below, let us defer the discussion about expressing separators to later for now.
- ***class* (P2308) and *relation* (P2309)**: in general, the SHACL component *sh:class* can be used to check the type of an item, also including hierarchical inference to check instances of subclasses. Yet, the subclasses mechanism is based on *rdfs:subClassOf* and *rdf:type* and requires an adaptation to work with WD's *wdt:P279* and *wdt:P31*: instead, in our formalization we use a `sh:path` together with `sh:hasValue` or `sh:in` for instance to encode *subject type* constraints (where, if you recall, we need to check the path *wdt:P31/wdt:P279** for testing with an *instanceOf* relation), as illustrated by the following example:

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

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

```
:P1086_RangeShape a sh:NodeShape;
    sh:targetSubjectsOf wdt:P1086; # atomic number
    sh:property [
        sh:path wdt:P1086 ; # instance of
        sh:minInclusive 0 ;
        sh:maxInclusive 155 ;
    ].
```

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

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

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

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

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

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

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

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

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

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

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

3.3. Tool to automatically convert Wikidata constraints to SHACL

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

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

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

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

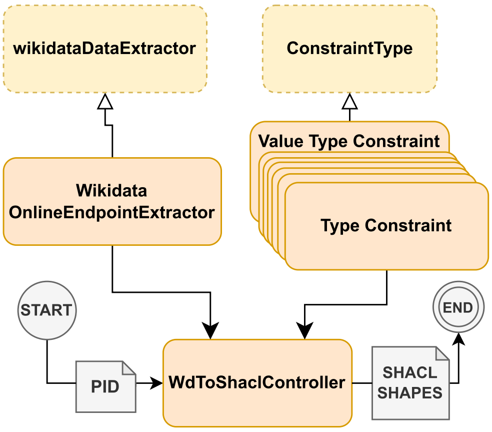


Fig. 11. Wikidata to SHACL architecture. Dashed lines represent abstract classes.

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

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

3.4. Limitations of SHACL-Core for checking Wikidata constraints

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.5. Beyond SHACL-Core: SHACL-SPARQL

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.6. Towards SPARQL

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4. Operationalizing Wikidata constraints with SPARQL

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

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

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

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

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

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

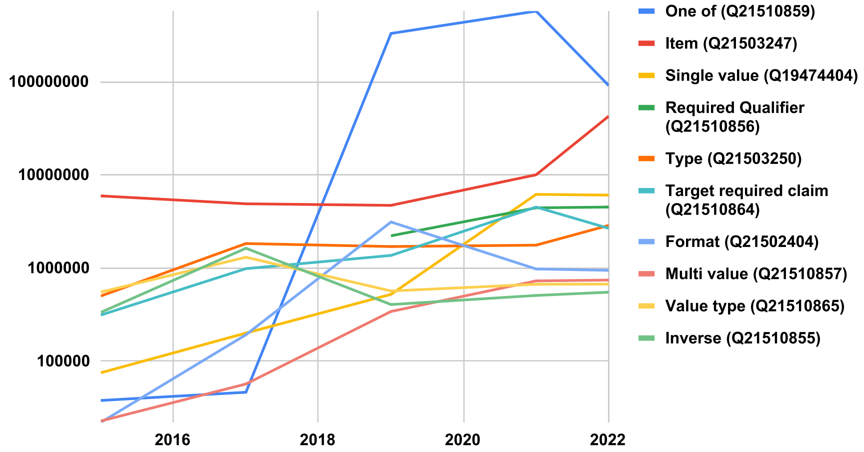
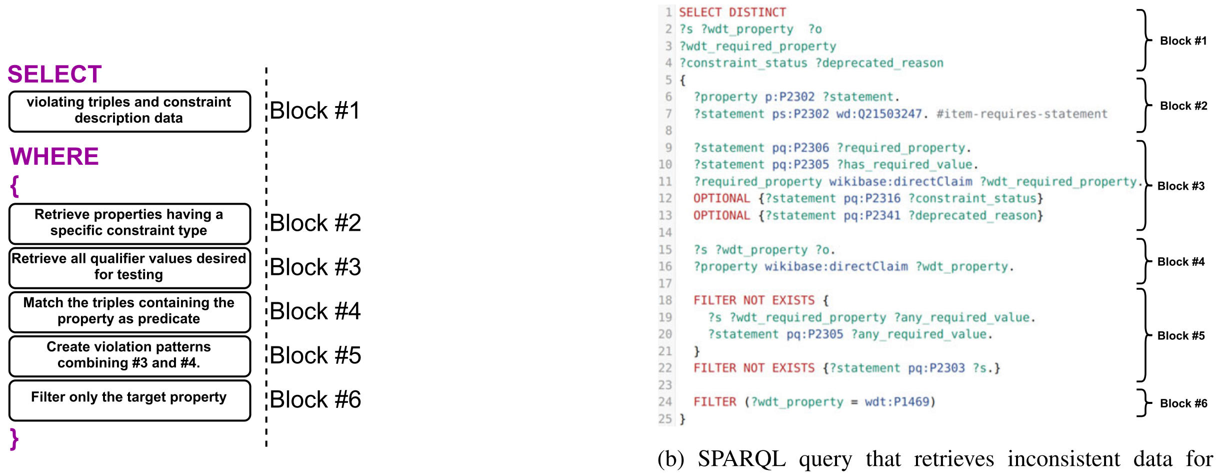


Fig. 14. #violations for top 10 most violated constraint types (logarithmic scale).


SELECT
violating triples and constraint description data
Block #1
WHERE
{
Retrieve properties having a specific constraint type
Block #2
Retrieve all qualifier values desired for testing
Block #3
Match the triples containing the property as predicate
Block #4
Create violation patterns combining #3 and #4.
Block #5
Filter only the target property
Block #6
}

```
SELECT DISTINCT
?s ?wdt_property  ?o
?wdt_required_property
?constraint_status ?deprecated_reason
{
  ?property p:P2302 ?statement.
  ?statement ps:P2302 wd:Q21503247. #item-requires-statement

  ?statement pq:P2306 ?required_property.
  ?statement pq:P2305 ?has_required_value.
  ?required_property wikibase:directClaim ?wdt_required_property.
  OPTIONAL {?statement pq:P2316 ?constraint_status}
  OPTIONAL {?statement pq:P2341 ?deprecated_reason}

  ?s ?wdt_property ?o.
  ?property wikibase:directClaim ?wdt_property.

  FILTER NOT EXISTS {
    ?s ?wdt_required_property ?any_required_value.
    ?statement pq:P2305 ?any_required_value.
  }
  FILTER NOT EXISTS {?statement pq:P2303 ?s.}

  FILTER (?wdt_property = wdt:P1469)
}
```

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

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

Fig. 15. SPARQL queries general template with exemplification.

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

4.1. *Expressing and validating Wikidata constraints in SPARQL*

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

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

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

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

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

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

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

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

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

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

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

[44] https://wikibase-registry.wmflabs.org/wiki/Main_Page

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

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

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

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

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

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

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

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

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

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

```
1   SELECT DISTINCT # Block 1
2       ?subject ?wdt_property_to_check_differ
3       ?min_val ?max_val
4       ?date_1 ?date_2
5       (COALESCE( (ABS(xsd:date(?date_1) - xsd:date(?date_2)))/365  , "null" ) as ?diff)
6   WHERE {
7       ?wd_property p:P2302 ?statement . # Block 2
8       ?statement ps:P2302 wd:Q21510854 . ## difference-within-range
9
10      ?statement pq:P2313 ?min_val . # Block 3
11      ?statement pq:P2312 ?max_val .
12      ?statement pq:P2306 ?property_to_check_differ .
13      ?property_to_check_differ wikibase:directClaim ?wdt_property_to_check_differ.
14      ?property_to_check_differ wikibase:claim ?p_property_to_check_differ.
15
16      ?subject ?wdt_property ?date_1; # Block 4
17               ?wdt_property_to_check_differ ?date_2.
18      ?wd_property wikibase:directClaim ?wdt_property .
19
20      FILTER ( # Block 5
21          COALESCE(   (ABS(xsd:date(?date_1) - xsd:date(?date_2)))/365  , "null" ) < ?min_val ||
22          COALESCE(   (ABS(xsd:date(?date_1) - xsd:date(?date_2)))/365  , "null" ) > ?max_val
23      )
24
25      FILTER (?wdt_property = wdt:P4602) #Block 6
26  }
```

5. Experiments

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

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

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

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

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

6. Results

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

6.1. One-of constraint

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

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

[47] https://github.com/nicolasferranti/wikidata-constraints-formalization/tree/main/experiment_data

[48] https://www.wikidata.org/wiki/Wikidata:Database_reports/Constraint_violations/P1469

[49] https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format#Truthy_statements

Table 3
One of constraint violations

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

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

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

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

6.2. Item-requires-statement constraint

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

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

Table 4

Item requires statement constraint violations

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

Table 5

Single value/best single value constraints violations

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

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

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

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

6.3. *Single-value constraint*

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

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

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

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

6.4. Required qualifier constraint

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

6.5. Value-requires-statement constraint

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

Table 6

Required qualifiers constraint violations

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

Table 7

Value requires statement constraint violations

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

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

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

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

7. Related work

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

7.1. Constraint languages for graph data

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

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

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

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

{ FOCUS wdt:P1469 _}@:P1469_ItemRequiresStatementShape

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

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

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

[51] https://spinrdf.org/

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

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

    :P734_conflictWithConstraint
        a sh:NodeShape ;
        sh:targetSubjectsOf wdt:P734;
        sh:property [
            sh:path wdt:P1560 ;
            sh:maxCount 0 ] .
```

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

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

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

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

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

7.2. Constraints in Wikidata

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

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

[53] https://short.wu.ac.at/p2zn

[54] https://www.wikidata.org/wiki/Wikidata:Database_reports/EntitySchema_directory

[55] https://short.wu.ac.at/g2ya, last accessed 13 February 2023.

[56] https://www.wikidata.org/wiki/Wikidata:List_of_properties, last accessed 13 February 2023.

[57] https://www.wikidata.org/wiki/EntitySchema:E10

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

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

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

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

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

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

7.3. SHACL and SPARQL benchmarks

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

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

[58] https://rdfshape.weso.es/shexConvert

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

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

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

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

8. Conclusions and future work

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

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

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

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

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

Acknowledgements

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

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[33] O. Hartig, Foundations of RDF⋆ and SPARQL⋆ (an alternative approach to statement-level metadata in RDF), in: *Proceedings of the 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web*, Montevideo, Uruguay, June 7–9, 2017, J.L. Reutter and D. Srivastava, eds, CEUR Workshop Proceedings, Vol. 1912, CEUR-WS.org, 2017, https://ceur-ws.org/Vol-1912/paper12.pdf.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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