Semantic Web 0 (0) 1–24
IOS Press

Literally Better: Analyzing and Improving the Quality of Literals

Editor(s): Amrapali Zaveri, Maastricht University, Netherlands; Dimitris Kontokostas, Universität Leipzig, Germany; Sebastian Hellmann, Universität Leipzig, Germany; Jürgen Umbrich, Wirtschaftsuniversität Wien, Austria
Solicited review(s): Heiko Paulheim, Universität Mannheim, Germany; Arthur Ryman, Ryerson University, Canada; Two anonymous reviewers

Wouter Beek, [a,*] Filip Ilievski, [a] Jeremy Debattista, [b] Stefan Schlobach, [a] and Jan Wielemaker [a]
[a] *VU University Amsterdam*
e-mail: {w.g.j.beek,f.ilievski,k.s.schlobach,j.wielemaker}@vu.nl
[b] *University of Bonn & Fraunhofer IAIS*
e-mail: debattis@iai.uni-bonn.de

Abstract. Quality is a complicated and multifarious topic in contemporary Linked Data research. The aspect of literal quality in particular has not yet been rigorously studied. Nevertheless, analyzing and improving the quality of literals is important since literals form a substantial (one in seven statements) and crucial part of the Semantic Web. Specifically, literals allow infinite value spaces to be expressed and they provide the linguistic entry point to the LOD Cloud. We present a toolchain that builds on the LOD Laundromat data cleaning and republishing infrastructure and that allows us to analyze the quality of literals on a very large scale, using a collection of quality criteria we specify in a systematic way. We illustrate the viability of our approach by lifting out two particular aspects in which the current LOD Cloud can be immediately improved by automated means: value canonization and language tagging. Since not all quality aspects can be addressed algorithmically, we also give an overview of other problems that can be used to guide future endeavors in tooling, training, and best practice formulation.

Keywords: Data Quality, Data Observatory, Quality Assessment, Quality Improvement, Linked Data

1. Introduction

In this paper we investigate the quality of literals in the Linked Open Data (LOD) Cloud. A lot of work has focused on assessing and improving the quality of Linked Data. However, the particular topic of literal quality has not yet been thoroughly addressed. The quality of literals is particularly important because (i) they provide a concise notation for large (and possibly infinite) value spaces and (ii) they allow text-based information to be integrated into the RDF data model. Also, one in seven RDF statements contains a literal as object term.[1]

Our approach consists of the following steps. We create a toolchain that allows billions of literals to be analyzed efficiently by using a stream-based approach. The toolchain is made available as Open Source code to the community. We show that the toolchain is easy to integrate into exist-

*Corresponding author, e-mail: w.g.j.beek@vu.nl

[1] A statistic derived from the LOD Laundromat data collection on 2016-05-18.

ing approaches and can be used in a sustainable manner: Firstly, important parts of the toolchain are integrated into the ClioPatria triple store and RDF library. Secondly, important parts of the toolchain are integrated into the LOD Laundromat infrastructure, and are used for clearning RDF content that is scraped from the web. Thirdly, the toolchain is used by Luzzu: a state-of-the-art Linked Data quality framework. We will use the here presented toolchain in order perform an *analysis* of the quality of literals in the LOD Cloud. Finally, we present automated procedures and concrete suggestions for *improving* the quality of literals in today's Web of Data. An important property of the here presented approach is that it can be applied to web-scale data, and ultimately to the LOD Cloud as a whole.

This paper focuses on a relatively isolated and restricted part of quality: the syntactic, semantic and linguistic aspects of literal terms. As such, it does not cover quality issues that may arise once more expressive vocabularies such as OWL are interpreted as well. Specifically, the problem of missing values may occur in this context, as may constraint violations, e.g., uniqueness constraints. These are considered to be future work.

This paper is structured as follows. Section 2 discusses related efforts on quality assessment and improvement. In Section 3 we give our motivation for performing this work. In Section 4 we define a set of quality criteria for literals. The next section describes the toolchain and its role in supporting the defined quality criteria. Section 6 reports our analysis in terms of the quality criteria defined in the previous section. In Section 7 we enumerate opportunities for improving the quality of literals based on our observations in the previous section. We implement two of those opportunities and evaluate their precision and recall. Section 8 concludes the paper and discusses further opportunities for research on literals quality.[2]

[2]This paper uses the following RDF prefixes for brevity:

```
dc: http://purl.org/dc/elements/1.1/
dct: http://purl.org/dc/terms/
dt: http://dbpedia.org/datatype/
rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs: http://www.w3.org/2000/01/rdf-schema#
sysont: http://ns.ontowiki.net/SysOnt/
xsd: http://www.w3.org/2001/XMLSchema#
```

2. Related work

Quality assessment for Linked Data is a difficult and multifarious topic. A taxonomy of problem categories for data quality has been developed by [23]. Not all categories are applicable to Linked Data quality. Firstly, due to its fluid schema and the Open World Assumption, the absence of an RDF property assertion does not imply a missing value. Secondly, because RDF does not enforce the Unique Names Assumption, the problem of value uniqueness does not arise.[3] However, most other data quality categories do apply to Linked Data and RDF literals: syntax violations, domain violations, and the problem of having multiple representations for the same value (what we will call 'Non-canonicity' in Section 4.3).

Empirical observations The large-scale aspects of Linked Data quality have been quantified in various 'LOD Observatory' studies: [2,16,17]. These studies, while focusing on Linked Data quality overall, have only included cursory analyses of quality issues for RDF literals. In [17], Hogan et al. conduct an empirical study on Linked Data conformance, assessing RDF documents against a number of Linked Data best practices and principles. They specifically cover (i) how resources are named, (ii) how data providers link their resources to external sources, (iii) how resources are described, and (iv) how resources are dereferenced.

Metadata Various metadata descriptions for expressing Linked Data quality have been proposed. In Assaf et al. [2], the authors give insight into existing metadata descriptions. This assessment checks the metadata of each RDF document for generic information, access information, ownership information and provenance information. No vocabulary for expressing literal quality metadata exists today. However, the taxonomy of literal quality in Section 4.3 may serve as a starting point

[3]For comparison, if distinct resources are described with the same value for one of the primary key attributes, then this is considered a schema violation in relational databases.

for such a vocabulary. There are already several data quality vocabularies that can be extended, e.g. [11,15]. The W3C has recently standardized a vocabulary (DQV) for expressing Linked Data quality[4].

Quality Frameworks A number of tools have been developed for assessing the quality of Linked data documents [7,12,20,22]. The authors in [20] present RDFUnit, a SPARQL based approach towards assessing the quality of Linked Data. Their framework uses SPARQL query templates to express quality metrics. The benefit of this tool is that it uses SPARQL as an extensibility framework for formulating new quality criteria. The drawback of this framework is that metrics that cannot be expressed in SPARQL, such as checking the correctness of language tags, cannot be assessed in RDFUnit. In [22], the authors make use of metadata of named graphs to assess data quality. Their framework, Sieve, allows for custom quality metrics based on an XML configuration. WIQA [7] is another quality assessment framework that enables users to create policies on indicators such as provenance. Luzzu [12] is an extensible Linked Data quality assessment framework that enables users to create their own quality metrics either procedurally, through Java classes, or declaratively, through a quality metric DSL.

Crowdsourcing Some aspects of quality are highly subjective and cannot be determined by automated means alone. In order to improve these quality aspects a human data curator is required. [1] present a crowdsourcing approach that allows data quality to be improved. Quality is not a static property of data but something that can change over time as the data gets updated. The dynamic aspects of data quality are observed in [19].

3. Motivation

This section gives motivation for analyzing and improving the quality of literals. We first explain the importance of literals overall. We then distinguish three perspectives from which the assessment and improvement of literal quality is important. We also enumerate the concrete benefits of improving literal quality.

3.1. The importance of literals

Literals have a crucial syntactic and semantic role within the Semantic Web's data model. Firstly, they introduce a concise notation for infinite value spaces. While one of the main Linked Data principles is to "use URIs as names for things" [6], URIs/IRIs are not a viable option for expressing values from infinite value spaces. The Linked Data principle of using URIs for everything is carried through to the absurd in Linked Open Numbers[5], where the authors (jokingly) present a dataset in which IRIs are minted for the natural numbers. The Linked Open Numbers dataset shows the scalability issues of assigning URIs to everything. In addition, relationships between the values of infinite value spaces are better expressed intensionally (by a shall number of of concisely defined functions) than extensionally (by explicitly asserting an infinitely large function graph). Indeed, literals allow an infinite number of values, and relations between them, to be represented through intensional definitions. For instance, floating point numbers, and the relations between them, are defined by the IEEE floating standard[6] and are implemented by operators in most programming languages.

The second main benefit of literals is that they allow linguistic or text-based information to be expressed complementary to RDF's graph-based data model. While IRIs are (also) intended to be human-readable [14], a literal can contain natural language or textual content without syntactic constraints. This allows literals to be used in order to convey human-readable information about resources. Also, in some datasets, IRIs are intentionally left opaque as the human-readability of universal identifiers may negatively affect their permanence [5]. Since the Semantic Web is a universally shared knowledge base, natural language an-

[4] See `http://www.w3.org/TR/vocab-dqv/`

[5] See `http://archive.is/QfNp`

[6] See `http://ieeexplore.ieee.org/document/4610935/`

notations are particularly important in order to ease the human processability of information in different languages.

3.2. Benefits of improved literal quality

Improving the quality of literals has (at least) the following concrete benefits for the consumption of Linked Data:

Efficient computation

If a data consumer wants to check whether two literals are identical she first has to interpret their values and apply a datatype-specific comparator operator [9]. For example, `2016-01-20T01-01-01` and `2016-01-20T02-01-01Z-01:00` denote the same date-time value, but this cannot be determined by checking for simple (character-for-character) string equality. Most defined RDF datatypes specify a canonical representation for each of their values. Canonicity allows all values in a given value space to be uniquely represented by exactly one representation. If all values in a dataset are known to be written in such a canonical way, many operations can be performed significantly faster. For example, the SPARQL query `select * { ?s ?p "2016-01-01T01-01-01Z"^^xsd:dateTime }` can be efficiently performed if the ground object term can be directly matched in the database. If values are not canonically represented, then all date-time values have to be interpreted and compared, which is significantly more costly. In general the use of canonicity makes operations such as identity checking and matching more efficient. For many datatypes the use of canonicity makes determining the relative order between literals ('smaller than' and 'larger than') more efficient as well.

Data enrichment

The availability of reliable language tags that indicate the language of a textual string is an enabler for data enrichment. Language-informed parsing and comparing of string literals is an important part of existing instance matching approaches [13]. Having language tags associated with string literals allows various notions of similarity to be defined that move beyond (plain) string similarity. This includes within-language similarity notions such as "is synonymous with" as well as between-language similarity notations such as "is translation of".

User eXperience

Knowing the language of user-oriented literals such as `rdfs:label` or `dc:description` helps to improve the User eXperience (UX) of Linked Data User Interfaces. Provided the language preference of the user is known or can be dynamically assessed, an application can prioritize language-compliant literals in the display of user-facing literals. Similar remarks apply to the approach of "value labeling" [24], in which natural language labels rather than IRIs are used to denote resources. Finally, the canonicalization of literals can result in more readable lexical forms overall (e.g., decimal `"01.0"` is canonicalized to `"1.0"`). While the data publisher may have intended to display literals in a certain serialization format, the utility of such formatting is application-specific and should therefore not be considered a good approach in Linked Data, where unanticipated reuse is a major goal.

Semantic Text Search

Tools for semantic text search over Linked Data such as LOTUS [18] allow literals, and statements in which they appear, to be retrieved based on textual relevance criteria. To enable users to obtain relevant information for their use case, these tools use retrieval metrics that are calculated based on structured data and meta-information. High-quality language-tagged literals allow more reliable relevance metrics to be calculated. For instance, 'die' is a demonstrative pronoun in Dutch but a verb in English. Searching for the Dutch pronoun becomes significantly easier once occurrences of it in literals are annotated with the language tag for Dutch (i.e., `nl`). Besides language-tagged literals, high-quality datatype information also significantly improves the results of semantic search. For example, it allows the weight of a bag of potatoes, `"007"^^dt:kilo`, to be distinguished from the name of a fictional character, `"007"^^xsd:string`.

The metadata on literal datatypes and language tags can be exploited by search systems to improve the effectiveness of their search and bring users closer to their desired results. However, as almost no previous work has focused on analysis and improvement of the quality of literals, contemporary semantic search systems will not make use of this potentially useful metadata. Certain text search tools allow queries to be enriched with meta-information about literals even though the reliability of this information is not high, which may lead to poor results. For instance, LOTUS attempts to improve the precision of literal search by looking up language tags, despite the fact that around 50% of the indexed literals in LOTUS have no language tag assigned to them, which could lead to a decrease in recall for literals with no language tag. Being able to assess whether a given dataset has sufficiently high literal quality would allow Semantic Search systems to improve their precision and recall.

4. Specifying quality criteria for literals

This section presents a theoretic framework for literal quality. It defines a taxonomy of quality categories in terms of the syntax and semantics of literals according to current standards. In addition to the taxonomy, different dimensions of measurement are described. The quality categories and dimensions of measurement can be used to formulate concrete quality metrics. Several concrete quality metrics that are used in later sections are specified at the end of this section.

4.1. Syntax of literals

We define the set of literal terms as $RDF_L := (RDF_I \times LEX) \cup (\{\texttt{rdf:langString}\} \times LEX \times LTAG)$, where RDF_I is the set of IRIs as per RFC 3987 [14], LEX is the set of Unicode strings in Normal Form C [7], and $LTAG$ is the set of language tags as per RFC 5646 (BCP 47)[8]. Literals that are triples are **language-tagged strings** LTS. The first element of a literal is its **datatype IRI**, the second element is its **lexical form**, and the third element, if present, is its **language tag**.

According to the RDF 1.1 standard [9], language-tagged strings are treated differently from other literals. Because the definition of a datatypes does not allow language tags to be encoded as part of the lexical space, language-tagged strings have a datatype IRI (`rdf:langString`), but no datatype. Language-tagged strings do denote values: they are pairs of strings and language tags. All and only literals with datatype IRI `rdf:langString` have a language tag. We use the term **datatyped literal** to refer to literals that have a datatype, i.e., literals that are not language-tagged strings.

RDF serialization formats such as Turtle allow some literals to be written without a datatype IRI. These are abbreviated notations that allow very common datatype IRIs to be inferred based on their lexical form. The datatype IRI that is associated with such a lexical form is determined by the serialization format's specification. For instance, the Turtle string `false` is an abbreviation of the Turtle string `"false"^^xsd:boolean`. Both strings denote the same literal $\langle false, \texttt{xsd:boolean}\rangle$.

4.2. Semantics of literals

The meaning of an RDF name, whether IRI or literal, is the resource it denotes. Its meaning is determined by the interpretation function I that maps RDF names to resources $\mathcal{R}$. Let us call the subset of resources that are datatypes $\mathcal{D}$. A datatype IRI i and the datatype d it denotes are related by the interpretation function: $I(i) = d$. In line with XML Schema 1.1 Datatypes, all RDF datatypes must define the following components:

1. The set of syntactically well-formed lexical forms LEX_d. This is called the **lexical space** of d.

[7] See `http://www.unicode.org/reports/tr15/tr15-37.html`

[8] See `http://www.rfc-editor.org/info/rfc5646`

2. The set of resources VAL_d that can be denoted by literals of that datatype. This is called the **value space** of d.
3. A functional **lexical-to-value mapping** $L2V_d$ that maps lexical forms to values. The resource that is denoted by a literal l is called its **value** or, symbolically, $I(l)$.
4. A **value-to-lexical mapping** $V2L_d$ that maps values to lexical forms. This mapping need not be functional.
5. Optionally, a functional **canonical value-to-lexical mapping** c_d that maps each value to exactly one lexical form.

Suppose we want to define a datatype for colors. We may choose a lexical space $LEX_{\texttt{color}}$ that includes color names like `"red"` and `"yellow"`, together with decimal RGB codes like `"255,0,0"` and `"255,255,0"`. If we define our lexical space in this way, then other strings such as `"FF0000"` do not belong to it (even tough this particular string is commonly used to denote the color red in hexadecimal representation). The lexical to value mapping $L2V_{\texttt{color}}$ maps lexical forms to the color resources they represent. `"red"` maps to the value of redness. `"255,255,0"` maps to the value of yellowness. `"red"` and `"255,0,0"` map to the same value. For the canonical mapping we have to decide which of these two lexical forms should be used to canonically represent redness. Let us say that the decimal RGB notation is canonical (canonicity is a mere convention after all). It follows that $c_{\texttt{color}}(L2V_{\texttt{color}}(\texttt{"red"})) = \texttt{"255,0,0"}$, i.e., the color name maps to the color resource, which maps to the decimal RGB representation. The decimal RGB representation maps to itself, i.e., first to the value of redness and then to the decimal RDF representation.

The denotation of literal terms is determined by the partial interpretation function $LI : RDF_L \to \mathcal{R}$ (Definition 1). LI is partial because a lexical form may not belong to the datatype's lexical space (i.e., an ill-formed literal). Which resource is denoted by which literal is determined by the combination of a specific datatype IRI i and a lexical form lex. Notice that the use of $I(i)$ is required in Definition 1 in order to consider cases in which i denotes the same datatype as `rdf:langString`, but with a different datatype IRI. lc is the function that maps strings to their lowercase variant.

Definition 1 (Literal value)**.**

$$LI(l) := \begin{cases} lex & \text{if } l = \langle lex \rangle \\ \langle lex, lc(tag) \rangle & \text{if } Cond_A(l) \\ L2V_{I(i)}(lex) & \text{if } Cond_B(l) \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

where

$$Cond_A(l) \iff l = \langle \texttt{rdf:langString}, lex, tag \rangle$$
$$Cond_B(l) \iff l = \langle i, lex \rangle \wedge lex \in LEX_{I(i)} \wedge I(i) \neq I(\texttt{rdf:langString})$$

RDF processors are not required to recognize datatype IRIs. Literals with unrecognized datatype IRIs are semantically treated as unknown names. An RDF processor that recognizes more datatypes is therefore not 'more correct' but it is able to distinguish and utilize more subtleties of meaning.

4.3. A taxonomy of literal quality

The categories of literal quality are shown in Figure 1. Because of their fundamentally different syntactic and semantic properties, the quality categories of language-tagged strings are specified separately from those of datatyped literals. The following quality categories are defined for datatyped literals (the categories in Figure 1 are described in the order of a depth-first traversal):

Undefined A datatyped literal is undefined if its datatype IRI does not denote a datatype. Formally: $I(i) \notin \mathcal{D}$. Whether an IRI denotes a datatype or not is not specified in the RDF standards. We therefore specify this ourselves in terms of the four or five component definition of a datatype given in Section 4.2. An IRI is defined iff dereferencing the IRI leads to either (i) a machine-processable datatype definition; (ii) a human-readable formal datatype definition; or (iii) a human-readable informal datatype definition that can be unam-

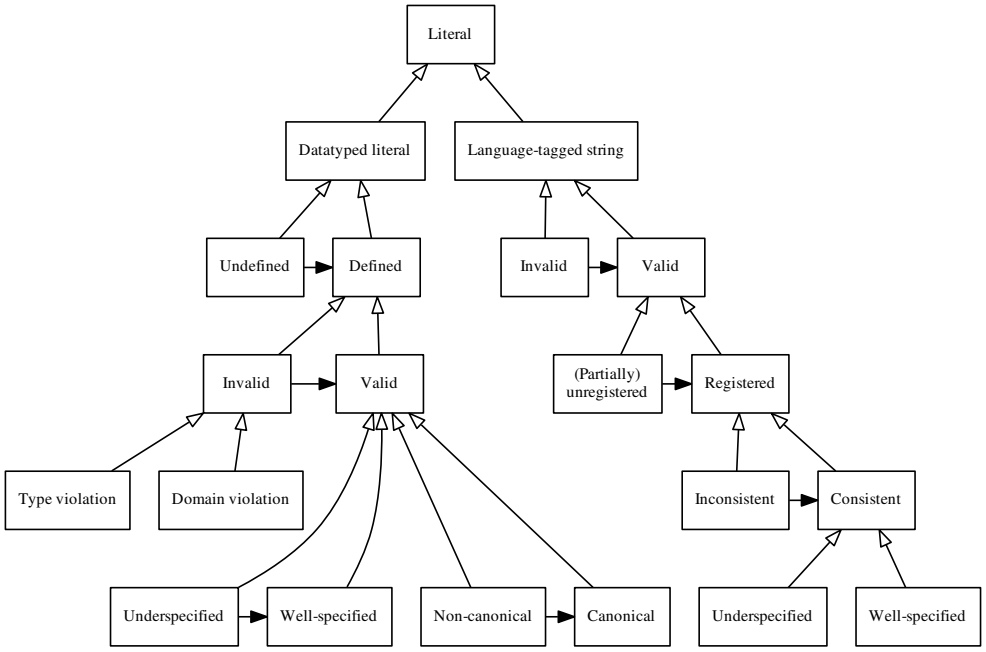

Fig. 1. A taxonomy of RDF literal quality. The nodes show the categories a literal's quality can be classified under. Vertical arrows denote specialization, from more specific to more generic. For instance, literals that are 'Invalid' are also 'Supported' and are also 'Datatyped literals'. Horizontal arrows denote possibilities for quality improvement. For instance, 'Non-canonical' datatyped literals can be made 'Canonical'

biguously turned into a formal one. For instance, the XSD datatype IRIs point to the XML Schema 1.1 Part 2: Datatypes specification, which includes (i) and (ii).

Defined A defined datatyped literal has a datatype IRI that denotes a defined datatype. Formally: $I(i) \in \mathcal{D}$.

Invalid A defined datatyped literal is invalid if its lexical form cannot be mapped to a legal value. Formally: $L2V_d(lex) \notin VAL_d$.

Type violation A supported datatyped literal has a type violation if its lexical form cannot be parsed according to the grammar associated with its datatype. Formally: $lex \notin LEX_d$. Example: `"nineteen hundred"` violates the grammar of datatype `xsd:gYear`. However, `"1900"` does not violate that grammar.

Domain violation A supported datatyped literal has a domain violation if its lexical form can be parsed according to the grammar associated with its datatype, but the parsed value violates some additional domain restriction. Formally: $lex \in LEX_d \wedge L2V_d(lex) \notin VAL_d$. Example: `"3000000000"` can be parsed according to the grammar for integer representations, but its value violates the maximum value restriction of datatype `xsd:int`. However, the same value does belong to the domain of `xsd:integer` which does not have a maximum value restriction.

Valid Supported datatyped literals whose lexical form can be mapped to a value that satisfies all additional constraints for its datatype are valid. Formally: $L2V_d(lex) \in VAL_d$. Valid literals are of higher quality than invalid ones because they expose more meaning, i.e., the RDF processor does not treat them as unknown names.

Underspecified A valid datatyped literal is underspecified if its datatype is too generic. Example: the number of people in a group can be correctly represented by a literal with datatype `xsd:integer`. However, since a group cannot contain a negative number of peo-

ple, it is more descriptive to use the datatype `xsd:nonNegativeInteger` instead. A special form of underspecification occurs when no explicit datatype is given and the datatype `xsd:string` is used as a default, even though a more descriptive datatype could have been chosen. An example of this is `"2016"^^xsd:string`. While this is a correct literal, `"2016"^^xsd:gYear` is more descriptive under the assumption that `2016` denotes a year in the Gregorian calendar. This quality issue is difficult to detect by automated means, because it relies on the intention of the original data creator.

Well-specified A valid datatyped literal that is not underspecified.

Non-canonical A non-canonical datatyped literal is a valid datatyped literal for which there are multiple ways in which the same value can be represented, and whose lexical form is not the one that is conventionally identified as the canonical one. Formally: $c_d(L2V_d(lex)) \neq lex$. Example: `"01"^^xsd:int` and `"1"^^xsd:int` denote the same value but the former is non-canonical. Some datatype definitions do not include a canonical value-to-lexical mapping. For these datatypes we cannot determine whether their lexical values are canonical or not.

Canonical A datatyped literal whose lexical form is canonical for the value denoted by that lexical form. Formally: $c_d(L2V_d(lex)) = lex$.

Language-tagged strings are sufficiently different from datatyped literals to receive their own quality categories. Specifically, language-tagged strings cannot be undefined ('Undefined' in Figure 1) because datatype IRI `rdf:langString` is not supposed to denote a datatype. In addition, their validity ('Valid' in Figure 1) cannot be defined in terms of a lexical-to-value mapping because such a mapping does not exist for language-tagged strings. We can distinguish the following quality categories for language-tagged strings:

Invalid A language-tagged string is invalid if its language tag violates the grammar specified in RFC 5646 (BCP 47). Example: `en-US` is a well-formed language tag, but `english` is malformed. It is also possible for the lexical form to be invalid, e.g., when a typo or a grammatical mistake was made.

Valid A language-tagged string is valid if it is not malformed/invalid.

Unregistered A well-formed language-tagged string is unregistered if the subtags of which its language tag is composed are not registered in the IANA Language Subtag Registry[9]. If only some subtags are not registered then the language-tagged string is partially unregistered.

Registered A well-formed language-tagged string is registered if it is not partially unregistered.

Inconsistent Since the values of language-tagged strings are pairs of strings and language tags, it is possible for the string to contain content that is not (primarily) encoded in the natural language denoted by the language tag. If this occurs then the string and language tag are inconsistent. Example: in language-tagged string `"affe"@en` the string `"affe"` is correct and the language tag `en` is both valid and registered, but the word 'affe' is a commonly used word in the German language (denoted by `de`) but not the English language (denoted by `en`).

Consistent A registered language-tagged string whose lexical form is a valid string in the language denoted by the language tag.

Underspecified Underspecification occurs when the language tag of a language-tagged string is correct, but there exists a more specific language tag that is also correct. For example, `"color"@en` can be more descriptively represented as `"color"@en-US`. As with underspecified datatyped literals, this quality issue is difficult to detect by automated means.

Well-specified A consistent language-tagged string that is not underspecified.

There is a variant of the quality issue of underspecification that connects the sub-hierarchies of datatyped and language-tagged literal quality. Literals with datatype `xsd:string` can be underspecified in case their lexical form contains linguistic content in a particular language. For exam-

[9] See `http://www.iana.org/assignments/language-subtag-registry/language-subtag-registry`

ple, `"semantics"^^xsd:string` can be more descriptively represented as `"semantics"@en`.

There are issues with the 'Underspecified' language-tagged string category in Figure 1. The current standards are insufficient for annotating natural language strings in several cases. For instance, proper nouns are often spelled the same way in different languages. Currently, the only option is to add distinct triples for each language in which the proper noun is used (e.g., `"Amsterdam"@en`, `"Amsterdam"@nl`, etc). This can obviously results in a prohibitively large number of triples. Unfortunately, leaving the string untagged (e.g., `"Amsterdam"^^xsd:string`) is not a good option either, because there is no way to distinguish a universally interpretable or cross-language string from a single-language string that just happens to be untagged.

Another limitation of the current standards surfaces when dealing with multi-lingual strings. The only solution for tagging these strings is to make assertions in which the string appears in the subject position and an RDF name that denotes a language appears in the object position. The predicate must denote a property that relates strings to one of the languages to which some of the string content belongs:

```
"Amsterdam" ex:hasLanguage ex:english,
                           ex:dutch   .
```

Since literals are normally not allowed to appear in the subject position, a blank node to literal mapping has to be maintained as well:

```
_:1 <-> "Amsterdam"
_:1 ex:hasLanguage ex:english,
                   ex:dutch   .
```

Because language tags cannot be used as RDF names, there is no way to link the object term identifier to its corresponding language tag.

4.4. Quality metrics

The quality of literals can be assessed at different levels of granularity. We distinguish between at least the following three levels:

Term level The quality of individual literals.

Datatype level The quality of literals that have the same datatype.

Document level The quality of literals that appear in the same document. The quality of literals in a document is an important ingredient for assessing the overall quality of that document.

The quality categories in Figure 1 can be measured at each of the three granularity levels. Measurements on the literal level are straightforward: every literal belongs to some leaf node(s) in the taxonomy. In most cases a literal belongs to exactly one leaf node. The only exception is valid datatyped literals: they belong to either 'Underspecified' or 'Well-specified' and to either 'Non-canonical' or 'Canonical'.

LOD Laundromat is a stream-based data cleaning infrastructure that is able to assess the quality of literals at the term and datatype levels. As such, it can give overviews of the state of literal quality on the LOD Cloud. Luzzu is a data quality framework that assesses literal quality at the document level.

More complex quality metrics can be defined in terms of the atomic quality categories in Figure 1. Since Luzzu is an extendable framework, new composed metrics can be defined in it. For example, Luzzu measures the ratio of valid literals, or $\frac{|Valid-Datatyped-Literal|}{|Datatyped-Literal|}$. It also calculates the ratio of consistent language-tagged strings, or $\frac{|Valid-LTS|}{|LTS|}$.

The literal quality metrics introduced in this section are used in the analyses performed in Section 6. The first analysis is conducted in LOD Laundromat and covers the term and datatype levels. The second analysis is conducted in Luzzu and covers the document level.

5. Implementation

In this section we describe the data and software that are used for the analysis in Section 6 and the automatic quality improvements in Section 7.

5.1. Data

The analysis and the evaluation of the improvement modules are conducted over the LOD Laundromat [4] data collection, which currently consists of about 650 thousand RDF documents that contain 38 billion ground statements. The data is collected from data catalogs (e.g., Datahub[10]) and contains datasets that users have uploaded through the Web API[11]. As such, the LOD Laundromat data collection only contains a subset of the Linked Open Datasets that are currently available on the web. Since it includes data that has been registered in Datahub, it at least includes the data that has traditionally been thought of as making up the LOD Cloud. The LOD Laundromat collection contains approximately 12.4 billion literals.

Since the LOD Laundromat only includes syntactically processable statements, it is missing all literals that are part of syntactically malformed statements. The reason for this is that whenever a statement is syntactically malformed it is impossible to reliably determine whether a literal term is present and, if so, where it occurs. For example, the syntactically malformed line (A) inside a Turtle document may be fixed to a triple (B), a quadruple (C) or two triples (D). The 'right' fix cannot be determined by automated means.

```
(i)     <a> <b> "c, d>     .
(ii)    <a> <b> <c,d>      .
(iii)   <a> <b> "c,"  <d> .
(iv)    <a> <b> "c" , <d> .
```

The absence of literals that appear within syntactically malformed statements does not influence the meaning of an RDF graph or dataset. A statement must (at least) be syntactically well-formed in order to be interpretable. RDF semantics describes meaning in terms of truth-conditions at the granularity of statements: $I(\langle s,p,o\rangle) = 1 \iff \langle I(s), I(o)\rangle \in EXT(I(p))$, where I is the interpretation function mapping terms to resources and EXT is the extension function mapping properties to pairs of resources. Even though a literal denotes a resource, that denotation alone does not express a basic thought or proposition. Paraphrasing Frege, it is only in the context of a (syntactically well-formed) triple that a literal has meaning.

Since we focus on the quality of literals, we do not cover quality issues that are not specific to literals. This mainly includes various encoding issues. For instance, a Turtle file that uses Latin-1 encoding, whereas the Turtle specification requires the use of UTF-8. When the encoding of a file is wrong or unknown, the file may contain characters that are probably not intended by the original data publishers. Such characters can then also appear in literals[12].

5.2. Toolchain

The toolchain consists of the following components:

ClioPatria The RDF libraries used for parsing, interpreting and serializing RDF data, including the literals.

LOD Laundromat The data cleaning and republishing framework whose data is used in our evaluations. The LOD Washing Machine uses ClioPatria libraries.

Frank A command-line tool that provides easy remote access to the LOD Laundromat data collection.

ALD libraries Existing libraries for detecting natural languages. These are run over data supplied by *Frank*.

[10] See `http://datahub.io`

[11] See `http://lodlaundromat.org/basket`

[12] An example of what is probably an encoding issue that appears in a literal: `http://lotus.lodlaundromat.org/retrieve?string=%C3%85%E2%84%A2&match=terms&rank=psf&size=500&noblank=false`

Luzzu A quality assessment framework for RDF documents. This is run over data supplied by *Frank*.

All components of the toolchain are (of course) published as Open Source software and/or as web services to the community. We now describe each component in more detail.

ClioPatria[13] *[25]* is a Prolog-based triple store and RDF library implemented in SWI-Prolog[14]. We have implemented datatype definitions according to the standards-compliant specification in Section 4.2 for datatype IRIs that commonly appear in the LOD Laundromat data collection and for which such a specification can be found (category 'Defined' in Figure 1).

We have compared the results of our datatype implementation in ClioPatria with RDF4J[15] [8], another Open Source triple store and RDF library. The comparison is carried out to the extent that both libraries now give the same canonical lexical forms for almost all standard XSD datatypes. 'Almost', since there is still some deviation in the canonical forms of XSD doubles and XSD floats. This deviation is allowed by the XML Schema 1.1 specification[16]. In addition, we have implemented several other often occurring datatypes such as `dct:W3CDTF` and `dct:RFC4646`. For these less common datatypes it is not so easy to check our implementation's correctness through comparison with other tools, since very few implementations of these datatypes exist.

LOD Washing Machine[17] is the Linked Data cleaning mechanism that powers the LOD Laundromat ecosystem. The analysis of datatyped literals (Section 6.1) is directly implemented into the Washing Machine. This means that all literals published by the LOD Laundromat are from now on guaranteed to be valid and canonical (if a canonical mapping exists).

The LOD Washing Machine cleans the data in a stream, on a per-tuple basis. This means that memory consumption is almost negligible. The grammars of the implemented datatypes are all $LL(1)$ grammars, i.e. they process the input string from left to right and return only the leftmost derivation. This implies that the grammars are deterministic context-free grammars that leave no choice points during parsing. This means that the computational complexity of parsing all lexical forms is linear in the length of the input string.

Frank[18] is a command-line tool that allows data from the LOD Laundromat data collection to be streamed at the level of triple pattern fragments [3]. This tool is used for the analysis of the quality of language-tagged strings (Section 6.2) and for the analysis of the quality of data documents (Section 6.3). It is also used for the automated improvement of language tags (Section 7.2).

ALD libraries For the assessment and improvement of language-tagged strings we use three existing state-of-the-art Automatic Language Detection (ALD) libraries:

Apache Tika We use a NodeJS wrapper[19] for the 1.10 version of Apache Tika[20]. Apache Tika constructs a language profile of the text to detect and compares it with the profile of the set of known languages. The profiles of these languages are collections of texts which should be representative for the usage of those languages in practice. Such language profile is called corpus. The corpus accuracy depends on the profiling algorithm chosen (word sets, character encoding, N-gram similarity, etc.). Apache Tika uses 3-gram similarity as such three-word groups are useful in most practical situations. According to the documentation, this algorithm is expected to work accurately also with short texts. Tika can detect 18 lan-

[13] See `https://github.com/ClioPatria/ClioPatria`
[14] See `http://www.swi-prolog.org`
[15] See `http://rdf4j.org/`
[16] See `http://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/`
[17] See `https://github.com/LOD-Laundromat/LOD-Laundromat`
[18] See `https://github.com/LOD-Laundromat/Frank`
[19] See `https://github.com/ICIJ/node-tika`
[20] See `http://tika.apache.org/1.10/index.html`

guages (17 languages with European origin and Thai language).

Compact Language Detection (CLD) The NodeJS CLD library[21] is built on top of Google's CLD2 library[22]. The original library recognizes text in 83 languages, while the NodeJS wrapper detects text in over 160 languages. CLD is programmed as a Naive Bayesian classifier which chooses one of the following three algorithms: based on unigrams, on quadrams or defined by the script itself. Aiming to improve upon its performance, this library makes use of hints supplied by the user, on text encodings, expected language or domain URL.

Language-detection This library[23], sometimes abbreviated as LangDetect or LD, is a Java-based library that is commonly used as a language detection plugin for ElasticSearch[24]. This library uses a 3-gram similarity metric and a Naive Bayesian filter. The language profiles (corpora) used by the library have been generated from Wikipedia abstracts. This library supports 53 languages and reports a precision of 99.8%.

The chosen ALD libraries are widely used and are known to have high accuracy for the supported languages and text sizes. Although the chosen set still remains – to some extent – arbitrary, it is trivial to include more libraries into our framework, as one sees fit.

Luzzu[25] is a quality assessment framework for Linked Data. The rationale of Luzzu is to provide an integrated platform that: (1) assesses the quality of RDF documents; (2) provides queryable quality metadata on the assessed documents; (3) assembles detailed quality reports on assessed documents. Luzzu allows the set of quality metrics to be easily extended by users by defining custom and domain-specific metrics.

The following literal-specific quality metrics are implemented in the Luzzu framework: (i) the validity of a datatype against its lexical value (category 'Valid' in Figure 1); and (ii) the consistency between a language-tagged string's lexical form and language tag (category 'Consistent' in Figure 1).

For calculating the ratio of consistent language-tagged strings (Section 4.4) Luzzu uses natural language web services. For single word literals it uses the Lexvo [10] web service[26]. Lexvo is a linguistics knowledge base that encodes relationships between words (e.g., different meanings and translations of words). It also contains links to other semantic resources in the LOD Cloud. For language-tagged strings, a request to the Lexvo API is made and a dereferenceable resource-denoting URI is returned. This resource is then queried and if a `rdfs:seeAlso` is found, then we deem a string literal to have the correct tag.

For checking the correctness of multi-lingual language-tagged strings Luzzu uses the Xerox Language Identifier[27]. This web service identifies the language of natural language phrases and sentences. While the service often returns correct results for languages that commonly occur in the LOD Cloud, this approach does not guarantee that the correct language will always be found [21]. The fact that this approach gives only approximately correct results is taken into account when encoding the metric into metadata.

6. Analysis

This section presents three analyses that are conducted to explore the framework presented in the previous section. The first analysis assesses multiple aspects of the quality of datatyped literals on a large scale. The second analysis assesses one quality aspect of language-tagged string, also on a large scale. The third analysis assesses quality aspects of datatyped literals and language-tagged

[21] See `https://github.com/dachev/node-cld`
[22] See `https://github.com/CLD2Owners/cld2`
[23] See `https://github.com/shuyo/language-detection`
[24] See `https://github.com/jprante/elasticsearch-langdetect`
[25] See `https://github.com/EIS-Bonn/Luzzu`
[26] See `http://lexvo.org`
[27] See `https://services.open.xerox.com/bus/op/LanguageIdentifier/GetLanguageForString`

string within documents. These three approaches are complementary, and together cover a large area of literal term use: datatyped as well as language-tagged, and LOD Cloud-wide as well as document-based.

6.1. Analysis 1: The quality of datatyped literals

We analyze 1,457,568,017 datatyped literals from the LOD Laundromat data collection. Table 1 gives an overview of the ten most occurring datatype IRIs. These are all from the RDF and XSD specifications.

Table 1

The ten most occurring datatype IRIs for the literals that were sampled from the LOD Laundromat data collection

Datatype IRI	Occurrences	Percentage
`xsd:string`	594,614,300	40.67%
`rdf:langString`	517,920,696	35.43%
`xsd:integer`	140,315,796	9.60%
`xsd:int`	74,920,049	5.12%
`xsd:date`	54,830,685	3.75%
`xsd:float`	30,152,391	2.06%
`xsd:double`	17,862,360	1.22%
`xsd:decimal`	11,839,366	0.81%
`xsd:gYear`	5,148,174	0.35%
`xsd:nonNegativeInteger`	3,535,255	0.24%
Others	10,872,531	0.74%
Total	1,462,011,603	100.00%

For the datatyped literals we investigate the following quality categories defined in Section 4.3: undefined, invalid, and non-canonical. Overall we find that the vast majority of literals are valid and a modest majority of them are also canonical. However, 76% (or 1,112,534,996 occurrences) of literals are (language-tagged or XSD) strings (see Table 1). This is not surprising since strings enforce the least syntactic restrictions. Specifically, XSD string is often chosen as the default datatype in case no explicit datatype IRI is provided. It is relatively uncommon for an XSD string literal to be invalid, since in order to do so it must contain unescaped non-visual characters such as the ASCII control characters. For the more complex datatypes, e.g., those that denote dates, times and floating point numbers, the grammar is more strict. In these cases we see that there is still a lot of room for improvement (see the results below).

Undefined Most datatype IRIs do not dereference to a proper definition of a datatype. Many datatypes that have some form of human-readable informal description do not provide enough information in order to properly implement them. An example of this is datatype IRI `Susan:Markdown` whose 'definition' is shown in Listing 1. This informal specification is insufficient in order to define a datatype: Firstly, the value space can either be defined as the set of Markdown-formatted strings or in terms of a formal abstraction of Markdown documents (e.g., a parse tree). For comparison, the value space for `rdf:XMLLiteral` is defined in terms of the XML DOM tree model. Secondly, the Markdown grammar that is pointed to by the `rdfs:seeAlso` property is itself not formally specified. Finally, there does not yet exist a (generally accepted) canonical form for writing down Markdown.

Listing 1: Informal description of the Markdown datatype

```
sysont:Markdown a rdfs:Datatype ;
  rdfs:comment "A string literal formated
                using markdown syntax." ;
  rdfs:label   "Markdown formated string" ;
  rdfs:seeAlso "http://daringfireball.net/
                projects/markdown/syntax" .
```

We notice that there is currently not a strong practice of defining datatypes in terms of XML Schema. In fact, we did not find such a definition outside of the original XSD specification. Also, while there is no inherent reason why an informally specified datatype should be ambiguous or incomplete, in practice we have not found an informal description that is unambiguous and complete. Table 2 shows the most often occurring undefined datatypes. The vast majority of these are DBpedia datatype IRIs (namespace `dt`). For instance, it is unclear whether `dt:second` should be able to denote partial seconds (floating point versus integer number) or whether it should be able to represent a negative number of seconds.

Some datatypes are defined but do not include the optional canonical mapping. An example of such

Table 2
The most often occurring undefined datatyped literals

Datatype IRI	Occurrences	Percentage
`dt:second`	2,326,298	0.160%
`dt:minute`	682,790	0.047%
`dt:squareKilometre`	643,493	0.044%
`dt:centimetre`	382,281	0.026%
`dt:kilogram`	356,321	0.024%

Table 3
The datatype IRIs with the highest number of invalid literals. The percentage is calculated relative to the total number of literals with a given datatype

Datatype IRI	Occurrences	Percentage
`xsd:int`	511,741	0.69%
`xsd:decimal`	122,738	1.28%
`xsd:dateTime`	98,505	8.54%
`xsd:gYearMonth`	16,469	15.45%
`xsd:gYear`	11,957	0.23%

a datatype IRI is `dct:W3CDTF`, a temporal datatype that allows multiple lexical forms to denote the same point in time. For instance, `+01:00` and `-23:00` represent the same time zone. We notice that a canonical mapping is sometimes hard to specify and may sometimes not be very useful. An example of this is `rdf:HTML` which does not specify a canonical mapping, which would have to map an arbitrary HTML DOM tree to a canonical HTML serialization (including white-space use, encoding decisions, canonization of non-HTML content like CSS or JavaScript, etc).

Invalid Table 3 shows the datatypes that have the highest number of invalid literals. Overall, only 0.11% of all literals are invalid. However, as was mentioned before, 79% of all literals are strings for which almost every lexical form is valid. As soon as the datatype becomes more complicated, the percentage of invalid occurrences goes up. For example, many integers with datatype IRI `xsd:int` exceed the short integer range constraints. Another example are literals with datatype IRI `xsd:gYearMonth`, whose lexical forms often swap the year and month parts, or where the month part is often an RFC 822-style month name (e.g., `Jan`).

Table 4
The datatype IRIs with the highest number of non-canonical literals. The percentages are calculated relative to the number of literals with a specific datatype IRI

Datatype IRI	Occurrences	Percentage
`xsd:float`	30,152,304	99.99%
`xsd:double`	17,783,414	99.56%
`xsd:decimal`	2,127,133	1.05%
`rdf:XMLLiteral`	245,457	100.00%
`xsd:dateTime`	224,994	7.14%

Non-canonical Table 4 shows the five datatypes with the highest number of non-canonical literals. Overall, 3.5% of all literals are non-canonical. Again, simple strings are canonical by definition, since they map onto themselves. On the other hand, the majority of the floating-point numbers (either `xsd:double` or `xsd:float`) are non-canonical. The reason for this is that their canonical format is quite specific: it must always be written in scientific notation and must use the uppercase exponent sign 'E'. In practice, we see that `1.0` is a much more common serialization of a floating-point number than its canonical counterpart `1.0E0`.

6.2. Analysis 2: The quality of language-tagged strings

We want to analyze literals with textual content, including textual content that has been explicitly tagged with a language tag (i.e., language-tagged strings) and textual content that is untagged (i.e., XSD strings). It is difficult to reliably determine when a literal contain textual content, which is an inherently vague notion. We want to at least exclude many XSD strings that are obviously non-textual. For this we require the lexical form of an XSD string to at least contain two consecutive Unicode letters. This coarse filter removes lexical forms that encode dates, lengths, telephone numbers, etc. Such non-textual strings are probably stored as XSD strings because the data publisher was unaware of an appropriate datatype, and/or did not have enough time to perform a proper transformation to an existing datatype.

When we use our coarse filter to distinguish literals with textual content, this results in 3,54

Table 5

Unregistered primary language tags that appear in the highest number of language-tagged strings

Language tag	Occurrences
ck	1,191,661
il	1,041,376
x-	155,782
pm	97,898
gs	80,119

billion literals. 2.26 billion (or 63.83%) of these are language-tagged strings (have an explicit language tag). The remaining 1.28 billion (or 36.17%) are XSD strings (do not contain a language tag). These literals originate from 569,422 documents from the LOD Laundromat data collection.

The distribution of language tags over this collection of literals is given in Table 6. By far the most language-tagged literals are in English, followed by German, French, Italian and Spanish. This shows that Linked Data contains a strong representation bias towards languages of European origin, with the 10 most frequent language tags representing European languages. 73.26% of all language-tagged literals belong to one of the 10 most frequently occurring languages.

Unregistered We analyze how many of the language-tagged literals are 'Registered' (see Figure 1) by assessing whether their primary language tag belongs to the IANA language codes registry. Out of 313 two-digit country codes in the LOD Laundromat collection, 186 (59.4%) are also registered in IANA. The vast majority of language-tagged literals (98.6%) contains a registered language tag. Table 5 presents the five most frequently occurring unregistered language tags.

6.3. Analysis 3: The literal quality of documents

In order to illustrate that the here presented toolchain integrates well with document-based quality frameworks, we run initial experiments of the Luzzu framework, receiving data and metadata through the *Frank* tool. Luzzu quantifies the quality of Linked Data documents, including the quality of literals. We used Luzzu in order to process 470 data document from the LOD Laundromat collection. For each of these documents Luzzu calculates the ratio of valid literals and the ratio of consistent language-tagged strings (Section 4.4). For these specific documents Luzzu determines that, on average, 70% of the language-tagged strings are consistent with respect to their language tag. On the other hand, only 39% of literals have a lexical form that belongs to the value space denoted by its datatype IRI (category 'Valid' in Figure 1).

Table 6

The distribution of language tags in the LOD Laundromat data collection

Language tag	Occurrences
en	878,132,881
de	145,868,558
fr	129,738,855
it	104,115,063
es	82,492,537
ru	77,856,452
nl	75,226,900
pl	59,537,848
pt	56,426,484
sv	47,903,859
other language tag	607,012,252
XSD string	1,281,785,207
textual literals	3,544,028,391

Manual inspection of the sample of documents whose quality value was less than 40% reveals the following four main issues for language-tagged strings:

- A literal contains linguistic content but lacks a language tag (category 'Underspecified' in Figure 1).
- A literal contains both linguistic and non-alphabetic content (the multi-linguality problem discussed in Section 4). Example: `"related_software"@en`.
- A literal contains linguistic content with syntax errors (category 'Invalid' in Figure 1). Example: `"flow cytometer sorter"@en`.
- A literal has a language tag that is not supported by the automated approaches used by Luzzu. Example: `"article"@en-US`.

The majority of problematic triples exhibit the first issue. The third issue, i.e., lexical forms containing syntax errors, actually result in incorrect identifications by the external services that are used to calculate the metrics. For example, in `"flow cytometer sorter"@en` the term *cytometer* should have been written as *cytometry*. The fourth issue points at another tooling issue, but one that cannot be so easily resolved. Although the tag `en-US` is correctly according to RFC 5646 (BCP 47), the Luzzu metric expects two- or three-letter language tags, these are supported by most NLP resources, such as the Linked Languages Resources[28].

Interestingly, many of the quality issues that were found upon manual inspection overlap with the ones that we considered problematic in Section 4. This leads us to believe that typo's and multi-linguality may not be fringe cases after all, and that current standards may actually be too coarse to deal with the real-world quality issues of language-tagged string as they are used in the LOD Cloud today. Future work into these issues is needed.

7. Improvement

In this section we show that the quality of literals can be significantly improved by using the processing and analysis framework presented in Section 5 and Section 6. The possible quality improvements are defined in the literal quality taxonomy in Section 4, as indicated by the horizontal arrows with filled arrows in Figure 1. Based on the analysis in the previous section we are informed about some of areas where literal quality can be improved.

We note that not all aspects of literal quality can be improved by automated means. For instance, the quality improvement from 'Underspecified' to 'Well-specified' in Figure 1 cannot be effectuated based on the available data alone but needs an interpretative decision from the original data publisher. Even though these quality issues cannot be fixed automatically, the current framework can still be used to automatically detect such problems. In general, suggestions for quality improvement can now be based on empirical observation rather than intuition.

In order to show that our toolchain indeed provides the required scale to fix quality issues in the LOD Cloud, we choose two quality aspects that can be automated. These two quality aspects also support two use cases in Section 3. The first one is the automatic conversion of non-canonical datatyped literals to canonical ones, supporting the efficient calculation of equivalence tests. The second one is the automatic assignment of language tags to textual literals that did not have a language tags before, supporting improved multilingual search indexing.

7.1. Improving datatyped literals

Undefined The analysis in Section 6 gives an overview of the size of the quality issue of undefined datatypes. Based on this overview we can see that defining the DBpedia datatype IRIs would solve the vast majority undefined datatype IRIs, thereby significantly increasing the overall quality of literals in the LOD Cloud.

Underspecified → well-specified An underspecified literal cannot be changed into a well-specified one based on the observed lexical form alone. For instance, the fact that the values `"0001"`, `"0203"`, `"9009"` appear in the data does not tell us whether the datatype IRI should be `xsd:positiveInteger` or `xsd:nonNegativeInteger`. Deciding on the most general primitive datatype, `xsd:decimal` in this case, also does not suffice since, for this particular example, `xsd:gYear` would apply just as well. The problem becomes even more complex when non-standard datatypes are considered, which can map lexical form `"11"` to Metaphysics and `"657"` to accountancy (e.g., this particular mapping is part of the Universal Decimal Classification (UDC)).

Invalid → valid Invalid literals can only be improved upon by the original data publisher. We cannot automate this task since it requires us to

[28] See `http://linkedvocabs.org/lingvoj/`

choose between (1) changing the datatype IRI to match the lexical form, (2) changing the lexical form to match the datatype IRI, or (3) changing both. However, we can give an overview of mistakes that occur most often in the data. Based on our empirical observations these are the top 4 mistakes, along with suggestions of how to avoid them in the future:

1. `xsd:int` is not the same as `xsd:integer`. The former is a short integer and cannot be used to express integers smaller than -2,147,483,648 or larger than 2,147,483,647. This results in range errors that would not have occurred if `xsd:integer` would have been used instead.
2. RDF IRIs are case sensitive [9]. Specifically `xsd:datetime` is not the same as `xsd:dateTime`. The former is not defined by the XSD standard and occurrences of it are probably typos.
3. `xsd:date` must not include a temporal specifier. `xsd:dateTime` is used for this instead.
4. Many datatype IRIs are not proper HTTP(S) IRIs. Since RDF serializations are very admissive when it comes to IRI syntax, many things that are parsed as literals contain datatype IRIs that do not parse according to the more strict IRI RFC specification [14]. Most of these improper datatype IRIs are due to undeclared prefixes (e.g., `xsd`) in the source document. Many of these can probably be expanded according to a list of common RDF aliases and their corresponding IRI prefixes, but the original data publisher should check whether this is indeed the case.

Non-canonical → canonical Canonical literals provide a significant computational benefit over non-canonical valid literals for several use cases. For instance, checking whether two terms or statements are identical or not no longer requires parsing and generating of lexical forms, i.e., string similarity suffices where one would have to calculate $c(L2V(l_1)) \equiv c(L2V(l_2))$ otherwise. ClioPatria now fully automates the canonicalization of RDF literals for which such a mapping is defined: all literals are stored in canonical form upon statement assertion.

7.2. Improving language-tagged strings

Unregistered → registered Standardizing the set of language tags in LOD Laundromat with respect to the central IANA registry would improve the quality of the literals. However, it is not trivial to adjust these language tags automatically. For instance, the unregistered tag `il` may be corrected to `he-il`, which denotes the Hebrew language spoken in Israel; or it might be a typo, where the author intended to point to the Italian language (denoted by `it`). Deciding on the correct registered language depends on the intention of the original data publisher.

No language tag → language tag We attempt to assign a language tag to textual literals that do not have one yet. For this purpose, we test the accuracy of automated language detection algorithms when applied to the textual lexical forms from the LOD Laundromat data collection. We define a textual lexical form as a lexical form of a literal with datatype IRI `xsd:string` or `xsd:langString` that contains at least two consecutive Unicode letters. This excludes many obvious non-textual lexical forms such as telephone numbers and years. As reported in Table 6, about half of these textual lexical forms already have a language tag associated with them inside the LOD Laundromat data collection. We can use these language tags that are specified by original data publisher in order to quantify the accuracy of automated language detection approaches.

We apply three language detection libraries (see Section 5.2) to the textual lexical forms in the LOD Laundromat data collection. We are interested in the following aspects: How often does the automatically detected language tag coincide with the user-assigned tag? How accurate are the language detection libraries? Does the accuracy of detection differ per primary language or for various string sizes? Are certain languages or string sizes easier for language detection? How often do the libraries refrain from assigning a language tag at all? Can we combine the libraries and thereby improve the overall accuracy of language detection?

We use the language-tagged strings that appear in LOD Laundromat and check whether the ALD li-

braries assign the same language tag to the lexical form as the one assigned by the original data publisher. We report the precision, recall and F1-value for each language detection library. We assume that the accuracy that we measure over language-tagged string extends to XSD string literals with a textual lexical form (i.e., ones that do not have a user-defined language tag).

The language tag assigned by the original data publishers can consist of multiple, concatenated subtags. Since our language detection tools only provide an ISO 639-2 two-character language code in most cases, we focus our comparison on the primary language subtag, i.e., the first two characters of each language-tagged string. Another motivation for enforcing this abstraction is that it is more difficult to distinguish fine-grained language differences from a semantic point of view. For instance, if the original data publisher supplied language tag `de-DE`, it is difficult to determine whether `de-AU` or de would also have been correct annotations. The granularity level that we choose, two-character primary language tags, is satisfactory for identifying most languages, although there are exceptional cases in which the secondary language subtag is also required for denoting the language. Most notably, this is the case for Chinese languages where `zh-CN` denotes a language that is different from `zh-TW`.

Table 7 gives an overview of the time that each ALD library needs for annotating the textual lexical forms. The (single-threaded) process of tagging takes 5 days. Most of this elapsed time (around 80%) is used by the ALD libraries themselves. The remaining time is used by the *Frank* tool to extract the required information from the LOD Laundromat web service. Considering that such an improvement procedure only needs to be executed once, and not necessarily in real time, we believe that the time needed to improve the coverage of language tags in the LOD Cloud is reasonable.

We also measure the accuracy of each ALD library and observe that the highest precision (75.42%) is achieved by the CLD library, which also covers the highest number of languages (160). It is notable that this library often gives no language suggestion, especially when it comes to short strings.

Table 7

Running time for each of the three ALD libraries. The difference between the total duration and the library-specific runtime shows that the overhead of streaming through the LOD Laundromat data collection accounts for 20.5 hours

Library	Library runtime	Total runtime
CLD	52 hours	72.5 hours
LD	103 hours	123.5 hours
Tika	109 hours	129.5 hours

We further investigate to which extent the accuracy of the libraries is dependent on specific language tags or string sizes. The outcome of this analysis for the most frequently occurring 10 languages is shown in Table 11. Each of the cells represents an intersection of a primary language tag and a string size bucket. We measure the string size n in terms of number of words that constitute the string. A string size bucket b contains strings whose length fall within the same logarithmic value: $b = |log_2(n)|$. Each cell contains three values that represent the F1-accuracies of the three respective libraries: Tika, CLD and LangDetect. While CLD has the highest accuracy for most buckets, there are notable exceptions. For instance, the LangDetect library is better than the CLD library in detecting Portuguese and long Dutch literals.

Figure 2 shows the aggregated accuracy per bucket for each of the three libraries. Note that there is hardly any intersection of the plotted lines: for any text size bucket (except for 0), CLD gives the highest F1-value, while Tika gives the lowest. However, the text size does correlate with the general success of language detection (by any library). Concretely, short strings which contain only one word (bucket 0) or two words (bucket 1) are much harder to detect correctly than longer strings. On the other hand, expressions from bucket 8 (between 129 and 256 words) can be detected with almost perfect accuracy.

This tendency is confirmed for the most frequent 10 languages (Figure 3). Every data point represents the average F1-value, calculated over the three libraries for a given language and bucket. This shows that libraries are successful in detecting the language of sufficiently long literals (literals in bucket 3 already have an F-measure

Table 8
Languages among the untagged strings according to CLD

Language tag	Occurrences
en	348,262,051
nl	27,700,617
de	25,166,990
da	12,574,645
ja	9,158,424
es	8,593,138
fr	7,248,383
nn	7,114,156
el	4,323,837
la	2,873,684

Table 9
Languages among the untagged strings according to LD

Language tag	Occurrences
en	331,196,693
de	163,920,782
tl	99,213,393
ca	86,113,223
ro	72,512,668
it	52,062,226
nl	49,516,711
fr	48,139,153
hr	39,207,813
zh	33,611,172

Table 10
Languages among the untagged strings according to Tika

Language tag	Occurrences
lt	273,304,150
ro	141,609,523
en	116,931,204
sk	91,248,715
et	91,082,766
it	69,331,167
fr	57,668,214
hu	54,721,330
no	54,121,551
is	51,962,628

of around 75%, increasing to around 85-90% for bucket 4). Almost all common languages, except for Portuguese, follow this distribution. From Figures 2 and 3 it is clear that the accuracy of the ALD libraries for strings of moderate and long size is very high. In contrast, the language detection for short strings and less common languages has a much lower accuracy.

We also observe a significant decline in accuracy for the extremely large strings (belonging to bucket 10 and above), which is somewhat unexpected. We hypothesize that this decline could be due to the following factors: (1) The buckets 10 and above contain much fewer strings, between a few hundred and a few thousand strings per language, which leads to less stable and less reliable results; (2) the strings that belong to these categories are generally non-standard in terms of size, format and content. Specifically, some strings are so long that they result in errors for entering an unanticipated form of input. One example is the full text of the book "THE ENTIRE PROJECT GUTENBERG WORKS OF CHARLES D. WARNER", which are stored in a single literal term.

In some instances a lexical form can be correctly annotated with multiple primary language tags. This is especially true for proper names – these often share the same surface form in a plurality of languages. For instance, what is the right language tag for "Amsterdam": English, Dutch, or a set of languages? Ideally, the world languages would be grouped into a hierarchy of language tags, thus allowing the data publisher to specify a group or category of similar language tags (e.g. all Germanic languages). This representation is not standardized at the moment (see also Section 4).

It could also be argued that proper names and other short strings should be kept as non-language tagged strings, because their lack of context often allows multiple language tags to be considered correct. This is demonstrated in Figure 2: strings that consist of only a few words are seldom tagged with a consistent language.

Finally, we enumerate the most frequently assigned languages by our libraries on the strings without a language tag in the published data (Tables 8, 9, 10). As on the language-tagged strings in Table 6, the major European languages, especially the English language, seem to prevail on the strings without a language tag. At the same time, we observe a set of small languages that are unexpectedly frequent, such as Tagalog (`tl`), (new) Norwegian (`no/nn`), Greek (`el`), and Estonian (`et`). The high frequency of such long-tail

languages might be an indicator of errors in these libraries.[29]

8. Conclusions

We have focused on the quality of literals, an area of Linked Data quality conformance that has not been thoroughly investigated before. We have systematically specified a taxonomy of quality criteria that are specific to RDF literals. We have shown that quality metrics can be defined in terms of this taxonomy. We have shown that existing platforms and libraries can be reused in order to automatically check for literal quality conformance. Two concrete analyses were conducted on a very large scale and a third analysis has shown that our toolchain can also be used by existing quality assessment frameworks in order to assess the quality of RDF documents. Finally, we have presented initial attempts at automatically effectuating large-scale improvements to the quality of literals.

Implementation reuse The implementations have not only be used for the here presented experiments, but have been consolidated into the ClioPatria Semantic Web library and triple store, which now supports a large number of datatypes as well as canonical forms. The literal quality categories for which conformance can be automatically checked have been integrated into the LOD Laundromat data cleaning process, which now records literal quality issues that are found in the data. Finally, the automatic detection of language tags is used by the Semantic Search engine LOTUS to improve its language-specific filters.

Literal quality & RDF processor conformance Since every empirical measurement of literal quality must use a concrete RDF processor that interprets the input data documents, every empirical measurement of literal quality is inherently relative to the processor that was used. Differences between RDF processors include the set of datatypes that is implemented as well as bugs and/or purposeful deviations from Linked Data standards. While implementing the XSD datatypes in ClioPatria, we discovered that it helps to cross-validate against another library (in our case RDF4J). In a similar way, we hope that the availability of ClioPatria will make it easier for others to add support for more datatypes in their RDF processors as well.

Since this paper has focuses on the quality of literals, conducting a broad investigation into literal support by triple stores and RDF libraries was out of scope. Specifically, we have not conducted a broad investigation into which datatypes are implemented by which triple store or RDF library. Future research should focus on the tool support of literals, because the fact that very few non-XSD datatypes are currently in use may be because of the chicken-and-egg problem that RDF tools do not implement them because they do not occur that often, etc.

Datatype definitions Even though we have shown that many quality issues can be automatically detected, and some can even be (semi-)automatically fixed, the issue of undefined (or badly defined) datatypes remains unresolved. By definition, a literal's quality can only be determined once its datatype is properly specified. The existing of so many undefined datatypes, i.e., almost all datatypes that do not belong to the standard collection of XSD datatypes, may point to a lacuna in today's RDF standards. In these standards, the issue of datatype definition is 'outsourced' to the XML Schema specifications. However, the current generation of Semantic Web practitioners may have less experience with XML Schema and related technologies. In order to improve the current situation, standardization organizations may consider introducing new ways of defining RDF datatypes and/or providing more assistance for using the existing ways.

Our approach This paper presents a novel approach towards quality assessment; one that is large-scale, automated, and easily reusable in the context of other tools and libraries. Current standards and best practices are based on what experts consider to be the most important data quality issues. We show that, in addition to these existing

[29] Most notably, the Tika library shows a bias towards Eastern European and Nordic languages.

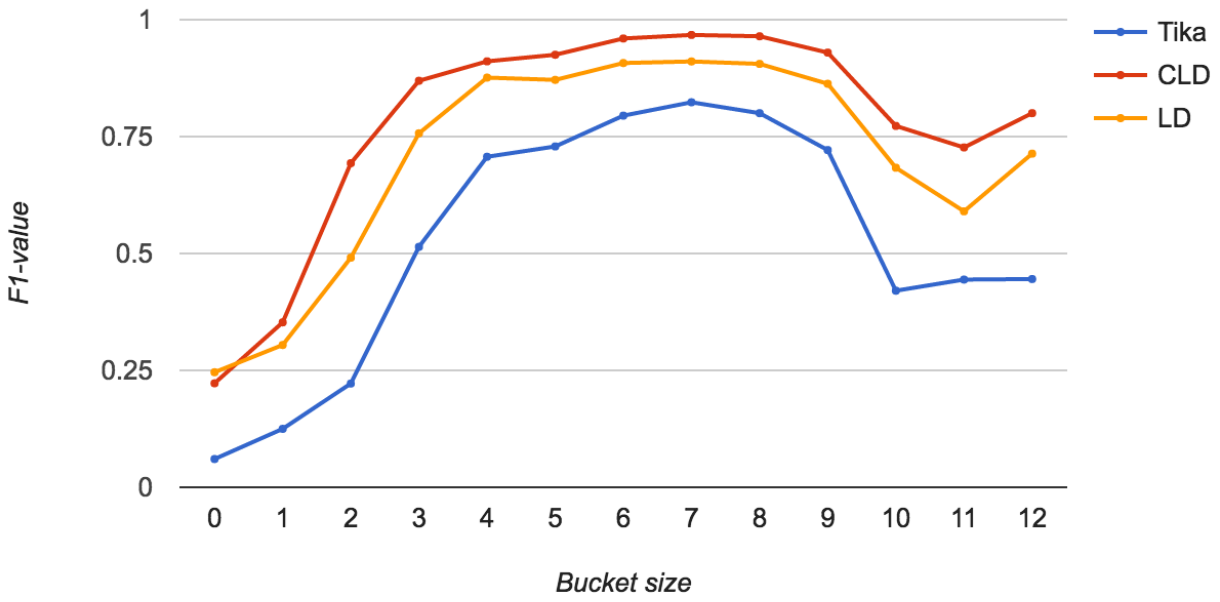

Fig. 2. Accuracy per language detection library

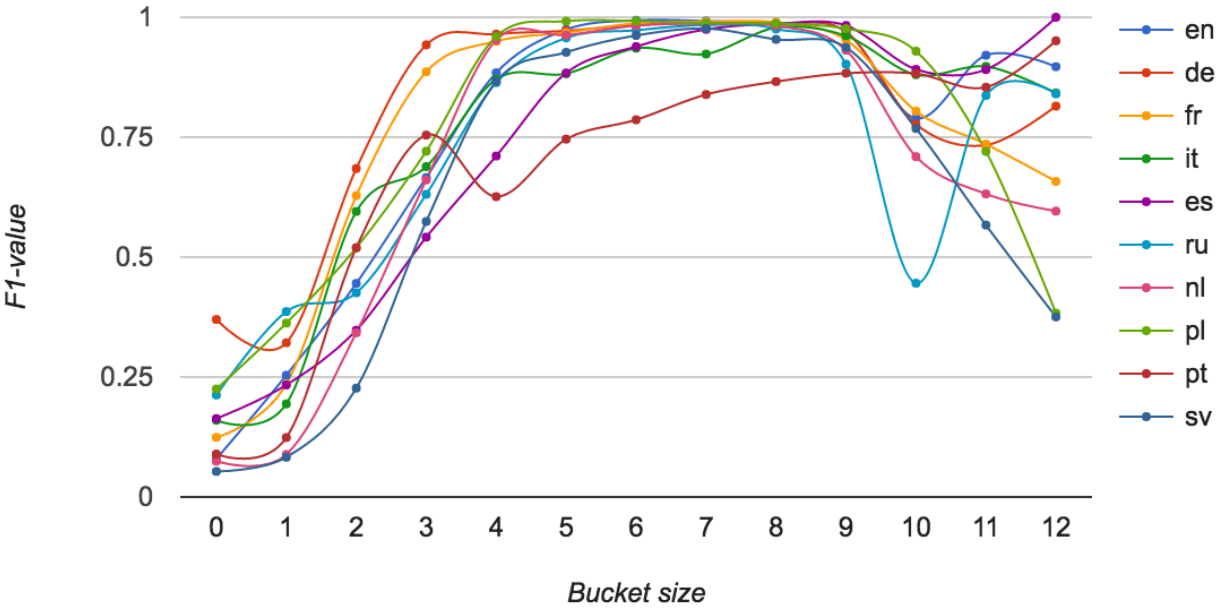

Fig. 3. Accuracy per language tag for the 10 most frequent languages

Table 11

F1-value accuracy of the libraries per bucket size and language. Library results are given in the following order: Tika, CLD, LangDetect. The language tags are ordered by frequency, with the most frequent languages on the top of the Table

	0	1	2	3	4	5	6	7	8	9	10	11	12
en	1.99	5.78	13.74	38.12	76.34	94.77	98.77	98.7	97.94	92.75	69.36	81.89	74.93
	11.88	26.18	41.95	70.09	92.48	98.45	99.58	99.32	98.89	97.14	86.58	94.3	89.38
	8.87	44.13	77.41	91.3	96.57	99.19	99.7	99.42	99.0	97.73	90.06	92.97	91.27
de	5.57	16.63	51.62	87.1	93.03	94.99	97.25	98.18	97.75	94.77	64.75	43.7	66.67
	55.27	48.87	87.39	98.66	98.94	98.79	99.35	99.36	99.4	98.55	85.61	90.76	94.87
	54.24	37.04	76.04	97.34	97.49	97.73	98.65	98.81	98.66	97.59	83.46	85.59	94.87
fr	8.85	16.46	44.88	82.56	90.85	95.24	98.54	98.98	98.79	94.36	76.71	70.72	54.32
	20.74	34.45	75.25	94.98	98.1	98.42	99.09	99.23	99.15	95.88	82.98	77.67	74.07
	8.12	22.29	67.39	88.58	96.07	96.81	98.6	99.02	98.97	95.67	81.55	72.77	63.58
it	14.11	19.06	61.02	71.54	90.19	89.42	93.63	91.81	98.25	96.79	94.73	96.65	89.61
	28.75	29.91	66.01	75.92	91.18	89.4	94.99	92.45	98.44	96.68	94.03	93.92	87.44
	5.45	8.43	34.14	58.26	79.84	85.86	92.06	92.76	96.86	94.99	75.26	78.57	85.36
es	2.56	6.92	21.92	37.32	53.01	75.89	86.31	94.72	97.11	96.66	85.57	86.69	94.37
	33.36	31.07	50.27	70.14	91.93	98.37	99.41	99.46	99.74	99.25	89.98	88.54	95.77
	13.75	22.02	31.3	53.86	68.23	90.91	95.97	98.27	99.09	98.97	91.87	92.06	95.77
ru	25.85	39.73	64.1	64.02	88.56	97.4	97.81	98.43	96.9	85.91	15.69	73.76	86.54
	28.21	47.65	42.05	70.64	93.07	98.92	99.2	99.2	98.42	94.63	79.63	91.96	96.15
	9.23	29.82	26.49	53.42	77.52	90.81	94.87	97.56	97.38	90.1	38.24	85.36	72.94
nl	3.29	4.44	22.47	63.24	96.11	96.9	98.61	98.58	97.78	90.79	59.21	43.86	35.29
	11.43	11.65	40.36	71.4	96.29	97.34	98.72	98.75	98.23	93.26	72.79	65.79	41.18
	6.47	9.15	38.52	62.8	93.33	94.57	97.43	98.66	98.73	95.33	80.96	79.68	74.42
pl	17.98	30.82	49.79	66.18	95.16	99.1	99.19	98.74	98.61	97.58	92.19	71.0	47.37
	30.43	35.41	56.63	73.62	96.55	99.19	99.13	98.65	98.58	97.48	91.67	73.0	47.37
	19.4	38.19	46.3	75.26	96.3	99.25	99.31	98.97	98.93	97.78	94.5	72.13	50.0
pt	3.45	5.3	9.73	39.47	20.29	33.99	39.4	53.72	61.22	67.34	69.42	70.51	82.86
	10.24	14.03	53.21	72.82	84.33	94.67	98.29	98.79	99.09	98.59	97.19	90.38	100.0
	13.33	15.75	59.86	75.81	83.16	95.13	98.22	99.19	99.45	99.14	98.47	96.0	100.0
sv	1.62	3.06	15.68	46.63	76.92	87.56	93.69	96.41	93.21	90.2	64.84	45.24	8.33
	10.82	12.39	28.91	68.63	92.85	96.35	98.3	98.24	94.61	94.24	81.1	57.14	58.33
	5.4	11.55	25.67	62.22	90.32	94.25	96.68	98.37	98.36	96.75	84.97	67.65	60.87

approaches, future initiatives towards data quality improvement may also be based on empirical evidence of the state of the LOD Cloud. Our approach has many benefits, e.g., it allows the impact of quality improvement initiatives to be *quantified* beforehand. This makes it possible to identify the quality issues for which quality improvement would result in the biggest overall impact.

Our approach extends to the topic of automatically improving data quality. We show that millions of literals (and thereby statements) can be improved very quickly, by algorithmic means. While this does not apply to every quality criterion, e.g., ones that rely on the intention of the original data publisher, there are many quality criteria for which this can be done. Indeed, we have shown that by using state-of-the-art libraries – for instance libraries for natural language identification – we are able to improve the overall quality of the LOD Cloud. This means that many of the quality criteria that could previously only be im-

proved on an ad-hoc basis can now be improved at a very large scale, thereby resulting in a Semantic Web that is more valuable for everyone.

References

[1] Maribel Acosta, Amrapali Zaveri, Elena Simperl, Dimitris Kontokostas, Sören Auer, and Jens Lehmann. Crowdsourcing linked data quality assessment. In Harith Alani, Lalana Kagal, Achille Fokoue, Paul T. Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha F. Noy, Chris Welty, and Krzysztof Janowicz, editors, *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II*, volume 8219 of *Lecture Notes in Computer Science*, pages 260–276. Springer, 2013. DOI `https://doi.org/10.1007/978-3-642-41338-4_17`.

[2] Ahmad Assaf, Raphaël Troncy, and Aline Senart. What's up LOD cloud? - Observing the state of Linked Open Data Cloud metadata. In Fabien Gandon, Christophe Guéret, Serena Villata, John G. Breslin, Catherine Faron-Zucker, and Antoine Zimmermann, editors, *The Semantic Web: ESWC 2015 Satellite Events - ESWC 2015 Satellite Events Portorož, Slovenia, May 31 - June 4, 2015, Revised Selected Papers*, volume 9341 of *Lecture Notes in Computer Science*, pages 247–254. Springer, 2015. DOI `https://doi.org/10.1007/978-3-319-25639-9_40`.

[3] Wouter Beek and Laurens Rietveld. Frank: The LOD Cloud at your fingertips. In Ruben Verborgh and Miel Vander Sande, editors, *Proceedings of the ESWC Developers Workshop 2015 co-located with the 12th Extended Semantic Web Conference (ESWC 2015), Portorož, Slovenia, May 31, 2015.*, volume 1361 of *CEUR Workshop Proceedings*, pages 41–46. CEUR-WS.org, 2015. URL `http://ceur-ws.org/Vol-1361/paper8.pdf`.

[4] Wouter Beek, Laurens Rietveld, Hamid R. Bazoobandi, Jan Wielemaker, and Stefan Schlobach. LOD Laundromat: A uniform way of publishing other people's dirty data. In Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig A. Knoblock, Denny Vrandecic, Paul T. Groth, Natasha F. Noy, Krzysztof Janowicz, and Carole A. Goble, editors, *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, volume 8796 of *Lecture Notes in Computer Science*, pages 213–228. Springer, 2014. DOI `https://doi.org/10.1007/978-3-319-11964-9_14`.

[5] Tim Berners-Lee. Cool URIs don't change. Style guide, W3C, 1998. URL `http://www.w3.org/Provider/Style/URI.html.en`.

[6] Tim Berners-Lee. Linked data, 2006. URL `http://www.w3.org/DesignIssues/LinkedData.html`.

[7] Christian Bizer and Richard Cyganiak. Quality-driven information filtering using the WIQA policy framework. *Journal of Web Semantics*, 7(1):1–10, 2009. DOI `https://doi.org/10.1016/j.websem.2008.02.005`.

[8] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF schema. In Ian Horrocks and James A. Hendler, editors, *The Semantic Web - ISWC 2002, First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002, Proceedings*, volume 2342 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2002. DOI `https://doi.org/10.1007/3-540-48005-6_7`.

[9] Richard Cyganiak, Markus Lanthaler, and David Wood, editors. *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation, 25 February 2014. URL `https://www.w3.org/TR/rdf11-concepts/`.

[10] Gerard de Melo. Lexvo.org: language-related information for the linguistic Linked Data Cloud. *Semantic Web*, 6(4):393–400, 2015. DOI `https://doi.org/10.3233/SW-150171`.

[11] Jeremy Debattista, Christoph Lange, and Sören Auer. Representing dataset quality metadata using multi-dimensional views. In Harald Sack, Agata Filipowska, Jens Lehmann, and Sebastian Hellmann, editors, *Proceedings of the 10th International Conference on Semantic Systems, SEMANTICS 2014, Leipzig, Germany, September 4-5, 2014*, pages 92–99. ACM, 2014. DOI `https://doi.org/10.1145/2660517.2660525`.

[12] Jeremy Debattista, Sören Auer, and Christoph Lange. Luzzu - A framework for linked data quality assessment. In *Tenth IEEE International Conference on Semantic Computing, ICSC 2016, Laguna Hills, CA, USA, February 4-6, 2016*, pages 124–131. IEEE Computer Society, 2016. DOI `https://doi.org/10.1109/ICSC.2016.48`.

[13] Zlatan Dragisic, Kai Eckert, Jérôme Euzenat, Daniel Faria, Alfio Ferrara, Roger Granada, Valentina Ivanova, Ernesto Jiménez-Ruiz, Andreas Oskar Kempf, Patrick Lambrix, Stefano Montanelli, Heiko Paulheim, Dominique Ritze, Pavel Shvaiko, Alessandro Solimando, Cássia Trojahn dos Santos, Ondrej Zamazal, and Bernardo Cuenca Grau. Results of the Ontology Alignment Evaluation Initiative 2014. In Pavel Shvaiko, Jérôme Euzenat, Ming Mao, Ernesto Jiménez-Ruiz, Juanzi Li, and Axel Ngonga, editors, *Proceedings of the 9th International Workshop on Ontology Matching collocated with the 13th International Semantic Web Conference (ISWC 2014), Riva del Garda, Trentino, Italy, October 20, 2014.*, volume 1317 of *CEUR Workshop Proceedings*, pages 61–104. CEUR-WS.org, 2014. URL `http://ceur-ws.org/Vol-1317/oaei14_paper0.pdf`.

[14] Martin J. Dürst and M. Suignard. Internationalized Resource Identifiers (IRIs). RFC 3987, Internet Egnineering Taskforce, January 2005. URL `https://www.rfc-editor.org/rfc/rfc3987.txt`.

[15] Christian Fürber and Martin Hepp. Towards a vocabulary for data quality management in Semantic Web architectures. In *Proceedings of the 1st International Workshop on Linked Web Data Management*, pages 1–8, New York, NY, USA, 2011. ACM. ISBN

978-1-4503-0608-9. DOI https://doi.org/10.1145/1966901.1966903.

[16] Aidan Hogan, Andreas Harth, Alexandre Passant, Stefan Decker, and Axel Polleres. Weaving the pedantic web. In Christian Bizer, Tom Heath, Tim Berners-Lee, and Michael Hausenblas, editors, *Proceedings of the WWW2010 Workshop on Linked Data on the Web, LDOW 2010, Raleigh, USA, April 27, 2010*, volume 628 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010. URL http://ceur-ws.org/Vol-628/ldow2010_paper04.pdf.

[17] Aidan Hogan, Jürgen Umbrich, Andreas Harth, Richard Cyganiak, Axel Polleres, and Stefan Decker. An empirical survey of linked data conformance. *Journal of Web Semantics*, 14:14–44, 2012. DOI https://doi.org/10.1016/j.websem.2012.02.001.

[18] Filip Ilievski, Wouter Beek, Marieke van Erp, Laurens Rietveld, and Stefan Schlobach. LOTUS: adaptive text search for Big Linked Data. In Harald Sack, Eva Blomqvist, Mathieu d'Aquin, Chiara Ghidini, Simone Paolo Ponzetto, and Christoph Lange, editors, *The Semantic Web. Latest Advances and New Domains - 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 - June 2, 2016, Proceedings*, volume 9678 of *Lecture Notes in Computer Science*, pages 470–485. Springer, 2016. DOI https://doi.org/10.1007/978-3-319-34129-3_29.

[19] Tobias Käfer, Ahmed Abdelrahman, Jürgen Umbrich, Patrick O'Byrne, and Aidan Hogan. Observing linked data dynamics. In Philipp Cimiano, Óscar Corcho, Valentina Presutti, Laura Hollink, and Sebastian Rudolph, editors, *The Semantic Web: Semantics and Big Data, 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. Proceedings*, volume 7882 of *Lecture Notes in Computer Science*, pages 213–227. Springer, 2013. DOI https://doi.org/10.1007/978-3-642-38288-8_15.

[20] Dimitris Kontokostas, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri. Test-driven evaluation of linked data quality. In Chin-Wan Chung, Andrei Z. Broder, Kyuseok Shim, and Torsten Suel, editors, *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014*, pages 747–758. ACM, 2014. DOI https://doi.org/10.1145/2566486.2568002.

[21] Nikola Ljubesic, Nives Mikelic, and Damir Boras. Language indentification: How to distinguish similar languages? In Vesna Luzar-Stiffler and Vesna Hijuz Dobric, editors, *Proceedings of the ITI 2007, 29th International Conference on Information Technology Interfaces, Cavtat/Dubrovnik, Croatia, June 25-28, 2007.*, pages 541–546. SRCE, University of Zagreb, 2007. DOI https://doi.org/10.1109/ITI.2007.4283829.

[22] Pablo N. Mendes, Hannes Mühleisen, and Christian Bizer. Sieve: Linked data quality assessment and fusion. In Divesh Srivastava and Ismail Ari, editors, *Proceedings of the 2012 Joint EDBT/ICDT Workshops, Berlin, Germany, March 30, 2012*, pages 116–123. ACM, 2012. DOI https://doi.org/10.1145/2320765.2320803.

[23] Paulo Oliveira, Fátima Rodrigues, and Pedro Rangel Henriques. A formal definition of data quality problems. In Felix Naumann, Michael Gertz, and Stuart E. Madnick, editors, *Proceedings of the 2005 International Conference on Information Quality (MIT ICIQ Conference), Sponsored by Lockheed Martin, MIT, Cambridge, MA, USA, November 10-12, 2006*. MIT, 2005. URL http://mitiq.mit.edu/iciq/iqdownload.aspx?ICIQYear=2005&File=AFormalDefinitionofDQProblems.pdf.

[24] Giovanni Tummarello, Richard Cyganiak, Michele Catasta, Szymon Danielczyk, Renaud Delbru, and Stefan Decker. Sig.ma: Live views on the Web of Data. *Journal of Web Semantics*, 8(4):355–364, 2010. DOI https://doi.org/10.1016/j.websem.2010.08.003.

[25] Jan Wielemaker, Wouter Beek, Michiel Hildebrand, and Jacco van Ossenbruggen. ClioPatria: A SWI-Prolog infrastructure for the Semantic Web. *Semantic Web*, 7(5):529–541, 2016. DOI https://doi.org/10.3233/SW-150191.