The Problem with XSD Binary Floating Point Datatypes in RDF

Jan Martin Keil[0000−0002−7733−0193] and Merle Gänßinger[0000−0003−4481−069X]

Heinz Nixdorf Chair for Distributed Information Systems,
Institute for Computer Science, Friedrich Schiller University Jena, Jena, Germany
{jan-martin.keil,merle.gaenssinger}@uni-jena.de

Abstract. The XSD binary floating point datatypes are regularly used for precise numeric values in RDF. However, the use of these datatypes for knowledge representation can systematically impair the quality of data and, compared to the XSD decimal datatype, increases the probability of data processing producing false results. We argue why in most cases the XSD decimal datatype is better suited to represent numeric values in RDF. A survey of the actual usage of datatypes on the relevant subset of the December 2020 Web Data Commons dataset, containing 19 453 060 341 literals from real web data, substantiates the practical relevancy of the described problem: 29 %–68 % of binary floating point values are distorted due to the datatype.

Keywords: Data Quality · Datatypes · Floating Point Numbers · Knowledge Graphs · Numerical Stability · RDF · XSD

1 Introduction

The Resource Description Framework (RDF) is the fundamental building block of knowledge graphs and the Semantic Web. In RDF, values are represented as literals. A literal consists of a lexical form, a datatype, and possibly a language tag. The RDF standard [1] recommends to use XML Schema Definition Language (XSD) built-in datatypes [2]. For numeric values, this includes the primitive types *decimal*, *double* and *float* as well as all variations of integer[1] which are derived from decimal.

The datatype decimal allows the representation of numbers with arbitrary precision, whereas the datatypes float and double allow the representation of binary floating point values of limited range and precision [2]. However, in practice, the binary floating point datatypes are regularly used for precise numeric values, although the datatype cannot accurately represent these values. For example, out of nine unit ontologies selected in a comparison study [3], five ontologies (OM 1, OM 2, QU, QUDT, SWEET) used XSD binary floating point values and only two ontologies or knowledge graphs (OBOE, Wikidata) used `xsd:decimal`

[1] *integer*, *long*, *int*, *short*, *byte*, *nonNegativeInteger*, *positiveInteger*, *unsignedLong*, *unsignedInt*, *unsignedShort*, *unsignedByte*, *nonPositiveInteger*, and *negativeInteger*.

values for unit conversion factors. Even a popular ontology guideline [4] and a World Wide Web Consortium (W3C) working group note [5] use binary floating point datatypes for precise numeric values in examples.

In general, binary floating point numbers are meant to approximate decimal values in a fixed length binary representation to limit memory consumption and increase computation speed. In RDF, however, binary floating point numbers are defined to represent the exact value of the binary representation: Binary floating point values do not approximate typed decimals, as in programming languages, but typed decimals are abbreviations for exact binary floating point values. This causes ambiguity about the intended meaning of numeric values. We show that 29 %–68 % of the floating point values in real web data are distorted due to the datatype. With regard to the growing use of RDF for the representation of data, including research data, this ambiguity is concerning.

Further, the use of binary floating point datatypes for precise numeric values regularly causes rounding errors in the values actually represented, compared to typed values provided as decimals. Subsequently, error accumulation may significantly falsify the result of processing these values. Disasters, such as the Patriot Missile Failure [6], which resulted in 28 deaths, illustrate the potential impact of accumulated errors in real world applications. The increasing relevance of knowledge graphs for real-world applications calls for general awareness of these issues in the Semantic Web community.

In this paper, we discuss advantages and disadvantages of different numeric datatypes. We demonstrate the practical relevance of the outlined problem with a survey of the actual usage of datatypes on the relevant subset of the December 2020 Web Data Commons dataset, containing 19 453 060 341 literals from real web data. We aim to raise awareness of the implications of datatype selection in RDF and to enable a more informed choice in the future. This work is structured as follows: In Section 2, we give an overview of relevant standards and related work, followed by a comparison of the properties of the binary floating point and decimal datatypes in Section 3. In Section 4, we discuss the implications of the datatype properties in different use cases. An approach for automatic problem detection is outlined in Section 5. In Section 6, we present a survey on the use of datatypes in the World Wide Web that demonstrates the practical relevance of the outlined problem. Finally, we indicate approaches for the general mitigation of the problem in Section 7.

2 Background

Each datatype in RDF consists of a lexical space, a value space, and a lexical-to-value mapping. This is compatible with datatypes in XSD [1].

Value space: the set of values for a datatype [1,2].

Lexical space: the prescribed set of strings, which the lexical mapping for a datatype maps to values of that datatype. The members of the lexical space are **lexical representations** (**lexical forms**) of the values to which they are mapped [1,2].

Lexical mapping (lexical-to-value mapping): a prescribed relation which maps from the lexical space of a datatype into its value space [1,2].

RDF reuses the XSD datatypes with only a few exceptions and additions of non-numeric datatypes [1]. For non-integer numbers, XSD provides the datatypes decimal, float and double. The XSD datatype **decimal** (`xsd:decimal`) represents a subset of the real numbers [2].

Value space of `xsd:decimal`: the set of numbers that can be obtained by dividing an integer by a non-negative power of ten: $\frac{i}{10^n}$ with $i \in \mathbb{Z}, n \in \mathbb{N}_0$, precision is not reflected [2].
Lexical space of `xsd:decimal`: the set of all decimal numbers with or without a decimal point [2].
Lexical mapping of `xsd:decimal`: set i according to the decimal digits of the lexical representation and the leading sign, and set n according to the position of the period or 0, if the period is omitted. If the sign is omitted,"+" is assumed [2].

The XSD datatype **float** (`xsd:float`) is aligned with the IEEE 32-bit binary floating point datatype [7][2], the XSD datatype **double** (`xsd:double`) is aligned to the IEEE 64-bit binary floating point datatype [7]. Both represent subsets of the rational numbers. They only differ in their three defining constants [2].

Value space of `xsd:float` (`xsd:double`): the set of the special values *positiveZero*, *negativeZero*, *positiveInfinity*, *negativeInfinity*, and *notANumber* and the numbers that can be obtained by multiplying an integer m whose absolute value is less than 2^{24} (double: 2^{53}) with a power of two whose exponent e is an integer between -149 (double: -1074) and 104 (double: 971): $m \cdot 2^e$ [2].
Lexical space of `xsd:float` (`xsd:double`): the set of all decimal numbers with or without a decimal point, numbers in exponential notation, and the literals `INF`, `+INF`, `-INF`, and `NaN` [2].
Lexical mapping of `xsd:float` (`xsd:double`): set either the according numeric value (including rounding, if necessary), or the according special value. An implementation might choose between different rounding variants that satisfy the requirements of the IEEE specification.

Numbers with a fractional part of infinite length, like the rational number $\frac{1}{3} = 0.\bar{3}$ or the irrational number $\sqrt{2} = 1.4142\ldots$, are not in the value space of `xsd:float` or `xsd:double`, as a number of finite length multiplied or divided by two is always a number of finite length again. Consequently, a finite decimal with sufficient precision can exactly represent every possible numeric value or lexical representation of an `xsd:float` or `xsd:double`, except of the special values *positiveInfinity*, *negativeInfinity*, and *notANumber*. In contrast, a finite binary floating point value can not exactly represent every possible decimal value.

[2] As the XSD recommendation refers to IEEE 754-2008 version of the standard, we do not refer to the subsequent IEEE 754-2019 version.

Some serialization or query languages for RDF provide a shorthand syntax for numeric literals without explicit datatype specification. In Turtle, TriG and SPARQL a number without fraction is an `xsd:integer`, a number with fraction is an `xsd:decimal`, and a number in exponential notation is an `xsd:double` [8,9,10]. In JSON-LD a number without fractions is an `xsd:integer` and a number with fraction is an `xsd:double`, to align with the common interpretation of numbers in JSON [11]. However, this is not necessary to comply with the JSON specifications [12,13]. The serialization languages RDF/XML, N-Triples, N-Quads, and RDFa do not provide a shorthand syntax for numeric literals [14,15,16,17]. Other languages for machine-readable annotation of HTML, which are regularly mapped to RDF, i.e. Microformats[3], and Microdata[4], do not incorporate explicit datatypes.

In addition to the core XSD datatypes, a W3C working group note introduces the `precisionDecimal` datatype [18]. It is aligned to the IEEE decimal floating-point datatypes [7] and represents a subset of real numbers. It retains precision and permits the special values *positiveZero*, *negativeZero*, *positiveInfinity*, *negativeInfinity*, and *notANumber*. Further, it supports exponential notation. The precision and exponent values of the `precisionDecimal` datatype are unbounded, but can be restricted in derived datatypes to comply with an actual IEEE decimal floating-point datatype. However, even though the RDF standard permits the use of `precisionDecimal`, it does not demand its support in compliant implementations [1]. Therefore, RDF frameworks can not be expected to support `precisionDecimal`.

Another W3C working group note addresses the selection of proper numeric datatypes [5]. It identified three relevant use cases of numeric values: count, measurement, and constant. According to the note, the appropriate datatypes are (derived datatypes of) `xsd:integer` for counts, `xsd:float` or `xsd:double` for measurements, and `xsd:decimal` for constants.

The common vocabulary schema.org[5] defines the alternative numeric datatypes `schema:Integer` and `schema:Float` and their super datatype `schema:Number`. A usage note restricts the lexical space of `schema:Number` to the digits 0 to 9 and at most one full stop. No further restrictions of the lexical or value space are made. `schema:Number` is directly in the range of 91 properties and `schema:Integer` is directly in the range of 47 properties. `schema:Float` is not directly in the range of any property.

The digital representation or computation of numerical values can cause numerical problems: An *overflow error* occurs, if a represented value exceeds the maximum positive or negative value in the value space of a datatype [19]. An *underflow error* occurs, if a represented value is smaller than the minimum positive or negative value different from zero in the value space of a datatype [19]. A *rounding error* occurs, if a represented value is not in the value space of a datatype. It is then represented by a nearby value in the value space that is de-

[3] `https://microformats.org`

[4] `https://html.spec.whatwg.org/multipage/microdata.html`

[5] `http://schema.org`, current version 13.0

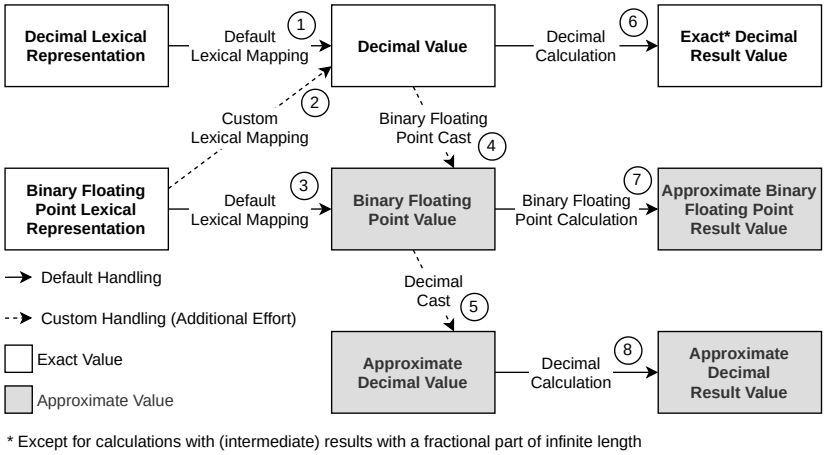

Fig. 1. Possible processing paths of numeric literals depending on their datatype.

termined by a rounding scheme [19]. A *cancellation* is caused by the subtraction of nearly equal values and eliminates leading accurate digits. This will expose errors in the input values of the subtraction [19]. *Error accumulation* is the insidious growth of errors due to the use of a numerically instable sequence of operations [19].

3 Properties of Binary Floating Point and Decimal Datatypes in RDF

Binary floating point and decimal datatypes in the context of RDF have individual properties, which make them more or less suitable for specific use cases:

`xsd:float` and `xsd:double` permit the use of **positive and negative infinite values**. `xsd:decimal` supports neither positive nor negative infinite values.

`xsd:float` and `xsd:double` permit the **exponential notation**. Especially in the case of numbers with many leading or trailing zeros, this is more convenient and less error-prone to read or write for humans. `xsd:decimal` does not permit the exponential notation. There is no actual reason for this limitation. For example, Wikibase[6] also accepts exponential notation for `xsd:decimal`.[7] The *XML Schema Working Group* decided against allowing exponential notation for `xsd:decimal`, as the requirement to have a decimal datatype permitting exponential notation was already met by `precisionDecimal` [20], which, however, has been dropped in the later process of the XSD standardization [21]. To our knowledge, this has not been considered during the RDF standardization.

Figure 1 presents the possible processing paths of numeric literals depending on their datatype. It shows that only decimal lexical representations can be

[6] `https://wikiba.se/`, SPARQL endpoint example: `https://query.wikidata.org`

[7] Example: `SELECT ("1e-9"^^<http://www.w3.org/2001/XMLSchema#decimal> AS ?d) WHERE {}`

used to produce exact result without custom operations. ① to ⑧ denote the mapping operations, cast operations, and calculation operations on values of different datatypes and will be used as references in the following explanations.

The value spaces of `xsd:float` and `xsd:double` only provide partial **coverage of the lexical space**. Therefore, the lexical mapping (③) might require rounding to a possible binary representation and the actual value might slightly differ from the lexical representation. For example, `xsd:float` has no exact binary representation of 0.1 and actually maps it to a slightly higher binary representation of 0.100 000 001 4..., if using the default *roundTiesToEven* rounding scheme [7]. Depending on the used RDF framework, it might be possible to preserve the exact value of the lexical representation by implementing a custom mapping to decimal (②). However, this causes additional development effort and introduces non standard compliant behavior. The value space of `xsd:decimal` covers all values in the lexical space. Therefore, the lexical mapping (①) always provides the exact numeric value described in the lexical representation without any rounding. All three datatypes, `xsd:float`, `xsd:double`, and `xsd:decimal`, do not cover the precision reflected by the lexical representation. For example, literals with the lexical representations 0.5 and 0.50 are considered equal although their lexical representations reflect different precision. The only discussed datatype that preserves the reflected precision is `precisionDecimal`.

The **accuracy of calculations** based on `xsd:float` or `xsd:double` literals (⑦) is limited, as a properly implemented RDF framework will use binary floating point arithmetic by default. For example, this happens during the execution of SPARQL queries that include arithmetic functions or aggregations. Therefore, the calculations might be affected by various numeric problems, i.e. underflow errors, overflow errors, rounding errors, cancellation, and error accumulation. Calculations based on `xsd:decimal` literals (⑥) will by default use a decimal arithmetic with arbitrary precision. Thus, they might only be affected by rounding errors in case of (intermediate) results with a fractional part of infinite length, as well as accumulations of these rounding errors. This different behavior is demonstrated in Figure 2. Depending on the used RDF framework, it might be possible to cast between the datatypes (④ and ⑤). However, a value cast from binary floating point to decimal (⑤) is still affected by the rounding error of the floating point value caused by the lexical mapping. Subsequent calculations (⑧) will still result in approximate results only. In contrast, the results of calculations based on a value cast from decimal to floating point (④) and based on an initial floating point value (③) do not differ, if the same rounding method is used. The SPARQL query in Figure 2 and the according result provided by Wikibase[6] demonstrate differing numerical problems of the datatypes. Other SPARQL endpoints, i.e. Virtuoso 8.3[8] and Apache Fuseki 5.16.0[9], provide similar results.

[8] `https://virtuoso.openlinksw.com/`
[9] `https://jena.apache.org/`

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?datatype
  (xsd:decimal(STRDT("0.1", ?datatype)) AS ?rounded)
  (xsd:decimal(STRDT("1", ?datatype) / STRDT("3", ?datatype)) AS
    ?roundedInfinit)
  (xsd:decimal(STRDT("1.0000001", ?datatype) - STRDT("1.0000000",
    ?datatype)) AS ?cancellation)
  (STRDT("1000000000000000000000000000000000000000", ?datatype) * STRDT("1",
    ?datatype) AS ?overflow)
  (STRDT("0.00000000000000000000000000000000000000000000001", ?datatype) *
    STRDT("1", ?datatype) AS ?underflow)
WHERE {VALUES ?datatype {xsd:float xsd:decimal}}
```

datatype	xsd:float	xsd:decimal
rounded	0.10000000149011612	0.1
roundedInfinit	0.3333333432674408	0.33333333333333333
cancellation	0.000000119209289550781250	0.0000001
overflow	Infinity	1000000000000000000000000000000000000000
underflow	0.0	0.00000000000000000000000000000000000000000000001

Fig. 2. Top: A SPARQL query that demonstrates differing numerical problems of the datatypes `xsd:float` and `xsd:decimal`. Bottom: The corresponding query output (transformed), as on `http://query.wikidata.org`.

4 Implications for the Selection of Numeric Datatypes

The traditional use case of RDF is knowledge representation. The XSD floating point datatypes provide two advantages for knowledge representation compared to `xsd:decimal`: Firstly, the permitted representation of positive and negative infinite might be needed in some cases. Secondly, the exponential notation eases the representation of very large and very small values and reduces the risk of typing errors due to missing or additional zeros. This would not be an issue in case of proper user interface support. But popular tools, like WebProtégé[10] and Protégé Desktop[10], do not help the user here. Further, projects that manipulate their RDF documents under version control using SPARQL UPDATE queries, custom generation scripts and manual edits do not have such a user interface at all.

However, in most cases, knowledge concerned with numbers deals with exact decimal numbers or intervals of decimal values. Intervals are typically described with two exact decimal numbers, either with a minimum value and a maximum value (e.g. $[0.05, 0.15]$) or a value and a measurement uncertainty (e.g. 0.1 ± 0.05) [22]. The binary floating point datatypes do not allow the accurate representation of exactly known or defined numbers in many cases. In addition, they entail the risk to fool data curators into believing that they stated the exact number, as the lexical representation on first sight appears to be exact. This becomes even more critical, if `xsd:double` was used unintentionally due to a shorthand syntax in Turtle, TriG, SPARQL, or JSON-LD. This way, the use of binary floating point datatypes produces ambiguity in the data: The intended meaning could be either the actually represented number in the value space or the verbatim interpretation

[10] `https://protege.stanford.edu/`

of the lexical representation. This ambiguity counteracts the basic ideas behind the Semantic Web and Linked Open Data to ease understanding and reuse of data. Therefore, binary floating point datatypes are not suitable to fulfill the requirements for knowledge representation.

In consequence, the knowledge cannot be used for exact calculations without programming overhead. The possible small rounding errors of binary floating point input values might accumulate to significant errors in calculation results. Disasters, as the Patriot Missile Failure [6], illustrate the potential impact of accumulated errors in real world applications.

This contradicts a W3C working group note [5], stating that binary floating point datatypes are appropriate for measurements. It provided the following example representation of a measurement in the interval of 73.0 to 73.2:

```
_:w eg:value     "73.1"^^xsd:float .
_:w eg:errorRange "0.1"^^xsd:float .
```

However, if using the default *roundTiesToEven* rounding scheme [7], this example actually represents a measurement in the interval 72.999 998 472 6... to 73.199 998 475 6..., as 73.1 and 0.1 are not in the value space of `xsd:float`.[11] In consequence, the actual represented error interval does not cover the points between 73.199 998 475 6... and 73.2. A common solution for this problem is the use of different rounding schemes for the calculation of the upper and lower bound of the interval (outward rounding) [23]. Unfortunately, this is not provided in current RDF frameworks and causes additional programming effort. The example shows that also in case of measurements binary floating point datatypes have clear disadvantages compared to `xsd:decimal`.

Further, the use of binary floating point values in RDF restricts the selection of the used arithmetic for calculations, as it causes an implementation overhead for the application of decimal arithmetic with arbitrary precision. It must be mentioned that calculations using decimal arithmetic with arbitrary precision probably are significantly slower, compared to calculations using binary floating point arithmetic with limited precision. Hence, floating point calculations are better suited for many use cases. However, in certain cases they are not. Therefore, the selection of an arithmetic must be up to the application, not to the input data, as applications might widely vary regarding the required accuracy and the numerical conditioning of the underlying problem.

The same problem arises in use cases that involve the comparison of values, like instance-based ontology matching or ontology based data validation, because comparison values become blurred due to rounding. For example, if using the default *roundTiesToEven* rounding scheme, an upper bound of `"0.1"^^xsd:float` in a constraint still permits a value of `0.100000001`. Thus, the use of binary floating point datatypes for knowledge representation can systematically impair the quality of data and increases the probability of false results of data processing.

[11] Lexical mappings (*roundTiesToEven* rounding scheme): $73.1 \rightarrow 73.099\,998\,474\,1\ldots$ and $0.1 \rightarrow 0.100\,000\,001\,4\ldots$, Interval calculations: $73.099\,998\,474\,1\ldots \pm 0.100\,000\,001\,4\ldots$

In other use cases, RDF might be used for the exchange of initially binary floating point values, as computational results or the output of analog-to-digital converters. If the data to exchange are binary floating point values, the original value can only contain values with an exact binary representation and corruption of data with rounding is impossible. Thus, the use of floating point datatypes for the exchange of computational results is reasonable.

5 Automatic Distortion Detection

The automatic detection of quality issues is key to an effective quality assurance. Therefore, RDF editors, like Protégé[10], or evaluation tools, like the OntOlogy Pitfall Scanner! [24], would ideally warn data curators, if the use of binary floating point datatypes would distort numeric values.

A simple test can be implemented by comparing the results of the default mapping to a binary floating point value (③ in Figure 1) followed by a cast to decimal (⑤ in Figure 1) and a custom mapping to a decimal value (② in Figure 1). The SPARQL query in Figure 3 demonstrates the approach.

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT
  (xsd:decimal(?xsdFloatValue) AS ?xsdFloatSemantic)
  (?xsdDecimalValue AS ?xsdDecimalSemantic)
  (xsd:decimal(?xsdFloatValue) != ?xsdDecimalValue AS ?distorted)
WHERE {
  VALUES ?lexical {"1" "0.1" "0.5"}
  BIND(STRDT(?lexical, xsd:float) AS ?xsdFloatValue)
  BIND(STRDT(?lexical, xsd:decimal) AS ?xsdDecimalValue)
}
```

xsdFloatSemantic	xsdDecimalSemantic	distorted
0.5	0.5	false
0.10000000149011612	0.1	true
1	1	false

Fig. 3. Top: A SPARQL query that demonstrates an approach to detect number distortion. Bottom: The corresponding query output, as on `http://query.wikidata.org`.

6 Datatype Usage Survey

To determine the practical relevancy of the described problem, we conducted a survey of the actual usage of datatypes. The survey is based on the December 2020 edition[12] of the Web Data Commons dataset [25]. The Web Data Commons dataset provides in several N-Quads files the embedded RDF data of 1.7e9 HTML documents extracted from all 3.4e9 HTML documents contained

[12] `http://webdatacommons.org/structureddata/#results-2020-1`

in the September 2020 Common Crawl archive,[13] a freely available web crawl archive. We selected it because of its large size, an expected large proportion of literals, the uniform access of the whole corpus, its heterogeneous original sources (15e6 domains), and a good reflection of RDF usage by a wide range of people. The December 2020 Web Data Commons dataset is divided into data extracted from embedded JSON-LD, RDFa, Microdata, and several Microformats. We only considered data from embedded JSON-LD (7.7e8 URLs, 3.2e10 triples) and RDFa (4.1e8 URLs, 5.9e9 triples), as Microdata and Microformats do not incorporate explicit datatypes.

We created a Java program based on Apache Jena[9] to stream and analyze the relevant parts of the Web Data Commons dataset. The dataset replicates malformed IRIs or literals as they appeared in the original source. To avoid parsing failures of whole files due to single malformed statements, each line was parsed independently and failures were logged separately. Overall, about 4.5e7 failures occurred. The main reasons for failures were malformed IRIs and illegal character encodings. Transaction mechanisms were used to ensure the consistency of the resulting dataset in case of temporary failures of involved systems. Per source type, dataset file, property, and datatype we measured:

1. **UnpreciseRepresentableInDouble**: the number of lexicals that are in the lexical space but not in the value space of `xsd:double`.
2. **UnpreciseRepresentableInFloat**: the number of lexicals that are in the lexical space but not in the value space of `xsd:float`.
3. **UsedAsDatatype**: the total number of literals with the datatype.
4. **UsedAsPropertyRange**: the number of statements that specify the datatype as range of the property.
5. **ValidDecimalNotation**: the number of lexicals that represent a number with decimal notation and whose lexical representation is thereby in the lexical space of `xsd:decimal`, `xsd:float`, and `xsd:double`.
6. **ValidExponentialNotation**: the number of lexicals that represent a number with exponential notation and whose lexical representation is thereby in the lexical space of `xsd:float`, and `xsd:double`.
7. **ValidInfOrNaNNotation**: the number of lexicals that equals either `INF`, `+INF`, `-INF` or `NaN` and whose lexical representation is thereby in the lexical space of `xsd:float`, and `xsd:double`.
8. **ValidIntegerNotation**: the number of lexicals that represent an integer number and whose lexical representation is thereby in the lexical space of `xsd:integer`, `xsd:decimal`, `xsd:float`, and `xsd:double`.

Unfortunately, the lexical representation of `xsd:double` literals from embedded JSON-LD was normalized during the creation of the Web Data Commons dataset to always use exponential notation with one integer digit and up to 16 fractional digits.[14] This is a legal transformation according to the definition of

[13] `https://commoncrawl.org/2020/10/september-2020-crawl-archive-now-available/`

[14] `https://github.com/jsonld-java/jsonld-java/blob/v0.13.1/core/src/main/java/com/github/jsonldjava/core/RDFDataset.java#L673`

Table 1. The number of datatype occurrences in the Web Data Commons December 2020 dataset from RDFa and embedded JSON-LD sources (Measure 3) in absolute numbers and relative to the total number of literals in the source type (Measure 3). Only the top ten, as well as selected further datatypes are shown.

RDFa		Embedded JSON-LD	
Datatype	Occurrences (rel)	Datatype	Occurrences (rel)
`rdf:langString`	3 179 161 585 (.68)	`xsd:string`	11 277 500 571 (.76)
`xsd:string`	1 305 371 136 (.28)	`xsd:integer`	2 021 243 795 (.14)
`xsd:dateTime`	102 987 223 (.02)	`schema:Date`	1 313 408 439 (.09)
`rdf:XMLLiteral`	62 337 177 (.01)	`xsd:double`	101 959 406 (.01)
`xsd:integer`	21 547 053 (.00)	`xsd:boolean`	26 144 338 (.00)
`xsd:float`	1 025 753 (.00)	`schema:DateTime`	25 002 464 (.00)
`use:sku`	729 858 (.00)	`rdf:langString`	12 934 431 (.00)
`xsd:date`	507 454 (.00)	`xsd:float`	90 895 (.00)
`xsd:boolean`	348 334 (.00)	`xsd:dateTime`	12 260 (.00)
`schema:Date`	246 995 (.00)	`rdf:HTML`	5785 (.00)
`xsd:decimal`	8288 (.00)	`xsd:decimal`	1 (.00)
`xsd:double`	234 (.00)	`schema:Number`	0 (.00)
`schema:Number`	0 (.00)	`schema:Integer`	0 (.00)
`schema:Integer`	0 (.00)	`schema:Float`	0 (.00)
`schema:Float`	0 (.00)		

`xsd:double`, as the represented value is preserved. However, this limits the use of the according *Valid...* and *Unprecise...* measures. At the same time, this demonstrates that the use of `xsd:float` or `xsd:double` might easily cause the loss of information due to legal transformation, if information is only reflected in the lexical representation.

The resulting dataset consists of a CSV file containing the measurement results (5.4e7 lines, 0.6 GiB compressed, 11.0 GiB uncompressed). The analysis was conducted with Python scripts. The tool [26], the resulting dataset [27], and the analysis scripts [28] are freely available for review and further use under permissive licenses.

For the analysis, we first applied some data cleaning: Some properties and datatypes were regularly denoted by IRIs in the `http` scheme as well as in the `https` scheme. To enable proper aggregation, the scheme of all IRIs in the dataset were unified to `http`. Further, the omission of namespace definitions in the source websites causes the occurrence of prefixed names instead of full IRIs. All prefixes in datatypes that occurred at least for one datatype more than 1000 times and all prefixes in properties that occurred at least for one property more than 1000 times have been replaced with the actual namespace, if we found a resource with a matching local name and matching default vocabulary prefix during a web search or in other used properties or datatypes. Rarer prefixes have not been replaced because of the high effort, the susceptibility to errors caused by ambiguity, and the lack of significance for the results. Further, we did not clean other kinds of typos like missing or duplicated `#` or `/` after the namespace, as these errors could also not easily be fixed by applications with, e.g., a maintained list of widely used prefixes.

Table 2. The number of property occurrences with XSD or schema.org numerical datatypes in the Web Data Commons December 2020 dataset from RDFa and embedded JSON-LD sources (Measure 3) in absolute numbers and relative to the total number of numeric literals in the source type (Measure 3). Only the top ten are shown.

RDFa		Embedded JSON-LD	
Property	Occurrences (rel)	Property	Occurrences (rel)
`sioc:num_replies`	21 391 187 (.95)	`schema:position`	893 910 601 (.42)
`gr:hasCurrencyValue`	525 491 (.02)	`schema:width`	448 036 253 (.21)
`gr:hasMinValue`	137 018 (.01)	`schema:height`	446 308 779 (.21)
`gr:amountOfThisGood`	94 978 (.00)	`schema:price`	71 045 655 (.03)
`gr:hasMaxValue`	52 772 (.00)	`schema:commentCount`	65 723 049 (.03)
`vcard:latitude`	49 428 (.00)	`schema:ratingValue`	26 261 677 (.01)
`vcard:longitude`	49 428 (.00)	`schema:longitude`	17 096 852 (.01)
`gr:hasValue`	25 800 (.00)	`schema:latitude`	17 093 196 (.01)
`dv:count`	24 672 (.00)	`schema:bestRating`	16 333 042 (.01)
`dv:price`	23 936 (.00)	`schema:userInteractionCount`	13 347 182 (.01)

Overall, we processed 14 778 325 375 literals from embedded JSON-LD and 4 674 734 966 literals from RDFa. Table 1 shows the number of occurrences of the most frequent datatypes.[15] Table 2 shows the most frequently used properties that occurred with numerical datatypes from XSD or schema.org. Although the use of the schema.org numeric datatypes instead of XSD numeric datatypes is expected by the definition of many schema.org properties, including widely used properties, like `schema:position` or `schema:price`, we found zero occurrences of schema.org numeric datatypes. The most probable reason is the existence of shorthand syntaxes for XSD numeric datatypes. In contrast, the usage of schema.org temporal datatypes `schema:Date` and `schema:DateTime` in JSON-LD exceeds the usage of XSD temporal datatypes by orders of magnitude. This emphasizes the importance of shorthand syntaxes for the choice of datatypes.

As shown in Table 1, the occurrences of `xsd:float` in RDFa and `xsd:double` in embedded JSON-LD surpass the occurrences of `xsd:decimal` by orders of magnitude. Remarkably, we did find only one single occurrence[16] of `xsd:decimal` among 14 778 325 375 literals from valid triples in embedded JSON-LD sources in the whole Web Data Commons December 2020 dataset. Table 3 shows properties that most frequently occurred with `xsd:float` in RDFa and with `xsd:double` in embedded JSON-LD. We manually classified the top ten properties using their definitions, if found, and the local names. Based on these figures, at least 62 % for `xsd:float` in RDFa and 54 % for `xsd:double` in embedded JSON-LD represent (monetary) amounts, position numbers or single rating values (later refereed to as T10NIFP literals), which are not initially binary floating point values. At least 33 % for `xsd:float` in RDFa and 35 % for `xsd:double` in embedded JSON-

[15] Prefixes used for results presentation: `dcterms:` `http://purl.org/dc/terms/`, `dv:` `http://rdf.data-vocabulary.org/#`, `gr:` `http://purl.org/goodrelations/v1#`, `rdf:` `http://www.w3.org/1999/02/22-rdf-syntax-ns#`, `rev:` `http://purl.org/stuff/rev#`, `schema:` `http://schema.org/`, `use:` `http://search.yahoo.com/searchmonkey-datatype/use/`, `vcard:` `http://www.w3.org/2006/vcard/ns#`, `xsd:` `http://www.w3.org/2001/XMLSchema#`

[16] `https://web.archive.org/web/20200919100939/https://open.nrw/dataset/telefonverzeichnis-alphabetisch-oktober-2019-odp`

Table 3. The number of property occurrences with `xsd:float` in RDFa and with `xsd:double` in embedded JSON-LD in the Web Data Commons December 2020 dataset (Measure 3) in absolute numbers and relative to the total number of literals with the same datatype in the same source type (Measure 3). Only the top ten are shown.

`xsd:float` in RDFa		`xsd:double` in Embedded JSON-LD	
Property	Occurrences (rel)	Property	Occurrences (rel)
`gr:hasCurrencyValue`	516 256 (.50)	`schema:price`	49 740 982 (.49)
`gr:hasMinValue`	134 954 (.13)	`schema:longitude`	17 055 600 (.17)
`gr:amountOfThisGood`	94 978 (.09)	`schema:latitude`	17 053 362 (.17)
`gr:hasMaxValue`	52 772 (.05)	`schema:ratingValue`	9 928 412 (.10)
`vcard:latitude`	49 428 (.05)	`schema:lowPrice`	2 240 110 (.02)
`vcard:longitude`	49 428 (.05)	`schema:highPrice`	1 840 080 (.02)
`gr:hasValue`	25 800 (.03)	`schema:value`	1 776 255 (.02)
`dv:price`	23 086 (.02)	`schema:worstRating`	311 374 (.00)
`dv:average`	21 038 (.02)	`schema:position`	240 577 (.00)
`rev:rating`	20 970 (.02)	`schema:minPrice`	197 850 (.00)

LD represent geolocation values, arbitrary quantity values or aggregated values, which might but do not need to origin from initially binary floating point values. `rev:rating` and `schema:ratingValue` cannot be assigned unambiguously to these categories. This shows that binary floating point numbers are regularly used for not initially binary floating point values.

As expected, because embedded RDF is not the proper place for vocabulary definitions, we found only few cases of property range definitions (Measure 4). They are limited to 54 unique property-datatype-pairs with two to 153 occurrences and for properties from only five different namespaces. This does not allow to draw further conclusions.

Table 4 shows the number of occurrences of different notations. Except for `xsd:double` in embedded JSON-LD, which is affected by normalization, exponential notation is only little used in the binary floating point datatypes. Special values occurred only in even more rare cases. From that, we conclude that the notation or needed special values are not the crucial consideration behind using binary floating point datatypes.

The number of lexical representations that are not precisely representable in binary floating point datatypes is presented in Table 5. 33 % of the represented `xsd:float` values in RDFa and 24 % in embedded JSON-LD differ from lexical representations. In embedded JSON-LD the initial lexical representation of 69 % of the `xsd:double` values must either have contained more then 17 significant digits or already been differing from the represented value. Referring to the most common properties used with `xsd:double` in embedded JSON-LD, shown in Table 3, the frequent occurrence of values with more then 17 significant digits is implausible. All together, this shows that 29 %–68 % of the values with binary floating point datatype in real web data are distorted due to the datatype.[17]

[17] $\frac{\sum_{\text{T10NIFP literals}} \text{Measures 1 \& 2}}{\sum_{\texttt{xsd:double},\texttt{xsd:float literals}} \text{Measure 3}} \approx 0.29$, $\frac{\sum_{\texttt{xsd:double},\texttt{xsd:float literals}} \text{Measures 1 \& 2}}{\sum_{\texttt{xsd:double},\texttt{xsd:float literals}} \text{Measure 3}} \approx 0.68$

Table 4. The number of numeric notations occurrences in the lexical representation of literals per numeric datatype in the Web Data Commons December 2020 dataset (Measures 5, 6, 7, 8) in absolute numbers and relative to the total number of literals with the same datatype (Measure 3). The notation of `xsd:double` in embedded JSON-LD was normalized during the dataset generation.

Embedded JSON-LD

Datatype	Notation			
	Integer	Decimal	Exponential	Inf / NaN
`xsd:decimal`	0 (.00)	1 (1)	0 (.00)	0 (.0)
`xsd:double`	0 (.00)	0 (.00)	101 959 382 (1)	24 (.0)
`xsd:float`	35 951 (.40)	24 837 (.27)	4252 (.05)	0 (.0)
`xsd:integer`	2 021 243 613 (1)	0 (.00)	0 (.00)	0 (.0)
`xsd:long`	36 (1)	0 (.00)	0 (.00)	0 (.0)

RDFa

Datatype	Notation			
	Integer	Decimal	Exponential	Inf / NaN
`xsd:decimal`	89 (.01)	7349 (.89)	0 (.00)	0 (.0)
`xsd:double`	26 (.11)	208 (.89)	0 (.00)	0 (.0)
`xsd:float`	353 851 (.34)	643 206 (.63)	0 (.00)	4 (.0)
`xsd:int`	16 751 (.86)	0 (.00)	0 (.00)	0 (.0)
`xsd:integer`	21 507 446 (1)	38 (.00)	0 (.00)	0 (.0)
`xsd:nonNegativeInteger`	585 (1)	0 (.00)	0 (.00)	0 (.0)
`xsd:positiveInteger`	6 (1)	0 (.00)	0 (.00)	0 (.0)

7 Conclusion

Binary floating point numbers are meant to approximate decimal values to reduce memory consumption and increase computation speed. However, in RDF, decimal representations are used to approximate binary floating point numbers. This way, the use of XSD binary floating point datatypes in RDF can systematically impair the quality of data and produces ambiguity in represented knowledge. Our survey reveals that a considerable proportion of real web data is distorted due to the datatype. Further, its use restricts the choice of the arithmetic in standards compliant implementations and can falsify the results of data processing. This can cause serious impacts in real world applications.

As a second outcome, our survey indicates that shorthand syntaxes for literals are a major cause for the choice of inappropriate datatypes. We conclude that the datatypes and shorthand syntaxes in current RDF related standards encourage the distortion of numeric values. We recommend an overhaul of relevant parts of the standards to make RDF well suited for numeric data.

A radical solution that requires no update of existing data would be the deprecation and replacement of `xsd:float` and `xsd:double` with an extended mandatory `xsd:decimal` datatype in RDF. The extended `xsd:decimal` datatype should additionally permit exponential notation and the special values *positiveInfinity*, *negativeInfinity*, and *notANumber* to cover the whole lexical space and value space of `xsd:float` and `xsd:double`. We recommend to declare it as default datatype in the different serialization and query languages for numbers in decimal and exponential notation. It should also be used for interpretation

Table 5. The number of lexical representation occurrences without exact representation in the value space of per numeric datatype in the Web Data Commons December 2020 dataset `xsd:float` and `xsd:double` (Measures 1, 2) in absolute numbers and relative to the total number of literals with the same datatype (Measure 3). The notation of `xsd:double` in embedded JSON-LD was normalized during the dataset generation.

	Embedded JSON-LD		RDFa	
	Unprecise In			
Datatype	xsd:float	xsd:double	xsd:float	xsd:double
`xsd:decimal`	0 (.00)	0 (.00)	3087 (.37)	3087 (.37)
`xsd:double`	69 648 087 (.68)	69 646 819 (.68)	58 (.25)	58 (.25)
`xsd:float`	21 750 (.24)	21 750 (.24)	339 583 (.33)	338 676 (.33)
`xsd:int`	-	-	0 (.00)	0 (.00)
`xsd:integer`	7 564 635 (.00)	996 (.00)	1492 (.00)	38 (.00)
`xsd:long`	2 (.06)	0 (.00)	-	-
`xsd:nonNegativeInteger`	-	-	136 (.23)	0 (.00)
`xsd:positiveInteger`	-	-	0 (.00)	0 (.00)

instead of the deprecated datatypes, if these are used in existing data. One or several additional new datatypes with hexadecimal lexical representations should be used for the actual representation of binary floating point values. However, this radical solution would make a decision for existing data in favor of the verbatim interpretation of the lexical representation. Thus, in (presumable not occurring) cases of an intended representation of e.g. 0.100 000 001 4... with `"0.1"^^xsd:float`, existing data would get distorted.

A more cautious mitigation of the problem should tackle the disadvantages of `xsd:decimal`: It would be desirable to introduce in RDF mandatory support for (a) an exponential notation for the decimal datatype, and (b) a decimal datatype that supports infinite values, like `precisionDecimal`, to eliminate these disadvantages. Further, binary floating point datatypes should only be used for numeric values if (a) a representation of infinity is required, or (b) the original source provides binary floating point values. In general, `xsd:decimal` must become the first choice for the representation of numbers. Semantic Web teaching materials should clearly name the disadvantages of the binary floating point datatypes, shorthand syntaxes should in future prioritize the decimal datatype, and Semantic Web tools should hint to use `xsd:decimal`.

Acknowledgments. Many thanks to Alsayed Algergawy, Sheeba Samuel, Sirko Schindler, Eberhard Zehendner, and the first author's supervisor Birgitta König-Ries, as well as 10 anonymous reviewers for very helpful comments on earlier drafts of this manuscript.

Author Contributions. Study conception and design, analysis and interpretation of results, and draft manuscript preparation were performed by Jan Martin Keil. Data collection was performed by Merle Gänßinger and Jan Martin Keil. All authors read and approved the final manuscript.

References

1. W3C RDF Working Group: *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation. W3C, Feb. 25, 2014. URL: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.
2. W3C XML Schema Working Group: *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*. W3C Recommendation. W3C, Apr. 5, 2012. URL: http://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/.
3. Keil, J.M. and Schindler, S.: "Comparison and evaluation of ontologies for units of measurement". In: *Semantic Web* 10.1 (2019), pp. 33–51. DOI: 10.3233/SW-180310.
4. Noy, N.F. and McGuinness, D.L.: *Ontology Development 101: A Guide to Creating Your First Ontology*. Tech. rep. KSL-01-05/SMI-2001-0880. Stanford Knowledge Systems Laboratory and Stanford Medical Informatics, Mar. 2001. URL: http://www.ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness-abstract.html.
5. W3C Semantic Web Best Practices and Deployment Working Group: *XML Schema Datatypes in RDF and OWL*. W3C Working Group Note. W3C, Mar. 14, 2006. URL: https://www.w3.org/TR/2006/NOTE-swbp-xsch-datatypes-20060314/.
6. *Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia*. Tech. rep. GAO/IMTEC-92-26. General Accounting Office, Information Management and Technology Division, Feb. 4, 1992. 20 pp. URL: https://www.gao.gov/products/IMTEC-92-26.
7. IEEE: *IEEE 754-2008 Standard for Floating-Point Arithmetic*. Standard 754. Aug. 29, 2008. 70 pp. DOI: 10.1109/IEEESTD.2008.4610935.
8. Beckett, D., Berners-Lee, T., Prud'hommeaux, E., and Carothers, G.: *RDF 1.1 Turtle: Terse RDF Triple Language*. W3C Recommendation. W3C, Feb. 25, 2014. URL: https://www.w3.org/TR/2014/REC-turtle-20140225/.
9. Bizer, C. and Cyganiak, R.: *RDF 1.1 TriG: RDF Dataset Language*. W3C Recommendation. W3C, Feb. 25, 2014. URL: https://www.w3.org/TR/2014/REC-trig-20140225/.
10. W3C SPARQL Working Group: *SPARQL 1.1 Query Language*. W3C Recommendation. W3C, Mar. 21, 2013. URL: https://www.w3.org/TR/2013/REC-sparql11-query-20130321/.
11. Sporny, M., Longley, D., Kellogg, G., et al.: *JSON-LD 1.1: A JSON-based Serialization for Linked Data*. W3C Recommendation. W3C, July 16, 2020. URL: https://www.w3.org/TR/2020/REC-json-ld11-20200716/.
12. Bray, T.: *The JavaScript Object Notation (JSON) Data Interchange Format*. Standard 8259. Dec. 2017. 16 pp. DOI: 10.17487/RFC8259.
13. Ecma International: *ECMA-404, The JSON Data Interchange Format*. Standard. 2017. URL: https://ecma-international.org/publications/standards/Ecma-404.htm.
14. W3C RDF Working Group: *RDF 1.1 XML Syntax*. Ed. by Gandon, F. and Schreiber, G. W3C Recommendation. Feb. 25, 2014. URL: https://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/.
15. Beckett, D.: *RDF 1.1 N-Triples: A line-based syntax for an RDF graph*. W3C Recommendation. W3C, Feb. 25, 2014. URL: https://www.w3.org/TR/2014/REC-n-triples-20140225/.
16. W3C RDF Working Group: *RDF 1.1 N-Quads: A line-based syntax for RDF datasets*. W3C Recommendation. W3C, Feb. 25, 2014. URL: https://www.w3.org/TR/2014/REC-n-quads-20140225/.

17. W3C RDFa Working Group: *RDFa Core 1.1 - Third Edition: Syntax and processing rules for embedding RDF through attributes*. W3C Recommendation. W3C, Mar. 17, 2015. URL: https://www.w3.org/TR/2015/REC-rdfa-core-20150317/.
18. W3C XML Schema Working Group: *An XSD datatype for IEEE floating-point decimal*. W3C Working Group Note. W3C, June 9, 2011. URL: https://www.w3.org/TR/2011/NOTE-xsd-precisionDecimal-20110609/.
19. Higham, N.J.: *Accuracy and stability of numerical algorithms, Second Edition*. SIAM, 2002. xxvii + 663. DOI: 10.1137/1.9780898718027.
20. W3C XML Schema Working Group: *RQ-28 Allow scientific notation for decimals (scientific-notn)*. Feb. 11, 2006. URL: https://www.w3.org/Bugs/Public/show_bug.cgi?id=2853.
21. W3C XML Schema Working Group: *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*. W3C Candidate Recommendation. W3C, July 21, 2011. URL: https://www.w3.org/TR/2011/CR-xmlschema11-2-20110721/.
22. *International Vocabulary of Metrology. Basic and general concepts and associated terms*. JCGM 200:2012 (JCGM 200:2008 with minor corrections). Joint Committee for Guides in Metrology, 2012.
23. Neumaier, A.: *Introduction to Numerical Analysis*. Cambridge University Press, Aug. 23, 2012. 366 pp.
24. Poveda-Villalón, M., Gómez-Pérez, A., and Suárez-Figueroa, M.C.: "OOPS! (OntOlogy Pitfall Scanner!): An On-line Tool for Ontology Evaluation". In: *International Journal on Semantic Web and Information Systems* 10.2 (2014), pp. 7–34. DOI: 10.4018/ijswis.2014040102.
25. Meusel, R., Petrovski, P., and Bizer, C.: "The WebDataCommons Microdata, RDFa and Microformat Dataset Series". In: *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*. Ed. by Mika, P., Tudorache, T., Bernstein, A., et al. Vol. 8796. Lecture Notes in Computer Science. Springer, 2014, pp. 277–292. DOI: 10.1007/978-3-319-11964-9_18.
26. Gänßinger, M. and Keil, J.M.: *RDF Property and Datatype Usage Scanner v1.0.0*. 2021. DOI: 10.5281/zenodo.6258887.
27. Keil, J.M. and Gänßinger, M.: *Web Data Commons (December 2020) Property and Datatype Usage Dataset*. 2022. DOI: 10.5281/zenodo.6205111.
28. Keil, J.M.: *Web Data Commons (December 2020) Property and Datatype Usage Analysis Scripts*. 2022. DOI: 10.5281/zenodo.6264286.