SoCK: SHACL on Completeness Knowledge

Muhammad Jilham Luthfi, Fariz Darari and Amanda Carrisa Ashardian

Faculty of Computer Science, Universitas Indonesia, Depok, West Java, Indonesia

Abstract

The proliferation of applications based on knowledge graphs (KGs) in the recent years has created increasing demands for high-quality KGs. Completeness is an important quality aspect concerning the breadth, depth, and scope of data in KGs. In that light, describing and validating completeness over KGs have become a must go in order to make an informed decision whether or not to use (parts of) KGs. In this paper, we propose SoCK (short for SHACL on Completeness Knowledge), a pattern-oriented framework to support the creation and validation of knowledge about completeness in KGs. The framework relies on SHACL, a W3C recommendation for validating KGs against a collection of constraints. In SoCK, we first offer a number of patterns capturing how completeness requirements are typically expressed in a high-level way. Such completeness patterns can then be instantiated in various manners over different KG domains. These instantiations result in SHACL shapes that can be validated against KGs to provide a completeness profile of the KGs. As a proof-of-concept, we implement and demonstrate the SoCK framework as a Python library, creating over 360k SHACL shapes for real-world KGs (in our case, DBpedia and Wikidata) based on the aforementioned completeness patterns. We also develop a web app to serve as an information point for anything about SoCK, available at https://sock.cs.ui.ac.id/.

Keywords

SHACL, Completeness, Patterns, Shapes, Validation, DBpedia, Wikidata

1. Introduction

The current massive development of knowledge graphs (KGs) may cause various problems related to data quality [1]. The quality of data can affect application quality and is related to problems in publishing and using data [2]. An aspect of quality is completeness of information. Completeness measures how much information is contained in a dataset [3]. It is an aspect of data quality related to the breadth, depth, and scope of information contained in data [4]. The aspect of completeness is one of the most important ones in measuring data quality and can indirectly affect other aspects, such as data accuracy and consistency [5]. As for open KGs (e.g., DBpedia and Wikidata), their collaborative nature causes the diversity of data to be higher and that data quality, especially the aspect of completeness, is of particular concern [6].

Figure 1 shows two Wikidata entities of type hotel with different level of completeness in that Beverly Hills Hotel is missing the hotel rating information as opposed to that of Hotel Indonesia (as of July 10, 2022). Nevertheless, both hotels are still complete with respect to the information of: instance of, country, and review score. Such a case indicates that the quality of applications using hotel data on Wikidata may suffer from data incompleteness depending on whether hotel rating information is required in the applications or not.

WOP2022: 13th Workshop on Ontology Design and Patterns, October 23-24, 2022, Hangzhou, China

muhammad.jilham@ui.ac.id (M. J. Luthfi); fariz@ui.ac.id (F. Darari); amanda.carrisa@ui.ac.id (A. C. Ashardian)

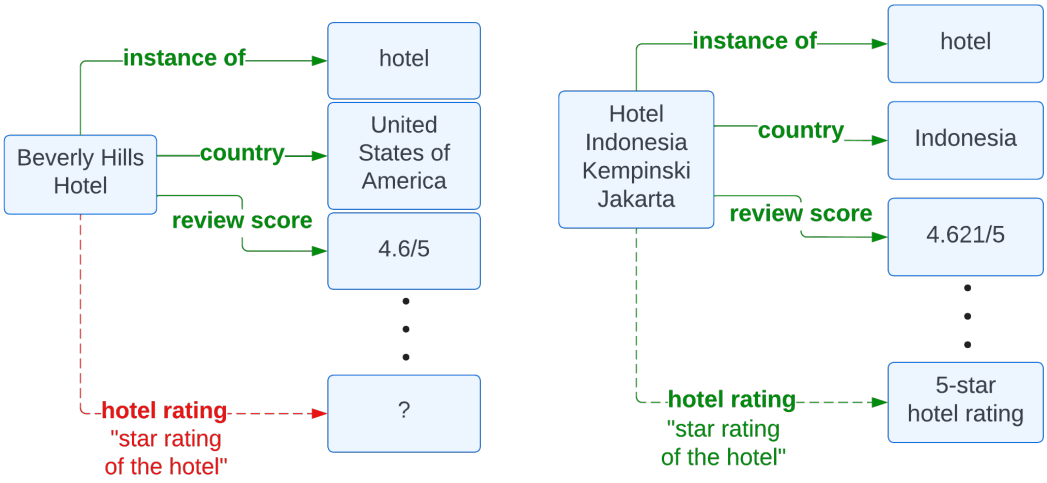

Figure 1: Data on Wikidata about (a) Beverly Hills Hotel and (b) Hotel Indonesia Kempinski Jakarta

One way to be better informed about completeness in KGs is by means of data validation. In 2017, W3C introduced a standard for validating KG data called Shapes Constraint Language (SHACL) [7]. This allows the higher need for consumption of good quality data through a validation process. SHACL validates data by applying a set of conditions that are combined into a "shape" and expressed in an RDF graph. Nevertheless, to the best of our knowledge there has been no research that specifically and systematically prioritizes SHACL as a basis for checking the quality of KG data on the aspect of completeness.

Validating data completeness in a KG could be done by collecting similar cases of completeness to form a completeness pattern. Such a pattern can then be reused and adapted to various domains in different KGs. An ontology design pattern (ODP) is a modeling solution related to repeated problems in ontologies and KGs [8]. Our study aims to fill the gap from previous studies in developing a pattern-oriented solution based on SHACL for checking KG completeness. Particularly, we focus on four issues that revolve around the problem of completeness patterns: (*i*) identification of completeness patterns; (*ii*) instantiation of completeness patterns into SHACL shapes; (*iii*) development of a (Python) library to support our completeness validation process and a web app to provide information points; and (*iv*) evaluation of completeness over real-world KGs, that is, DBpedia and Wikidata.

2. Preliminaries

Data Completeness. Data quality provides an outlook to the fitness for use by data consumers [4]. Poor data quality can pose substantial risks to decision making in organizations. Beyond accuracy, data quality aspects include relevancy, timeliness, & completeness. Data completeness concerns the breadth, depth, and scope of data w.r.t. the task at hand [4].

In the context of KGs, the quality of completeness can be classified into seven types [5]. The first two are schema completeness, which is the degree to which properties of classes are sufficiently represented, and property completeness, which concerns the completeness of property values (e.g., Albert Einstein was married twice). The third type is population completeness, which checks the coverage of real-world objects as stored in a KG. The fourth one is interlinking completeness, referring to the degree to which entities in different KGs are interlinked. The last three types are: currency completeness, examining how property values are well represented over time; metadata completeness, observing if sufficient metadata is available; and labeling completeness, which has to do with the existence of human- and machine-readable labels.

SHACL. Shapes Constraint Language (SHACL) is a W3C recommendation to describe constraints and validate KGs against those constraints [7]. Such constraints are provided as SHACL shapes, which themselves can be written as an RDF graph [9]. In SHACL, RDF graphs representing shapes are called "shapes graphs" and that these shapes graphs can then be employed to validate RDF graphs (= "data graphs"). SHACL offers some advantages in that SHACL is high-level, declarative, concise, and yet feature-rich [10].

SHACL comes in two flavors: SHACL Core and SHACL-SPARQL. The former captures features frequently needed for the representation of shapes and constraints, whereas the latter extends the former by adding advanced features of constraints based on SPARQL, a query language for RDF KGs. SHACL Core supports a variety of basic constraint components, such as value types, cardinalities, and string filters. On the other hand, SHACL-SPARQL provides more expressiveness due to the flexibility of SPARQL SELECT queries in defining SHACL constraints.

SHACL use cases include documentation, user interface generation, validation during RDF data production/consumption, and quality control [11]. In the context of quality control, a typical scenario for SHACL usage is that first a domain expert expresses quality requirements in natural language and then a knowledge engineer translates these requirements into SHACL shapes, to be validated against KGs of interest. Figure 2 illustrates a SHACL shape for the constraint that all instances of type person must have a name and a birth date. Validating a KG against that constraint may uncover the quality issue as to whether there exist person instances in that KG without a name or a birth date.

```
# these prefixes are used throughout this paper
@prefix  ex: <http://example.org/>.
@prefix  sh: <http://www.w3.org/ns/shacl#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

ex:PersonShape a sh:NodeShape;
  sh:targetClass ex:Person;
  sh:property [ a sh:PropertyShape;
    sh:path ex:name;
    sh:minCount 1 ];
  sh:property [ a sh:PropertyShape;
    sh:path ex:birthDate;
    sh:minCount 1 ].
```

Figure 2: A SHACL shape in Turtle for "every person must have a name and a birth date"

Patterns. Ontology design patterns (ODPs) or for simplicity, patterns, are a modeling solution to recurrent problems in ontology designs [12, 8]. Patterns capture problems commonly encountered in the development of ontologies & KGs, and make more explicit and reusable best practices in solving such problems. Benefits of using patterns include easier ontology design processes by both knowledge engineers and domain experts, as well as increased quality & interoperability of developed ontologies & KGs [13]. Several interesting ontologies and KG-based applications developed using pattern-based approaches are chess games [14], historic slave trade [15], and open data conversion to Wikidata [16].

Gangemi and Presutti [8] present a classification of patterns for ontology design. Those six families of patterns are: structural patterns, correspondence patterns, content patterns, reasoning patterns, presentation patterns, and lexico-syntactic patterns. In this work, we focus on content patterns, which encode conceptual solutions for ontology & KG (quality) modeling problems. In principle, content patterns do not depend on any specific language. Nevertheless, in our work we rely on SHACL as a reference formalism for representing reusable, practical building blocks of content patterns.

3. Completeness Patterns and Instantiations

This section introduces data completeness patterns developed based on SHACL and approaches to instantiating them.

3.1. Completeness Patterns

Issa et al. [5] synthesized previous studies that discussed the completeness aspect of KGs. Of the seven completeness types proposed, we use five types as completeness patterns. We also add one additional completeness pattern which we will see below. Each completeness pattern may have several SHACL patterns, that is, patterns that generalize from the SHACL syntax [7] with the addition of `%% VARIABLE %%` that later can be instantiated into SHACL shapes. Table 1 shows eight SHACL patterns according to the six completeness patterns.

Schema completeness pattern (SC). A completeness pattern about the extent to which entities in a KG have the required properties for a class, such as entities of the class "Human" must have the properties (among others) of name and date of birth. The definition of mandatory properties in the same class may differ depending on the specific needs for the task at hand.

Property completeness pattern (PC). A completeness pattern related to the degree to which property values of a specific property in a particular entity are present. For example, the entity Joe Biden must have four values for a "child" property. This completeness only focuses on one property with varying cardinalities among entities.

Table 1: List of SHACL patterns based on completeness patterns

Code	SHACL Pattern	Description

SC1	`%% SHAPE-NAME %%`
	`  a sh:NodeShape;`
	`  sh:targetClass %% CLASS %%;`
	`  sh:property [ a sh:PropertyShape;`
	`  sh:path %% PROPERTY-01 %%;`
	`  sh:minCount 1 ];`
	`  sh:property [ a sh:PropertyShape;`
	`  sh:path %% PROPERTY-NN %%;`
	`  sh:minCount 1 ].`
		All members of a class are required to have the mandatory properties from PROPERTY-01 to PROPERTY-NN. A SHACL shape instantiating such a pattern is exemplified in Figure 2.

PC1	`%% SHAPE-NAME %%`
	`  a sh:NodeShape;`
	`  sh:targetNode %% NODE %%;`
	`  sh:property [ a sh:PropertyShape;`
	`  sh:path %% PROPERTY %%;`
	`  sh:minCount %% COUNT %% ].`
		An entity (= node) is complete for PROPERTY whenever the number of the property values equals COUNT.

NVC1	`%% SHAPE-NAME %%`
	`  a sh:NodeShape;`
	`  sh:targetNode %% NODE %%;`
	`  sh:property [ a sh:PropertyShape;`
	`  sh:path %% PROPERTY %%;`
	`  sh:minCount 0 ].`
		An entity (= node) is complete for PROPERTY despite the absence of property values.

POC1	`%% SHAPE-NAME %%`
	`  a sh:NodeShape;`
	`  sh:targetNode %% NODE %%;`
	`  sh:property [ a sh:PropertyShape;`
	`  sh:path [`
	`  sh:inversePath %% TYPE-PROPERTY %% ];`
	`  sh:minCount %% COUNT %% ].`
		A population is complete whenever the number of its members equals COUNT. The use of sh:inversePath makes it possible to count the subjects of TYPE-PROPERTY.

LDC1	`%% SHAPE-NAME %%`
	`  a sh:NodeShape;`
	`  sh:targetClass %% CLASS %%;`
	`  sh:property [ a sh:PropertyShape;`
	`  sh:path %% LABEL-OR-DESCRIPTION-PROPERTY %%;`
	`  sh:minCount 1 ].`
		All members of a class are required to have a LABEL-OR-DESCRIPTION-PROPERTY. The pattern can be modified to specific nodes using sh:targetNode.

LDC2	`%% SHAPE-NAME %%`
	`  a sh:NodeShape;`
	`  sh:targetClass %% CLASS %%;`
	`  sh:property [ a sh:PropertyShape;`
	`  sh:path %% LABEL-OR-DESCRIPTION-PROPERTY %%;`
	`  sh:qualifiedMinCount 1;`
	`  sh:qualifiedValueShape [`
	`  sh:languageIn (%% LANGUAGE %%) ].`
		This pattern extends the pattern LDC1 by adding the LANGUAGE requirement for the given LABEL-OR-DESCRIPTION-PROPERTY.

IC1	`%% SHAPE-NAME %%`
	`  a sh:NodeShape;`
	`  sh:targetClass %% CLASS %%;`
	`  sh:property [ a sh:PropertyShape;`
	`  sh:path %% INTERLINKING-PROPERTY %%;`
	`  sh:minCount 1 ].`
		All members of a class are required to have an `INTERLINKING-PROPERTY`, which can be generic ones (e.g., `owl:sameAs`, `schema:sameAs`, and `skos:exactMatch`) or specific ones (e.g., `dbo:isbn` for DBpedia and `wdt:P214` as VIAF ID for Wikidata).

IC2	`%% SHAPE-NAME %%`
	`  a sh:NodeShape;`
	`  sh:targetClass %% CLASS %%;`
	`  sh:property [ a sh:PropertyShape;`
	`  sh:path %% INTERLINKING-PROPERTY %%;`
	`  sh:qualifiedMinCount 1;`
	`  sh:qualifiedValueShape [`
	`  sh:pattern %% NAMESPACE-URI %% ].`
		This pattern extends that of IC1 by adding a requirement that the linked entity must come from a specific `NAMESPACE-URI` (e.g., http://dbpedia.org/resource/).

No-value completeness pattern (NVC). A completeness pattern that deals with to which entities in a KG capture information about the absence of a property value. In this context, property values might be "unavailable" due to two possible reasons: the property values do not exist in the real world, or they are intentionally removed for privacy or confidentiality reasons. For example, the entity Keanu Reeves has no children. To be more precise, this pattern is a special case of the property completeness pattern. Nevertheless, the no-value completeness pattern deserves a special attention due to its peculiarity in that shapes instantiated from the no-value completeness pattern may represent negative facts (i.e., the non-existence) [17] and any KG would trivially satisfy the shape.

Population completeness pattern (POC). A completeness pattern related to the extent to which entities in the real world are present as members (or instances) of a population. For example, all provincial entities in Indonesia are members of the class "Provinces in Indonesia". In such a pattern, the membership of a population is usually expressed through typing properties like `rdf:type`, `dct:subject`, `dbo:type` (for DBpedia), and `wdt:P31` (for Wikidata).

Label & description completeness pattern (LDC). A completeness pattern related to the degree to which entities in a KG have human-readable labels and descriptions. One of the most commonly used label properties is `rdfs:label`. An alternative property is `skos:prefLabel`. In addition, the language aspect of the label can be considered in this pattern. Apart from the label, the pattern also concerns description properties, such as `rdfs:comment`, `schema:description`, and `dbo:abstract` (for DBpedia).

Interlinking completeness pattern (IC). A completeness pattern concerning the degree to which the same entity links to each other in various KGs. For example, the entity "Indonesia"

on DBpedia is linked to the same entity on Wikidata. One of the commonly used properties to define such linking is `owl:sameAs`. Other general properties with the same intention are `schema:sameAs` and `skos:exactMatch`. Furthermore, there are also properties that specifically link to particular external authorities, such as ISBN, DOI, and VIAF. It is worth noting that the pattern IC1 in Table 1 would only check the existence of interlinking properties regardless of the number of property values. We argue that attempting to impose some value counting on interlinking properties would be tricky to do since there can be an arbitrary number of KGs which may contain the entities to be linked. However, should one want to check the existence of interlinking properties to entities in some specific KG, the pattern IC2 which features namespace qualifiers can be utilized.

3.2. Completeness Pattern Instantiations

Given the above completeness patterns, a question arises as to how one may instantiate those patterns. We identify a number of such approaches: manual, spreadsheet, automatic, ontology, and statistics. We also list the advantages and disadvantages of each approach in Table 2.

Manual. In this approach, a user instantiates a completeness pattern by hand. For example, with respect to the pattern SC1, the user will selectively choose the properties of a class that must exist. Afterwards, the selected properties will be substituted into the property variables of the SHACL pattern.

Spreadsheet. This approach gathers completeness information (e.g., number of children and number of authors) into a spreadsheet. Next, a program reads the spreadsheet and generates the corresponding SHACL shapes based on the collected completeness information. Here we use this approach to instantiate the property completeness pattern.

Automatic. The automatic approach relies on data from KGs themselves to provide completeness information. Wikidata, for example, has the information of "properties for this type" (P1963) which lists properties that normally apply for a class. Furthermore, Wikidata also owns counting properties like "number of children" (P1971) which may give hints on the cardinality of property values (in this case, the property "child"). External APIs may also be leveraged to provide completeness information, such as Crossref for the number of complete authors of a scholarly article.[1]

Ontology. One may also instantiate completeness patterns by utilizing the ontology structure of a KG, such as using `rdfs:domain`.[2] Such ontological information provides properties that typically apply for a class. More specifically, referring to the statement of `P rdfs:domain C`, we can assume that property `P` associates with instances of `C`.

[1] https://www.crossref.org/documentation/retrieve-metadata/rest-api/
[2] https://www.w3.org/TR/rdf-schema/#ch_domain

Statistics. Statistical information may be useful for instantiating completeness patterns. Over a class, one may list the frequency of property usage in a class, and then take the top-N most frequent properties as the mandatory properties of the class.

Table 2: Advantages and Disadvantages of Various Completeness Instantiation Approaches

Approach	Advantages	Disadvantages
Manual	Enables a more detailed and customized making of shapes thus resulting in a completeness evaluation with high precision and quality.	Manual checking can be time consuming, requires prior knowledge about SHACL, and is prone to human errors.
Spreadsheet	Enables easier data collection with high quality.	It is time consuming and is not as flexible nor as detailed as the manual approach.
Automatic	Gives the benefit of quick and easy generations over an abundance of completeness instantiations (of some form).	Requires quality checking over generated instantiations.
Ontology	Uses terminological knowledge from ontologies made by domain experts, thus enabling the reuse of existing ontologies.	The quality of the results depends on the source of ontology being used and that a poor quality ontology would tend to produce a lower quality of instantiations.
Statistics	Data driven and does not require domain understanding.	Manual checking is still required to ensure that the captured instantiations make sense.

4. SoCK Library and Web App

In this section, we present our SoCK library for completeness instatiation and validation, as well as our SoCK web app as an information point for the SoCK framework.

4.1. SoCK Library

The SoCK library provides a Python-based implementation of instantiating and validating completeness patterns.[3] The library supports the pattern instantiation process as discussed in Subsection 3.2. We use the library to create SHACL shapes based on the aforementioned completeness patterns for real-world KGs like DBpedia and Wikidata.

The SoCK library can also be used to validate the completeness of KGs based on the instantiated patterns. The validation process requires both the data graph and (completeness) shapes graph as the input. Thus, the first step is SPARQL querying to get the corresponding data of

[3]Our library is available at https://github.com/JillyCS15/sock-validator

relevant entities and properties. Here we rely on SPARQLWrapper[4] for querying and RDFLib[5] for RDF data creation and manipulation. The library currently does not support validation over remote data graphs and hence, the data graphs to be validated have to be available locally in the same machine where the library resides. Next, the collected data graph is validated against the (completeness) shapes graph from the instantiation process as described before. The validation results in validation reports, informing the completeness profiles of KG entities. We reuse PySHACL[6] to support the validation step. The whole flow of validation process is displayed in Figure 3.

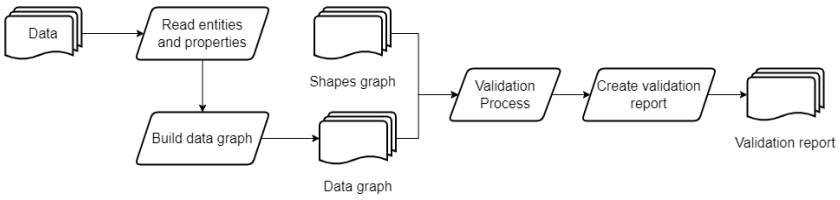

Figure 3: Validation flow in SoCK

4.2. SoCK Web App

The SoCK web app is developed to serve as an information point about the SoCK framework. The web app is available at https://sock.cs.ui.ac.id, including a demo video at https://youtu.be/FtJ3HO_6YHcq. This web app offers features for users in learning about completeness patterns and creating their instances or shapes. For example, an instance of the label & description pattern is the SHACL shape of "American Film".[7] Another example is an instance of the population completeness pattern, about "G20 Nations".[8] Further pattern instances can be viewed at https://sock.cs.ui.ac.id/instance/?page=1.

The SoCK web app uses the client-server architecture. In particular, it adapts a three-tier architecture that separates the application into three logical layers: presentation, application, and database. The presentation relies on basic web stack (i.e., HTML, CSS, and JavaScript), whereas the database makes use of SQLite. In developing the application, we employ the Django framework.

5. Case Studies: Wikidata and DBpedia

In this section, we discuss the evaluation results of our SoCK framework to real-world KGs, that is, Wikidata and DBpedia. We create SHACL shapes out of our presented completeness patterns and validate those shapes to the KGs. Overall, we successfully validate 928,310 entities

[4]https://sparqlwrapper.readthedocs.io/
[5]https://rdflib.readthedocs.io/
[6]https://pypi.org/project/pyshacl/
[7]https://sock.cs.ui.ac.id/instance/show-pattern-instance-detail/12/
[8]https://sock.cs.ui.ac.id/instance/show-pattern-instance-detail/5/

from Wikidata and DBpedia with 360,162 completeness pattern instances (= shapes). Further evaluation discussion for each completeness pattern is provided as follows.

Schema Completeness Validation. Here we instantiate the pattern SC1 via three approaches, that is, automatic, ontology, and statistics, collecting in total 1,106 SHACL shapes. Through the automatic approach using "properties for this type" (P1963), we successfully validate 469,891 Wikidata entities using 1,095 pattern instances. In the validation, we record for each entity the completeness percentage of properties listed in "properties for this type". The validation results are as follows: 35% entities are complete for 0–19% of properties, 17% entities for 20–39% of properties, 17% entities for 40–59% of properties, 11% entities for 60–79% of properties, and 20% entities for 80–100% of properties.

Next, through the ontology approach, we validate 5,000 DBpedia sample entities using five pattern instances from the class of "Hotel", "Magazine", "Museum", "University", and "Person". The validation results show that the completeness of DBpedia entities from those classes are still quite low in that 78% of all entities have no more than 10% of the corresponding class properties from the ontology approach.

As for the last schema completeness experiment, we validate 6,000 DBpedia sample entities from the class of "Country", "Person", "Activity", "Film", "Museum", and "University" using six pattern instances generated with statistics approach, where we take top-10 most frequent properties from each class. The result indicates that entities on DBpedia with property selection using a statistical approach have a good average completeness value in that almost 70% of the entities are complete for more than 65% of class properties.

Property Completeness Validation. In this validation scenario, we make 357,892 shapes of the pattern PC1 to validate the property completeness of entities. On the automatic approach, we validate 357,749 Wikidata sample entities for the completeness of the number of authors from "Scholarly Article" (Q13442814), the number of children from "Human" (Q5), the number of episodes from "Television Series Season" (Q3464665), the number of seasons from "Television Series" (Q5398426), and the number of participants from "Sports Season" (Q27020041). Then, on the spreadsheet approach, we gather completeness information about the number of children of Indonesian actors, the number of cities from Indonesian provinces, and so on. The validation results from both approaches show a relatively good completeness value, ranging from 60–70%.

No-Value Completeness Validation. We conduct experiments using this pattern through an automatic approach, instantiating the pattern NVC1 based on Wikidata. Here we take all entities having no children, as stated by the value of zero (0) for the property of "number of children" (P1971). There are in total 1,277 SHACL shapes generated for no-value completeness. The validation results are trivially 100% complete by the definition of no-value completeness.

Population Completeness Validation. We create eleven population completeness pattern instances with the code POC1. We examine on DBpedia the populations of cantons of Switzerland, continents, countries in Africa and Europe, G20 nations, Indonesian active volcanoes,

Indonesian legislative election events, Indonesian provinces, NATO nations, oceans, and Summer Olympics events. Based on the experiment results, DBpedia includes in total 92.8% of the entities from all the populations above. This shows that DBpedia has covered almost all the population entities tested.

Label & Description Completeness Validation. Using nine label & description pattern instances, we conduct validation experiments involving 69,045 entities on DBpedia and Wikidata. On DBpedia, we validate 5,000 sample entities from the class of mountain, politician, country, disease, and musical artist, for the existence of a basic label & description property according to the pattern of the code LDC1. Based on the validation results, more than 90% of the entities have a complete label and description property, such as `rdfs:label` (99.96%), `rdfs:comment` (94.96%), and `dbo:abstract` (94.96%). As for Wikidata, we validate 64,045 entities using instantiations of the pattern with the code LDC02 to check whether the entities have a label and description property with a proper language tag. We examine the entities from the class "National Hero of Indonesia", "South Korean Music Group", "American Film", and "Japanese Manga". The validation results show that over 98.72% of entities have a `rdfs:label` and 93.80% of entities have `schema:description` with a proper language tag. However, only 28.41% of the entities capture the property of alias (`skos:altLabel`) so it indicates that the `skos:altLabel` property is rather incomplete. One possible reason is there is sometimes no alias for entities of interest.

We also compare the label & description completeness of DBpedia entities that have equivalent Wikidata entities through `owl:sameAs`. This way, we can have a fair comparison between DBpedia and Wikidata. The validation checks whether entities from the class of country, mountain, film, hotel, and song, have a label & description in English according to the pattern code LDC2. Based on the validation results of 5,000 entities, DBpedia entities have completeness of 99.86% for the label property (`rdfs:label`) and 99.06% for the description property (`rdfs:comment`). Meanwhile, the corresponding Wikidata entities have completeness of 99.5% for the label property (`rdfs:label`) and 95.04% for the description property (`schema:description`). Note that Wikidata relies on `schema:description` for describing entities instead of `rdfs:comment`. From the above results, we can say that DBpedia entities have slightly higher label & description completeness than those of Wikidata.

Interlinking Completeness Validation. We experiment with interlinking completeness patterns, creating ten completeness pattern instances (i.e., five with the code IC1 and five with IC2), to validate 9,194 sample entities of DBpedia and Wikidata. On DBpedia, we check entities from the country, actor, island, museum, and hotel classes. The validation result shows that the completeness of the `owl:sameAs` property is the highest with 95.24% DBpedia entities linked to Wikidata and 72.36% linked to YAGO. Meanwhile, the completeness of the `schema:sameAs` property is only about 20.80%, and even smaller for `skos:exactMatch`, which is only 1.40%. This shows that most entities on DBpedia are not yet complete in covering external entities through `schema:sameAs` and `skos:exactMatch` properties.

On Wikidata, we examine the existence of external ID properties for entities of the class country, hotel, film, singer-songwriter, and university. The validation result shows that interlinking completeness varies: countries are mostly complete for VIAF ID (96%) and GND ID

(96%); hotels are somewhat incomplete – reaching the highest of only 25% for Agoda hotel ID;[9] films are the most complete for IMDb ID (99%); singers & songwriters are the most complete for VIAF ID (97%); and universities are also the most complete for VIAF ID (86%).

6. Related Work

A number of research studies have investigated the problem of describing and checking KG completeness. Prasojo et al. [18] introduce SP-statements, in the form of (s, p), declaring that the entity s is complete for all values of the property p for the KG of interest. Such SP-statements are practically relevant for entity-centric KGs, such as DBpedia and Wikidata. A more generic approach to describe completeness is provided in [19, 20] in that more expressive completeness statements based on arbitrary basic graph patterns (BGPs), including their consequences to query completeness, are well-studied.

In [21], Wisesa et al. develop a framework for completeness profiling. The framework analyzes the completeness of a KG based on the Class-Facet-Attribute (CFA) profiles. A class represents a set of entities with a number of facets, which allow attribute completeness to be analyzed. The framework can be used, for example, to answer the question: how does the completeness of the attributes "residence" and "eye color" fare for humans with the occupation of actor?

Now, we would like to report on several use cases of SHACL in data quality. Hammond et al. [22] discuss how SHACL can be leveraged to supervise the quality of ETL pipelines from various data sources to build the Springer Nature SciGraph. The graph describes the Springer Nature publishing world, containing data about sponsors, research projects, conferences, publications, and affiliations from across the research landscape. Cimmino et al. [23] introduce an approach to generate SHACL shapes automatically from OWL ontologies. To this end, they provide two resources: (*i*) Astrea-KG, a KG for mappings between ontology constraint patterns and SHACL constraint patterns; and (*ii*) Astrea, a tool implementing the mappings from the Astrea-KG for concrete ontologies. The tool has been experimented over various real-world ontologies to create automatically SHACL restrictions, such as `sh:datatype` and `sh:minInclusive`, in order to maintain KG quality. Pandit et al. [24] envision reusing ODP axioms to define SHACL shapes. They provide a simple case study in the context of validating the data quality of microblogs (e.g., Twitter posts). In that study, constraints and relationships within the ODP, defined using RDFS subclasses and OWL cardinality restrictions, are mapped to the corresponding SHACL shapes using `sh:class` and `sh:qualified(Max/Min)Count` conditions.

7. Conclusions

Through this paper, we present SoCK (SHACL on Completeness Knowledge), a pattern-oriented framework for describing and validating the completeness of KGs. We propose six completeness patterns based on general data quality problems related to completeness, namely those patterns of schema completeness, property completeness, no-value completeness, population completeness, label & description completeness, and interlinking completeness. From those patterns,

[9] Out of the external ID properties of VIAF ID, Booking.com numeric ID, Agoda hotel ID, Hotels.com hotel ID, and TripAdvisor ID

we manage to create over 360k instances (i.e., SHACL shapes) and validate nearly 1 million Wikidata and DBpedia entities using our SoCK Python library (https://github.com/JillyCS15/sock-validator). To disseminate our framework, we develop a SoCK web app, accessible via https://sock.cs.ui.ac.id, which provides a catalog of completeness patterns as well as instances and use cases.

There are a few things that can be done to improve the SoCK framework. Currently, five patterns are formed based on [5]. The other two patterns, namely, currency completeness and metadata completeness, are not yet investigated in this work due to the time limitations, and are left for future research. At the moment, our evaluation is limited to Wikidata and DBpedia entities. Using entities in other KGs, such as YAGO and KBpedia, or even enterprise KGs, may help further explore the general quality of KGs. The current version of our work only concerns the syntactic level of RDF graphs and not yet the semantic level. Investigating the consequences in incorporating the semantics (and inferences) of RDFS and OWL would be relevant for future directions. Working on the semantic level would also be potential in better generalizing our SHACL-based approach and aligning SHACL & ODPs for describing and validating KG completeness. Last but not least, it is also of our interest to analyze the completeness of ontologies, that is, how complete is an ontology in capturing some domain of interest (i.e., whether the classes and properties are sufficient enough).

References

[1] J. Debattista, C. Lange, S. Auer, D. Cortis, Evaluating the quality of the LOD cloud: An empirical investigation, Semantic Web 9 (2018).

[2] B. F. Lóscio, C. Burle, N. Calegari (Eds.), Data on the Web Best Practices, W3C Recommendation, 31 January 2017. https://www.w3.org/TR/dwbp/.

[3] A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann, S. Auer, Quality assessment for linked data: A survey, Semantic Web 7 (2015).

[4] R. Y. Wang, D. M. Strong, Beyond accuracy: What data quality means to data consumers, J. Manag. Inf. Syst. 12 (1996).

[5] S. Issa, O. Adekunle, F. Hamdi, S. S. Cherfi, M. Dumontier, A. Zaveri, Knowledge graph completeness: A systematic literature review, IEEE Access 9 (2021).

[6] M. Luggen, D. E. Difallah, C. Sarasua, G. Demartini, P. Cudré-Mauroux, Non-parametric Class Completeness Estimators for Collaborative Knowledge Graphs - The Case of Wikidata, in: ISWC, 2019.

[7] H. Knublauch, D. Kontokostas (Eds.), Shapes Constraint Language (SHACL), W3C Recommendation, 20 July 2017. https://www.w3.org/TR/shacl/.

[8] A. Gangemi, V. Presutti, Ontology design patterns, in: S. Staab, R. Studer (Eds.), Handbook on Ontologies, International Handbooks on Information Systems, Springer, 2009.

[9] F. Manola, E. Miller (Eds.), RDF Primer, W3C Recommendation, 10 February 2004. http://www.w3.org/TR/rdf-primer/.

[10] S. Steyskal, K. Coyle (Eds.), SHACL Use Cases and Requirements, W3C Working Group Note, 20 July 2017. https://www.w3.org/TR/shacl-ucr/.

[11] J. E. L. Gayo, E. Prud'hommeaux, I. Boneva, D. Kontokostas, Validating RDF Data, Synthesis

Lectures on the Semantic Web: Theory and Technology, Morgan & Claypool Publishers, 2017.
[12] A. Gangemi, Ontology Design Patterns for Semantic Web Content, in: ISWC, 2005.
[13] K. Hammar, Ontology Design Patterns in WebProtege, in: ISWC Posters & Demos, 2015.
[14] A. Krisnadhi, V. Rodríguez-Doncel, P. Hitzler, M. Cheatham, N. Karima, R. Amini, A. Coleman, An ontology design pattern for chess games, in: WOP, 2015.
[15] C. Shimizu, P. Hitzler, Q. Hirt, D. Rehberger, S. G. Estrecha, C. Foley, A. M. Sheill, W. Hawthorne, J. Mixter, E. Watrall, R. Carty, D. Tarr, The enslaved ontology: Peoples of the historic slave trade, JWS 63 (2020).
[16] M. Faiz, G. M. F. Wisesa, A. A. Krisnadhi, F. Darari, OD2WD: From Open Data to Wikidata through Patterns, in: WOP, 2019.
[17] F. Darari, Representing and querying negative knowledge in RDF, in: ESWC Posters and Demos, 2013.
[18] R. E. Prasojo, F. Darari, S. Razniewski, W. Nutt, Managing and Consuming Completeness Information for Wikidata Using COOL-WD, in: COLD@ISWC, 2016.
[19] F. Darari, W. Nutt, G. Pirrò, S. Razniewski, Completeness statements about RDF data sources and their use for query answering, in: ISWC, 2013.
[20] F. Darari, S. Razniewski, R. E. Prasojo, W. Nutt, Enabling fine-grained RDF data completeness assessment, in: ICWE, 2016.
[21] A. Wisesa, F. Darari, A. Krisnadhi, W. Nutt, S. Razniewski, Wikidata Completeness Profiling Using ProWD, in: K-CAP, 2019.
[22] T. Hammond, M. Pasin, E. Theodoridis, Data integration and disintegration: Managing Springer Nature SciGraph with SHACL and OWL, in: ISWC Posters & Demos, 2017.
[23] A. Cimmino, A. Fernández-Izquierdo, R. García-Castro, Astrea: Automatic Generation of SHACL Shapes from Ontologies, in: ESWC, 2020.
[24] H. J. Pandit, D. O'Sullivan, D. Lewis, Using ontology design patterns to define SHACL shapes, in: WOP, 2018.