Undefined 0 (0) 1
IOS Press

Literally Better: Analyzing and Improving the Quality of Literals

Wouter Beek [a] Filip Ilievski [a] Jeremy Debattista [b] Stefan Schlobach [a] Jan Wielemaker [a]
[a] *VU University Amsterdam*
e-mail: {w.g.j.beek,f.ilievski,k.s.schlobach}@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.

1. Introduction

In this article 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 (1) they provide a concise notation for large (and possibly infinite) values spaces and (2) 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. First, we create a toolchain that allows billions of literals to be analyzed. The toolchain is made available as Open Source code to the community. We show that the toolchain is easy to integrate into existing 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. Thirdly, the toolchain is used by Luzzu: a state-of-the-art Linked Data quality framework. In addition to presenting, implementing and integrating this toolchain, we also illustrate how it may be used to perform analyses of the quality of literals. Finally, based on the previous step, we present automated procedures and concrete suggestions for improving the quality of literals in today's Web of Data. The automated procedures are implemented and evaluated in order to show that our approach and toolchain are well-suited for running on a Web scale.

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

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 following 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. 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 [27]. Not all categories are applicable to Linked Data quality. For instance, due to a fluid schema and the Open World assumption, the absence of an RDF property assertion does not imply a missing value. Because RDF does not enforce the Unique Names Assumption the problem of value uniqueness does not arise. Other data quality categories however 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).

The large-scale aspects of Linked Data quality have been quantified in the so-called 'LOD Observatory' studies: [19,20,14,2]. These studies have not included anything but a cursory analysis of RDF literals. In [21], Hogan et al. conduct an empirical study on Linked Data conformance, assessing datasets 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. Debattista et al. [14] have conducted an empirical study to analyze, quantify, and understand the quality of how well data is represented on the Web of Data. For this study, the authors assessed a number of datasets that are present in the LOD Cloud [2] against metrics classified under the "Representational Category" in [33].

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 dataset for general 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 for such a vocabulary. There are already several data quality vocabularies that can be extended, e.g. [13,17]. A W3C Working Group is currently developing a standard vocabulary (DQV) for expressing Linked Data quality[3].

A number of tools have been developed for assessing the quality of Linked Datasets [23,7,12,26]. The authors in [23] 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 [26], 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 and scalable 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.

Some aspects of quality are highly subjective and cannot be determined by automated means. In order to improve these quality aspects a human data curator has to edit the data. [1] present a crowd sourcing 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 [24].

[2] `http://lod-cloud.net`

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

The paper focus on only 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.

3. Motivation

Literals are an important syntactic and semantic component of the Semantic Web's data model. Their first benefit is that they allow a concise notation for infinite value spaces. One of the Linked Data principles is "Use URIs as names for things" [6]. This principle is carried through to the absurd in [31] where the authors jokingly present the Linked Open Numbers dataset in which IRIs are minted for natural numbers. The Linked Open Numbers dataset shows that the Linked Data principle of assigning names to everything does not scale, and does not make much sense, for infinite value spaces. In addition, *relations* between the values of infinite value spaces are better expressed intensionally (by a limited number of functions) than extensionally (by explicitly asserting an infinite number of pairs). 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 [34] 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 in addition to RDF's graph-based data model. While IRIs are (also) intended to be human-readable [15], 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 specifiers are particularly important in order to ease the human processability of information in different languages.

Assessing the quality of literals for each dataset is an important ingredient for assessing the overall quality of a dataset. Specifically, indicators of literal quality can be fed into Luzzu [12], a state-of-the-art quality assessment framework. Secondly, a LOD Cloud-wide overview indicates what are current problems for the consumption of literals and can give informed information about the areas where data publication practices can be improved. Specifically, quantified quality indicators can be used in order to improve the cleaning process of the LOD Laundromat [4], a price winning data cleaning and republishing Web Service. We believe that it is best to base tomorrow's tooling, training, best practices and standards on an empirical overview of today's problems.

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 then compare them according to the comparator operators that are defined for that value space. 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, i.e., character-for-character, string equality. Most defined datatypes that are used in RDF specify a canonical representation for each value. 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, then many operations can be performed significantly faster. For instance, a SPARQL query `SELECT * WHERE { ?s ?p "2016-01-01T01-01-01Z"^^xsd:dateTime }` can be very efficiently executed if we only need to match the explicit object string. If values are not canonically represented then we have to interpret (and compare to) all date-time values that appear in the data. For all datatypes the use of canonicity makes identity checking and matching more efficient. For many datatypes the use of canonicity makes determining the relative order between literals, i.e., '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 [18]. Having languages tags associated with string literals allows 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" [30], 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 intended formats 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 [22] allow literals, and statements in which they appear, to be retrieved based on 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 (`nl`). Besides language-tagged literals, high-quality datatype information also significantly improves the results of semantic search. For example, it allows the weights 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 thus 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 non-languagetagged literals. 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 $L := (IRI \times LEX) \cup (\{\texttt{rdf:langString}\} \times LEX \times LTAG)$, where IRI is the set of Internationalized Resource Identifiers as per RFC 3987 [16], LEX is the set of Unicode strings in Normal Form C [10], and $LTAG$ is the set of language tags as per RFC 5646 [29]. Literals that are triples are called **language-tagged**

strings LTS. The first element of a literal is called its **datatype IRI**, the second element is called its **lexical form**, and the third element – if present – is called its **language tag**.

According to the RDF 1.1 standard [9], language-tagged strings are treated very differently from other literals. Firstly, they have a datatype IRI `rdf:langString` but no datatype. Secondly, because language tags cannot be expressed as part of a literal's lexical form, language-tagged string have no such lexical form. Language-tagged strings do have values: they are pairs of strings and language tags. Thirdly, all and only literals with datatype IRI `rdf:langString` have a language tag. We use the term **datatyped literal** to refer to a literal with a datatype IRI that may denote a datatype. Only literals whose datatype IRI is `rdf:langString` are not datatyped literals.

RDF serialization formats such as Turtle allow literals to be denoted by only a lexical form. These are abbreviated notations that allow very common datatype IRIs to be inferred based on the lexical form. The datatype IRI that is associated with such a lexical form is determined by the serialization format's specification. For instance, the string `false` in Turtle is an abbreviated form of `"false"^^xsd:boolean`.

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 an interpretation function I that maps RDF names to resources. The set of resources is denoted IR. Let us call the subset of resources that are datatypes D. A datatype IRI i and the datatype d it denotes are related by the interpretation function: $I(i) = d$. Following the XML Schema Datatypes standard [28], 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.
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, not necessarily functional, **value-to-lexical mapping** $v2l_d$ that maps values to lexical forms.
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_{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 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 $c(l2v(\texttt{"red"})) = \texttt{"255,0,0"}$, 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 mapping $IL : L \to IR$ (definition 1). IL is partial because a lexical form may not belong to the datatype's lexical space. 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 IRI.

Definition 1 (Literal value)**.**

$$IL(l) := \begin{cases} lex & \text{if } l = \langle lex \rangle \\ \langle lex, lc(tag) \rangle & \text{if } Cond_A(l) \\ l2v(I(e))(lex) & \text{if } Cond_B(l) \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

where

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

$$\wedge\, I(i) \neq I(\texttt{rdf:langString})$$

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 a 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):

Unsupported A datatyped literal is unsupported if it is either undefined or unimplemented.

Undefined A datatyped literal is undefined if its datatype IRI does not denote a datatype. Formally: $I(i) \notin D$. Whether an IRI denotes a datatype or not is not specified in the RDF standards. We therefore specify this ourselves in the following way: an IRI is defined **iff** dereferencing of the IRI leads to either (i) a machine-processable specification; (ii) a human-readable formal specification; or (iii) a human-readable informal description that can be unambiguously turned into a formal specification. For instance, the XSD datatype IRIs point to the XML Schema 1.1 Part 2: Datatypes specification [28], which includes (i) and (ii). We require the specification to include a lexical space, a value space and mappings between the two (Section 4.2). If a specification is only missing a canonical value-to-lexical mapping we include its literals under the 'non-canonical' category (see below).

Unimplemented A datatyped literal is unimplemented if its IRI denotes a defined datatype but is not implemented by a specific RDF processor. Example: many RDF processors do not implement `xsd:duration`, probably because it does not occur that often and is also somewhat difficult to implement.

Supported A supported datatyped literal has a datatype IRI that denotes a defined and implemented datatype. Formally: $I(i) \in D$. Example: almost every RDF processor supports datatype `xsd:boolean`, probably because it occurs often and is also very easy to implement.

Invalid A supported datatyped literal is invalid if its lexical form cannot be mapped by 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 are valid. Formally: $l2v_d(lex) \in Val_d$. Valid literals are of higher quality than invalid or unsupported 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 people, 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, 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 assuming 2016 denotes the Gregorian year. Another instance of underspecification occurs when natural language content ap-

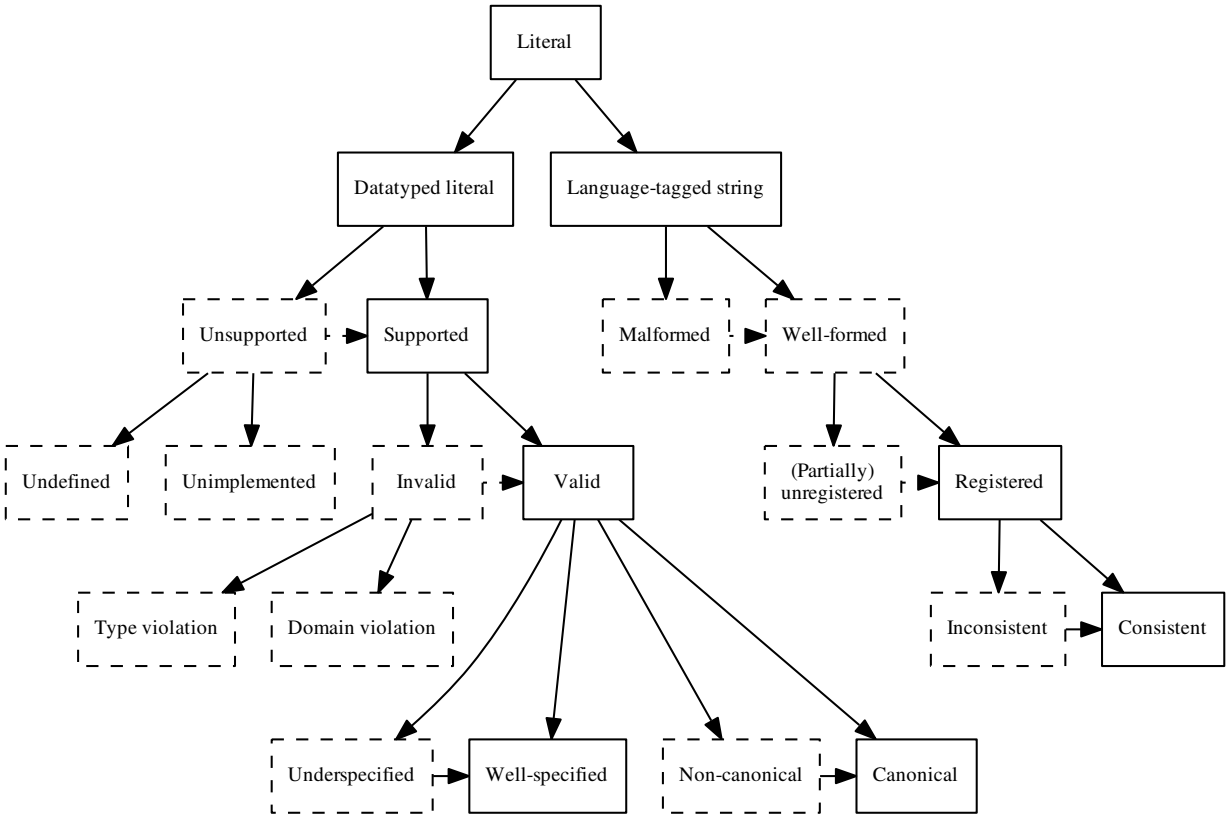

Figure 1. A taxonomy of RDF literal quality. The nodes show the categories a literal's quality can be classified under. Vertical arrows must be interpreted hierarchically. For instance, literals that are 'Invalid' are also 'Supported' and are 'Datatyped literals'. Horizontal arrows denote the possibilities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

pears with datatype XSD string even though a specific language tag could have been used instead. Example: `"semantics"^^xsd:string` can be more descriptively represented as `"semantics"@en`. Another example is `"color"@en` which can be more descriptively represented as `"color"en-US`.

Non-canonical A non-canonical datatyped literal is a valid datatyped literal for which (i) there are multiple ways in which the same value can be represented, and (ii) 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.

Language-tagged strings are sufficiently different from datatyped literals to receive their own quality categories. Specifically, language-tagged strings cannot be undefined because datatype IRI `rdf:langString` is not supposed to denote a datatype. They cannot be unsupported either because every RDF processor must support language-tagged strings in order to be a compliant RDF processor. Finally, their validity 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 string:

Malformed A language-tagged string is malformed if its language tag violates the grammar specified in RFC 5646 [29]. Example: `en-US` is a well-formed language tag, but `en:US` is malformed.

Well-formed A language-tagged string is well-formed if it is not malformed.

(Partially) 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[4]. 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 well-formed and registered, but the word 'affe' belongs to the German language (denoted by `de`) and not the English language (denoted by `en`).

4.4. Dimensions of measurement

In addition to the quality categories distinguished above, the quality of literals can also be measured across the following two dimensions:

Granularity dimension The granularity level at which literal quality is quantified. We distinguish at least the following three granularity levels:

1. **Term level** The quality of individual literals.
2. **Datatype level** The quality of literals that have the same datatype.
3. **Document level** The quality of literals that appear in the same document.

Implementation dimension Since every empirical measurement of literal quality much 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 the Linked Data standards.

All quality categories can be measures at each of the three granularity levels. Category 3 (unimplemented) is measured along the implementation dimension.

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

4.5. Quality metrics

Quality metrics can be defined in terms of the quality categories (Section 4.3) and dimensions of measurement (Section 4.4). Measurements on the literal level are straightforward: every literal is assessed to belong to one of the quality categories in Figure 1. For some purposes the differentiation over quality categories provides too much detail. For instance, an application may only be interested in whether a literal is sufficiently compliant or not. In such cases the quality categories can be clustered. For instance, Luzzu considers literals of sufficient quality when they are valid (category 'Valid') and invalid otherwise. Measurements on the datatype and document level can be performed by calculating the ratio of literals in each of the respective quality categories within the scope of a given datatype or document.

Since Luzzu calculates the quality of data documents, it quantifies literal quality on the document level as well. By default, Luzzu includes two metrics for literal quality. (Since Luzzu is an extendable data quality framework, a Luzzu user may define additional quality metrics for literals as she sees fit.) For the first metric Luzzu taken the total number of literals in a document $T_{literals}$ together with the total number of valid literals (category 'Valid') $T_{correctLiterals}$. Luzzu calculates the *defined literals quality metric* (Q_{dt}) as follows:

Definition 2. $Q_{dt} = \frac{T_{correctLiterals}}{T_{literals}}$

For the second literal quality metric Luzzu takes the number of string literals in a document $|LTS_D|$ together with the total number of language-tagged strings that have the correct language tag (category 'Consistent') assigned $|LTS_{D,correct}|$. Luzzu calculates the *correct language tag quality metric* (Q_{lt}) as follows:

Definition 3. $Q_{lt} = \frac{|LTS_{D,correct}|}{|LTS_D|}$

The here introduced literal quality metrics are used in the two 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 as well as the evaluation of the improvement modules is conducted over the LOD Laundromat [4] data collection, which currently consists of about 650K data documents and 38 billion ground statements. The data is collected from data catalogs (e.g., Datahub) and contains datasets that users have uploaded through the Web API[5]. 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 locate whether a literal term is present and, if so, where it occurs. For example, the syntactically malformed line [0] inside a Turtle-family document may be fixed to a triple [1], a quadruple [2] or two triples [3]. The right fix cannot be determined by automated means.

```
[0]   <a> <b> "c, d>    .
[1]   <a> <b> <c,d>     .
[2]   <a> <b> "c,"  <d> .
[3]   <a> <b> "c" , <d> .
```

The absence of well-formed literals that appear within 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: $I(\langle s, p, o\rangle) = 1 \Leftrightarrow \langle I(s), I(o)\rangle \in IExt(I(p))$, where I is the interpretation function and $IExt$ is the extension function mapping resources (called 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.

5.2. Tooling

While all LOD Laundromat data can also be accessed through Open Web APIs, we have used the following dedicated tools that have been developed to support bulk processing and running large-scale experiments over LOD Laundromat data. All tools are (of course) published as Open Source software and/or as Web Services to the community.

ClioPatria[6] [32] is a Prolog-based triple store and RDF library implemented in SWI-Prolog[7]. 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.

We have compared the results of our datatype implementation in *ClioPatria* with *RDF4J*[8] [8], another state-of-the-art 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 specification [28]. While we have implemented several other often occurring datatypes such as `dct:W3CDTF` and `dct:RFC4646` as well, it is not easy to check our implementation's correctness since very few other implementations of these datatypes exist.

The **LOD Laundromat Washing Machine**[9] [4] is the Linked Data data cleaning mechanism that powers the LOD Laundromat ecosystem. The analysis of datatyped literals (Section 6.1) is directly implemented into the LOD Laundromat Washing Machine. This means that all literals published by the LOD Laundro-

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

[6] See `https://github.com/ClioPatria/ClioPatria`

[7] See `http://www.swi-prolog.org`

[8] See `http://rdf4j.org/`

[9] See `https://github.com/LOD-Laundromat/Washing-Machine`

mat are now guaranteed to be valid and canonical (if a canonical mapping exists).

The LOD Laundromat Washing Machine cleans the data in a stream of tuples. This means that memory consumption is almost negligible. The grammars of the implemented datatypes are all LL(1) grammars: 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: there are 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 [3] is a command-line tool that allows data from the LOD Laundromat data collection to be streamed at the level of singular triple pattern fragments. This tool is used in the analysis of the quality of language-tagged strings (Section 6.2) and in the analysis of the quality of data documents (Section 6.3). *Frank* is also used in the automated improvement of language tags (Section 7.2).

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

1. **Apache Tika** - We use a NodeJS wrapper[10] for the 1.10 version of Apache Tika[11]. 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 with short texts. Tika can detect 18 languages (17 languages with European origin and Thai language).
2. **CLD (Compact Language Detection)** - The NodeJS CLD library[12], which is built on top of Google's CLD2 library[13]. 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.
3. **Language-detection**[14] (abbreviated as LangDetect or LD) is a library developed by Nakatani Shuyo in Java. A commonly used plugin for language detection in ElasticSearch[15] is based on this library. This library uses 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 reportedly widely used (for e.g. in ElasticSearch) and characterized with remarkable accuracy on the supported languages and text sizes. Although the chosen set still remains – to some extent – arbitrary, note that it is trivial to include more libraries as one sees fit.

Luzzu[16] [12] is a quality assessment framework for Linked Data. The rationale of Luzzu is to provide an integrated platform that: (1) assesses Linked Data quality using a library of generic and user-provided domain specific quality metrics in a scalable manner; (2) provides query-able quality metadata on the assessed datasets; (3) assembles detailed quality reports on assessed datasets. Furthermore, Luzzu aims to create an infrastructure that:

- can easily be extended by users by defining custom, domain-specific metrics;
- implements quality-driven dataset ranking algorithms facilitating use-case driven discovery and retrieval.

[10] https://github.com/ICIJ/node-tika
[11] http://tika.apache.org/1.10/index.html
[12] https://github.com/dachev/node-cld
[13] https://github.com/CLD2Owners/cld2
[14] https://github.com/shuyo/language-detection
[15] https://github.com/jprante/elasticsearch-langdetect
[16] Sources: https://github.com/EIS-Bonn/Luzzu; Website: http://eis-bonn.github.io/Luzzu/

The following literal-specific quality metrics are implemented within the Luzzu framework[17]: (i) to assess the validity of a datatype against their lexical value; and (ii) to assess the correctness of a string's language tag.

For calculating the *correct language tag quality metric)* (Section 4.5) Luzzu uses natural language Web Services. For single word literals it uses the Lexvo [11] Web Service[18]. Lexvo is an enriched linguistics knowledge base that interconnects entities to each other (for example different meanings and translations of a word) and also to entities on the Web of Data. Given a string literal with its corresponding tag, a request to the Lexvo API is made and a dereferenceable RDF resource 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 the language tag of muli-word literals, Luzzu uses the Xerox Language Identifier[19], a Web Service that identifies the language of natural language phrases and sentences. We experimented with the service for its identification correctness by providing various sentences in English, Italian and German. The service always returned the correct identification. On the other hand, this cannot be considered as a guarantee for all languages. In fact, the authors in [25] state that the service's identification correctness is not guaranteed for certain languages. The result of this metric is dependent of these two services, therefore we consider this metric to give us an estimate.

6. Analysis

This section present three analysis 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 string within documents.

6.1. Analysis 1: The quality of datatyped literals

We analyze 1,457,568,017 datatyped literals from the LOD Laundromat data collection for the following datatyped literal quality categories defined in Section 4.3: undefined, invalid, non-canonical. Overall we find that the vast majority of literals are valid and a modest majority of them are also canonical (Figure 1). However, 79% (or 1,108,813,673 occurrences) of literals are of type XSD string. This is not surprising since this is the datatype that enforces the least syntactic restrictions and is also the one that is chosen in many serializations and applications as the default datatype. For an XSD string literal to be invalid it must contain non-visual characters such as the ASCII control characters without escaping. Since this does not occur very often, it is not surprising that the overall percentage of valid datatyped literals is also quite high. However, for the more complex datatypes that denote dates, times and floating point numbers, the lexical forms are stricter and must follow a very particular grammar. 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 implemented them. An example of this is datatype IRI `sysont: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. For comparison, the value space for `rdf:XMLLiteral` is defined in terms of the XML DOM 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 Markdown.

Listing 1: Informal description of Markdown datatype.

```
sysont:Markdown a rdfs:Datatype ;
  rdfs:comment "A string literal formated using
```

[17] All metric implementations are external to Luzzu, therefore the results obtained are through the metrics imported to the framework.

[18] See `http://lexvo.org`

[19] See `https://services.open.xerox.com/bus/op/LanguageIdentifier/GetLanguageForString`

Datatype IRI	Occurrences
dt:second	2,326,298
dt:minute	682,790
dt:squareKilometre	643,493
dt:centimetre	382,281
dt:kilogram	356,321

Table 1
The most often occurring undefined datatyped literals.

Datatype IRI	Occurrences
xsd:int	511,741
xsd:decimal	122,738
xsd:dateTime	98,505
xsd:gYearMonth	16,469
xsd:gYear	11,957

Table 2
The datatype IRIs with the highest number of invalid literals.

```
              markdown syntax." ;
rdfs:label   "Markdown formated string" ;
rdfs:seeAlso "http://daringfireball.net/
              projects/markdown/syntax" .
```

Datatype IRI	Occurrences
xsd:float	30,152,304
xsd:double	17,783,414
xsd:decimal	2,127,133
rdf:XMLLiteral	245,457
xsd:dateTime	224,994

Table 3
The datatype IRIs with the highest number of non-canonical literals.

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 a single informal descriptions that is unambiguous and complete. Table 1 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. Literals that use these datatypes can therefore never end up in the canonical quality category. An example of such a datatype IRI is `dct:W3CDTF`, which allows multiple lexical forms to denote the same value. For instance, the same time zone can be represented in multiple ways: as `+01:00` or as `-23:00`. We note 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 whitespace use, encoding decisions, canonization of non-HTML content like CSS or JavaScript, etc).

Invalid Table 2 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 data gets more complicated, the percentage of invalid occurrences goes up.

Non-canonical Table 3 shows the eight 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 with exponent sign 'E'. For instance, the floating point number `1.0` must be canonically written as `1.0E0`.

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

We analyze language tags for 569,422 documents from the LOD Laundromat data collection. This set contains 3.54 billion literals whose lexical form contains natural language expressions. 2.26 billion literals (63.83%) are language-tagged strings that have a language tag specified by the original data creator. This means that 36.17% of the LOD Laundromat natural language literals do not have a language tag. The distribution of language tags is given in Table 4. By far the most language-tagged literals are English, followed by German, French, Italian and Spanish. This

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	607,012,252
no tag	1,281,785,207
text literals	3,544,028,391
number of datasets	569,422

Table 4
The distribution of language tags in the LOD Laundromat data collection.

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 literal belong to one of the 10 most frequently occurring languages.

6.3. Analysis 3: The quality of LOD documents

In order to illustrate that the here presented toolchain integrates well with existing 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 calculated the *compatible datatype metric* and the *correct language tag usage metric* (Section 4.5). For these specific documents Luzzu determined that, on average, 70% of the RDF string literals in LOD Laundromat have a correct language tag. On the other hand, only 39% of RDF literals have a compatible datatype.

We inspected a sample of documents whose quality value was less than 40% for quality aspects that cannot be automatically calculated. We observe the following low-level quality problems with the incorrect literals:

- A literal contains linguistic content but lacks a language tag.
- A literal partially contains linguistic content but also contains non-alphabetic characters. Example: `"related_software"@en`.
- A literal has linguistic content but also contains syntax errors. Example: `"flow cytometer sorter"@en`.
- A literal has an unrecognized language tag. Example: `"article"@en-US`.

The majority of problematic triples fall into the first category. The third and fourth categories are the most interesting. The former category deals with literals that were identified incorrect by the metric's external service due to some language syntactic flaw. For example, in `"flow cytometer sorter"@en` the term *cytometer* should have been written as *cytometry*. The final category deals with the actual syntax expected of the language tag. Although the tag `en-US` is correctly defined as per the BCP47 standard [29], our metric expects two/three letter language tag as defined by the Linked Languages Resources[20].

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 Sections 5 and 6. The possible quality improvements are defined in the literal quality taxonomy in Section 4, as indicated by the horizontal lines 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 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.

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

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 efficient computation for equivalence tests by automatically converting non-canonical datatyped literals to canonical ones. The second one is improved natural language processing by automatically assigning language tags to textual literals that did not have language tags before.

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 IRI would solve the vast majority undefined datatype IRIs, thereby significantly increasing the 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 forms alone. For instance, the fact that the values "0001", "0203", "9009" appear in the data does not tell us whether the datatype should be defined in terms of `xsd:positiveInteger` or in terms of `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 could map lexical form "0001" to the Mount Everest and "0203" to afternoon sunsets. (While this example may seem extreme, there are library standards where 3- or 4-digit numbers are used to denote concepts and topics.)

Unimplemented → implemented Unimplemented literals can only be improved upon by adding support for them in a particular RDF processor. At the moment, only few datatypes beyond the XSD primitive types are implemented by any processor. Section 6 shows that it is not easy to implement most datatypes since many of them are underspecified. This may point to a lacuna in today's RDF 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. Also, after more than a decade most RDF tooling has not implemented means of interpreting datatypes defined in XML Schema. In order to improving the current situation, assistance in defining RDF datatypes must be significantly improved. Also, 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.

Invalid → valid Invalid literals can only be improved upon by the original data publisher. We cannot automate this task since it requires us to choose between changing the datatype IRI to match the lexical form, changing the lexical form to match the datatype IRI, or changing both. We can however give a list of mistakes that occur most often. Based on our empirical observations these are the top 5 mistakes, along with suggestions of how to avoid them:

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.
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 IRI specification [16]. Most of these improper datatype IRIs are due to undeclared prefixes appearing in the source document. Most of these can probably be expanded according to list of common RDF 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, i.e., string similarity suffices where one would have to calculate $v2c(l2v(l_1)) \equiv v2c(l2v(l_2))$ otherwise. The improvement of non-canonical literals is

conditional on other quality improvements. Firstly, the datatype has to specify a canonical mapping. Secondly, there has to be a tool that implements this mapping. Only when these two preconditions are met is it possible to algorithmically generate canonical out of non-canonical literals. ClioPatria now fully automates the canonicalization of RDF literals: all literals are stored in canonical form upon statement assertion.

7.2. Improving language-tagged strings

No language tag → language tag We attempt to assign a language tag to textual literals without one. For this purpose, we test language detection improvements on textual lexical form from the documents of the LOD Laundromat data collection. We define a textual lexical form as a lexical form of datatype `xsd:string` or `xsd:langString` which has at least two consecutive Unicode letters.

In an attempt to improve the language tags coverage of LOD Laundromat, we apply three language detection libraries (Section 5.2). 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? Can we combine the libraries and thus improve the accuracy of language detection?

In our experiments, we primarily focus on the set of tagged literals. These contain a "golden" language tag, which allows us to easily evaluate the accuracy of our solutions. For the set of tagged literals, we report the precision, recall and F1-value of each of the language detection libraries. We assume that the accuracy of the language detection on the language-tagged strings is comparable to the accuracy on textual lexical forms with no user-defined language tag.

The language tag assigned by the users and by the automatic detection libraries can be of arbitrary length or complexity. Since our language detection tools provide an ISO 639-2 two-character language code in most of the cases, we focus our comparison on the initial two characters of each language-tagged string. This granularity of comparison is satisfactory for most cases, although in exceptional situations the secondary language tag can also describe the language. This is the case for Chinese languages where `zh-cn` denotes a different language than `zn-tw`.

We present the time needed to annotate the non-tagged language expressions from the LOD Laundromat collection for each of the library in Table 5. According to this Table, the process of tagging every non-tagged lexical forms lasts around 8 days. Most of this elapsed time (around 90% of the total time) is an overhead accounted to the library algorithm itself. Considering that such an improvement procedure is to be executed once and not necessarily in real time, we believe that the period needed to improve the coverage of language tags in the LOD Cloud is reasonable.

We measure the accuracy of each ALD library and observe that the highest precision (75.42%) is achieved by the CLD library, which covers highest number of languages (160). It is notable that this library often gives no language suggestion, especially when it comes to short strings. 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 6. Each of the cells in the Table represents an intersection of a primary language tag and a string size bucket.[21] Each cell contains three values that show the F1-accuracy of each of the three libraries: Tika, CLD and LangDetect, correspondingly. While CLD has highest accuracy in the majority of the cells, there are notable exceptions. For instance, the LangDetect library is mostly better than the CLD library in detecting English literals or short Portuguese literals.

Figure 2 shows the aggregated accuracy per bucket for each of the libraries. Note that there is hardly any intersection of the plotted lines: for any bucket of text size (except 0), CLD has the highest F1-value, while Tika has 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

[21] We measure the string size N in terms of number of words that constitute the string. A string size bucket B contains a range of string sizes whose logarithmic value shares the same natural number: $B = \lfloor log_2(N) \rfloor$

Library	Duration	Library Overhead
No library	21.5 hours	/
CLD	182.5 hours	161 hours
LD	203 hours	181.5 hours
Tika	206 hours	184.5 hours

Table 5
Running time for each of the three ALD libraries

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 an average F1-value over the three libraries for a given language and bucket. Libraries can successfully detect the language of sufficiently long literals (bucket 3 literals already have an F-measure of around 75%, growing to around 85-90% for bucket 4). Almost all languages, except from Portuguese, closely follow this distribution.

Guided by these insights, we combine the libraries by applying the library with the highest F1-score for each pair of language tag and bucket size. By following this approach, we obtain an F1-value of 50.24%. This score is far below the reported accuracy of the employed libraries, which is an artifact of the averaging over all strings in the LOD cloud, irrespective of their length and language. As the Figures 2 and 3 demonstrate, the accuracy of the ALD libraries for strings of moderate and long size is very high, whereas the language detection on short strings and non-supported languages has a much lower accuracy. Hence, to guarantee very high precision and recall of language detection, the improvement of language tags should focus on strings with sufficient length, i.e. avoid to detect the language of short strings.

8. Conclusions

We have presented a toolchain for the large-scale analysis and improvement of the quality of RDF literals. The toolchain is build on top of the LOD Laundromat data cleaning and republishing architecture. The toolchain has not only be used for the here presented experiment, but has also been consolidated in the ClioPatria triple store. It has also been reused by the Luzzu quality framework. Literal quality indicators have been fully integrated into the LOD Laundromat and metadata about each literal quality violation is now recorded next to other cleaning metadata.

We have focused on the quality of literals, an area of Linked Data quality 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. Our toolchain is able to implement such quality metrics, allowing data quality to be systematically analyzed. Two analyses were conducted on a very large scale and a third analysis has shown that out toolchain can be used by existing quality assessment frameworks as part of assessing overall Linked Data quality.

We have also illustrated a new way for the Semantic Web and Linked Data communities to discover those areas where literal quality would be most effectively improved, because it is now possible to quantify the impact of data cleaning, and other quality improvement attempts.

Finally, we have shown that it is possible to improve the quality of millions of literals (and thereby statements) very quickly, by algorithmic means. This does of course not apply to every quality criterion, e.g., it does not apply to subjective criteria or ones that require the original data publisher to edit the data. We have shown that, when given state-of-the-art algorithms in – for instance – natural language identification, our toolchain is able to improve the overall quality of the LOD Cloud. This means that quality criteria that can be automatically improved *in theory* can now also be automatically improved *in practice*.

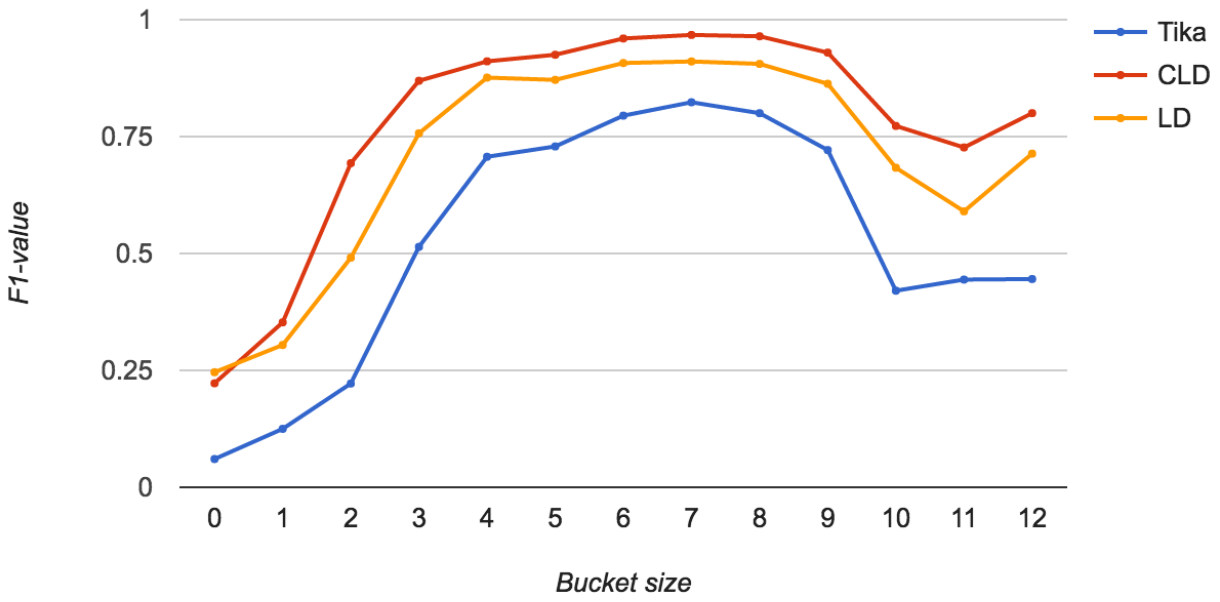

Figure 2. Accuracy per language detection library

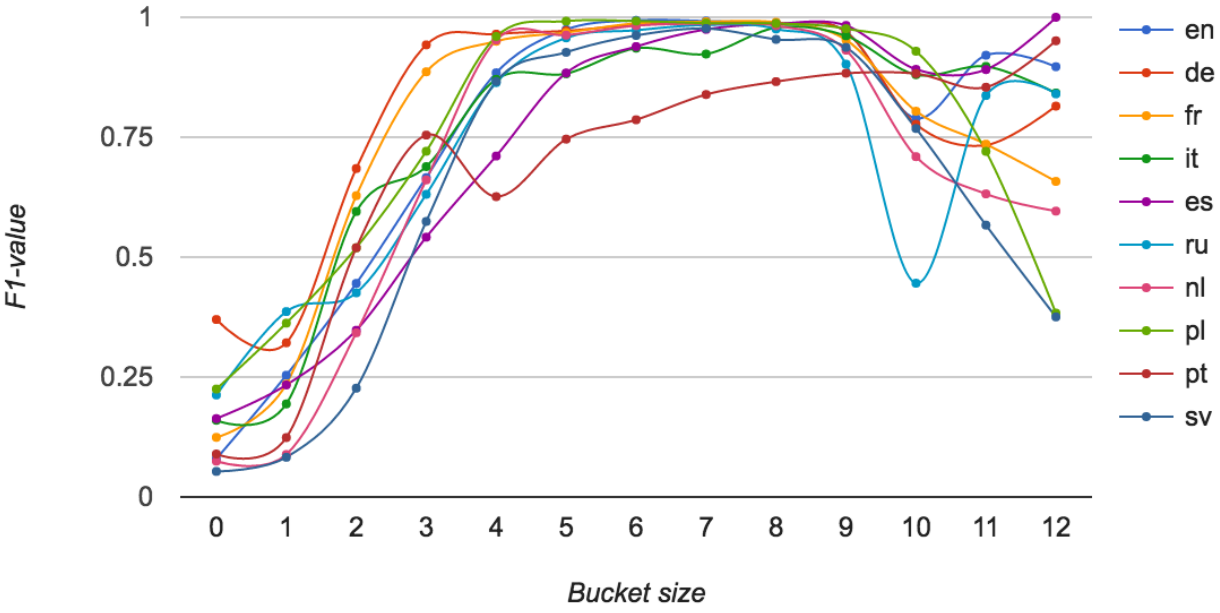

Figure 3. Accuracy per language tag for the 10 most frequent languages

	0	1	2	3	4	5	6	7	8	9	10	11	12
en	2.08	5.72	13.84	38.24	76.29	94.7	98.77	98.77	98.06	95.28	63.75	86.76	83.01
	12.5	26.46	42.19	70.17	92.44	98.44	99.6	99.4	98.94	97.78	83.56	95.35	92.34
	9.29	43.98	77.57	91.32	96.54	99.18	99.7	99.48	99.04	97.95	89.5	94.14	93.73
de	5.26	15.44	47.56	86.94	93.03	94.99	97.25	98.18	97.75	94.77	64.75	43.7	59.26
	53.17	46.4	86.66	98.57	98.94	98.79	99.35	99.36	99.41	98.53	84.89	90.76	92.59
	52.57	34.6	71.19	97.23	97.49	97.73	98.65	98.81	98.66	97.59	83.46	85.59	92.59
fr	8.65	15.03	39.04	82.44	90.85	95.24	98.54	98.98	98.79	94.36	76.71	70.72	58.73
	20.45	33.5	77.99	94.95	98.11	98.42	99.09	99.22	99.14	95.88	82.98	77.17	71.43
	8.18	22.07	71.41	88.52	96.07	96.81	98.6	99.02	98.97	95.67	81.55	72.77	67.21
it	14.03	19.63	68.99	71.76	90.19	89.42	93.63	91.81	98.25	96.79	94.73	96.65	87.42
	28.57	30.48	73.25	76.17	91.17	89.41	95.0	92.44	98.44	96.65	94.1	94.06	83.65
	5.38	8.1	36.31	58.67	79.84	85.86	92.06	92.76	96.86	94.99	75.26	78.57	81.58
es	2.51	7.55	21.72	37.72	53.04	75.89	86.31	94.72	97.11	96.66	85.57	86.69	100.0
	32.87	35.09	50.77	70.58	91.94	98.37	99.41	99.46	99.74	99.26	89.98	88.54	100.0
	13.64	27.43	31.78	54.19	68.25	90.91	95.97	98.27	99.09	98.97	91.87	92.06	100.0
ru	26.04	39.99	71.39	64.34	88.56	97.4	97.81	98.43	96.9	85.91	15.69	73.76	83.72
	28.56	47.0	34.95	71.02	93.07	98.92	99.2	99.2	98.41	94.67	79.75	91.96	93.02
	9.2	28.95	21.35	53.92	77.53	90.81	94.87	97.56	97.38	90.1	38.25	85.36	75.36
nl	3.02	4.32	22.52	63.35	96.11	96.9	98.61	98.58	97.78	90.79	59.21	43.86	45.45
	11.69	12.34	41.2	71.7	96.31	97.35	98.71	98.75	98.22	93.31	72.67	66.01	50.0
	7.7	9.9	38.96	63.25	93.34	94.57	97.43	98.66	98.73	95.33	80.96	79.68	83.33
pl	17.91	35.31	50.51	66.62	95.16	99.1	99.19	98.74	98.61	97.58	92.19	71.0	37.5
	30.3	34.91	57.55	73.98	96.55	99.19	99.13	98.66	98.58	97.52	92.09	73.0	37.5
	19.3	38.5	47.8	75.61	96.3	99.25	99.31	98.97	98.93	97.78	94.5	72.13	40.0
pt	3.45	5.34	8.52	59.83	20.3	33.99	39.4	53.72	61.22	67.33	69.42	70.51	85.29
	10.2	14.88	70.93	81.92	84.35	94.67	98.29	98.79	99.1	98.57	96.87	89.74	100.0
	13.23	16.95	76.43	84.58	83.17	95.13	98.22	99.19	99.45	99.14	98.47	96.0	100.0
sv	1.3	2.64	14.87	44.94	76.84	87.54	93.68	96.41	93.21	90.2	64.84	45.24	10.0
	10.22	11.34	28.27	66.67	92.79	96.33	98.28	98.23	94.61	94.24	80.66	57.14	50.0
	4.47	10.87	24.91	60.71	90.25	94.24	96.67	98.36	98.36	96.75	84.97	67.65	52.63

Table 6

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.

References

[1] Maribel Acosta, Amrapali Zaveri, Elena Simperl, Dimitris Kontokostas, Sören Auer, and Jens Lehmann. Crowdsourcing linked data quality assessment. In *The Semantic Web–ISWC 2013*, pages 260–276. Springer, 2013.

[2] Ahmad Assaf, Aline Senart, and Raphaël Troncy. What's up LOD cloud? observing the state of linked open data cloud metadata. In *2nd Workshop on Linked Data Quality*, 2015.

[3] Wouter Beek and Laurens Rietveld. Frank: Algorithmic Access to the LOD Cloud. *Proceedings of the ESWC Developers Workshop 2015*, 2015.

[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 *ISWC 2014*, pages 213–228. Springer, 2014.

[5] Tim Berners-Lee. Cool uris don't change, 1998.

[6] Tim Berners-Lee. Linked Data, 2006.

[7] Christian Bizer and Richard Cyganiak. Quality-driven information filtering using the wiqa policy framework. *Web Semant.*, 7(1):1–10, January 2009.

[8] Jeen Broekstra, Arjohn Kampman, and Frank Van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF Schema. In *The Semantic Web—ISWC 2002*, pages 54–68. Springer, 2002.

[9] Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 concepts and abstract syntax, 2014.

[10] Mark Davis and Ken Whistler. Unicode normalization forms, August 2012.

[11] Gerard de Melo. Lexvo. org: Language-related information for the linguistic linked data cloud. *Semantic Web*, page 7, 2013.

[12] Jeremy Debattista, Sören Auer, and Christoph Lange. Luzzu - A framework for linked data quality assessment, 2016.

[13] Jeremy Debattista, Christoph Lange, and Sören Auer. Representing dataset quality metadata using multi-dimensional views. In *SEMANTiCS*, 2014.

[14] Jeremy Debattista, Christoph Lange, and Sören Auer. Are lod datasets well represented? a data representation quality survey.

Under Review, 2016.

[15] M. Duerst and M. Suignard. Internationalized Resource Identifiers, January 2005.

[16] Martin J. Dürst and M. Suignard. Internationalized resource identifiers (iris), January 2005.

[17] 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. ACM, 2011.

[18] Bernardo Cuenca Grau, Zlatan Dragisic, Kai Eckert, Jérôme Euzenat, Alfio Ferrara, Roger Granada, Valentina Ivanova, Ernesto Jiménez-Ruiz, Andreas Kempf, Patrick Lambrix, et al. Results of the ontology alignment evaluation initiative 2013. In *International Workshop on Ontology Matching, collocated with the 12th International Semantic Web Conference-ISWC 2013*, pages pp–61, 2014.

[19] A. Hogan, A. Harth, A. Passant, S. Decker, and A. Polleres. Weaving the pedantic web. In *Linked Data on the Web Workshop (LDOW2010) at WWW'2010*, 2010.

[20] Aidan Hogan, Jürgen Umbrich, Andreas Harth, Richard Cyganiak, Axel Polleres, and Stefan Decker. An Empirical Survey of Linked Data Conformance. *Web Semantics: Science, Services and Agents on the World Wide Web*, 14:14–44, 2012.

[21] Aidan Hogan, Jürgen Umbrich, Andreas Harth, Richard Cyganiak, Axel Polleres, and Stefan Decker. An empirical survey of linked data conformance. *Web Semantics: Science, Services and Agents on the World Wide Web*, 14:14–44, 2012.

[22] Filip Ilievski, Wouter Beek, Marieke van Erp, Laurens Rietveld, and Stefan Schlobach. Lotus: Linked open text unleashed. In *COLD workshop, ISWC*, 2015.

[23] Dimitris Kontokostas, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri. Test-driven evaluation of linked data quality. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 747–758, New York, NY, USA, 2014. ACM.

[24] Tobias Käfer, Ahmed Abdelrahman, Jürgen Umbrich, Patrick O'Byrne, and Aidan Hogan. Observing linked data dynamics. In Philipp Cimiano, Oscar Corcho, Valentina Presutti, Laura Hollink, and Sebastian Rudolph, editors, *The Semantic Web: Semantics and Big Data*, volume 7882 of *Lecture Notes in Computer Science*, pages 213–227. Springer Berlin Heidelberg, 2013.

[25] N. Ljubesic, N. Mikelic, and D. Boras. Language indentification: How to distinguish similar languages? In *Information Technology Interfaces, 2007. ITI 2007. 29th International Conference on*, pages 541–546, June 2007.

[26] Pablo N. Mendes, Hannes Mühleisen, and Christian Bizer. Sieve: Linked data quality assessment and fusion. In *Joint EDBT/ICDT Workshops*. ACM, 2012.

[27] Paulo Oliveira, Fátima Rodrigues, and Pedro Rangel Henriques. A formal definition of data quality problems. In *IQ*, 2005.

[28] David Peterson, Shudi (Sandy) Gao, Ashok Malhotra, C.M. Sperberg-McQueen, and Henry S. Thompson. XML Schema Definition Language (XSD) 1.1 part 2: Datatypes, April 2012.

[29] A. Phillips and M. Davis. Tags for identifying languages, September 2009.

[30] Giovanni Tummarello, Richard Cyganiak, Michele Catasta, Szymon Danielczyk, Renaud Delbru, and Stefan Decker. Sig.ma: Live views on the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(4):355–364, 2010.

[31] Denny Vrandečić, Markus Krötzsch, Sebastian Rudolph, and Uta Lösch. Linked Open Numbers. In *Proceedings of RAFT 2010*, 2010.

[32] Jan Wielemaker, Wouter Beek, Michiel Hildebrand, and Jacco van Ossenbruggen. ClioPatria: A logical programming infrastructure for the Semantic Web. *Semantic Web Journal*, page to appear, 2015.

[33] Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann, and Sören Auer. Quality assessment for linked data: A survey. *Semantic Web Journal*, 2015.

[34] Dan Zuras, Mike Cowlishaw, Alex Aiken, Matthew Applegate, David Bailey, Steve Bass, Dileep Bhandarkar, Mahesh Bhat, David Bindel, and Sylvie Boldo et al. IEEE standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–70, Aug 2008.