Foundations of RDF* and SPARQL*
(An Alternative Approach to Statement-Level Metadata in RDF)

Olaf Hartig

Dept. of Computer and Information Science (IDA), Linköping University, Sweden
olaf.hartig@liu.se

Abstract The standard approach to annotate statements in RDF with metadata has a number of shortcomings including data size blow-up and unnecessarily complicated queries. We propose an alternative approach that is based on nesting of RDF triples and of query patterns. The approach allows for a more compact representation of data and queries, and it is backwards compatible with the standard. In this paper we present the formal foundations of our proposal and of different approaches to implement it. More specifically, we formally capture the necessary extensions of the RDF data model and its query language SPARQL, and we define mappings based on which our extended notions can be converted back to ordinary RDF and SPARQL. Additionally, for such type of mappings we define two desirable properties, information preservation and query result equivalence, and we show that the introduced mappings possess these properties.

1 Introduction

The term *statement-level metadata* refers to a form of data that captures information about another piece of data representing a single statement or fact. A typical example are so called *edge properties* in graph databases; such an edge property takes the form of a key-value pair that captures additional information about the relationship represented by the edge with which the key-value pair is associated [9]. Popular use cases for applying edge properties (and statement-level metadata in general) include capturing certainty scores, weights, temporal restrictions, and provenance information.

While the Resource Description Framework (RDF) [1] presents another graph-based approach to represent statements about entities and their relationships, its triple-based data model does not natively support the representation of statement-level metadata. To mitigate this limitation the RDF specification introduces the notion of *RDF reification* which can be used to provide a set of RDF triples that describe some other RDF triple [4]. This approach requires the user to include four additional RDF triples to refer to the triple for which the user wants to provide metadata. Then, to query so-represented statement-level metadata using the standard RDF query language SPARQL [2], each query has to contain additional SPARQL triple patterns to match the triples that establish the reification. The following example illustrates the use of RDF reification.

Example 1. Consider the RDF triples in Figure 1a (represented in the standard Turtle syntax [8]).[1] The first two of these triples (i.e., the first two lines of Figure 1a) indicate the age of somebody named Bob. To capture metadata about a given RDF triple as per RDF reification, we have to introduce an IRI or a blank node and use it as the subject of four RDF triples that reify the given triple by employing the RDF reification vocabulary [4]. For instance, the last four triples in Figure 1a use a blank node labeled `_:s` to reify the second of the example triples about Bob. Now, we can provide metadata

[1] We omit prefix declarations in this paper. The prefixes used can be found at http://prefix.cc.

```
:bob foaf:name "Bob" ;
     foaf:age 23 .
_:s rdf:type rdf:Statement ;
    rdf:subject   :bob ;
    rdf:predicate foaf:age ;
    rdf:object    23 .
```

(a)

```
SELECT ?x ?age ?src WHERE {
   ?x foaf:age ?age .
   ?r rdf:type rdf:Statement ;
      rdf:subject ?x ;
      rdf:predicate foaf:age ;
      rdf:object ?age ;
      dct:source ?src . }
```

(b)

```
_:s dct:creator <http://example.com/crawlers#c1> ;
    dct:source <http://example.net/listing.html> .
```

(c)

```
SELECT ?x ?age ?src WHERE {
   ?x foaf:age ?age .
   ?x   ?p     ?age .
   ?p rdf:singletonPropertyOf foaf:age .
   ?x dct:source ?src . }
```

(d)

```
SELECT ?x ?age ?src WHERE {
   GRAPH ?g {
      ?x foaf:age ?age .
   }
   ?g dct:source ?src . }
```

(e)

Figure 1: Existing approaches to RDF statement-level metadata: (a, b, c) data and query for RDF reification, (d) query for singleton properties, and (e) query for named graphs.

about the reified triple by using this blank node as illustrated in Figure 1c. Given such data (including the metadata), assume we want to retrieve a list containing the age of persons in our data and the respective sources of the statements about the persons' ages. To this end, we may use the SPARQL query in Figure 1b. Note that the given query contains four triple patterns to identify the reified triple whose metadata we want to see.

The example highlights two major shortcomings of RDF reification: First, adding four reification triples for every reified triple is inefficient for exchanging RDF data. Second, writing queries to access statement-level metadata is cumbersome because any metadata-related (sub)expression in a query has to be accompanied by another subexpression to match the corresponding four reification triples. To address these shortcomings other authors have proposed alternative approaches including *singleton properties* [6], and an application of *named graphs* [5]. However, as can be observed in Figures 1d and 1e, each of the two proposals still requires verbose constructs in queries whose only purpose is to match artifacts that the respective proposal introduces to associate a triple with the metadata about it (Figures 1d and 1e present the example query from Figure 1b in terms of the two proposals, respectively). An additional issue of the singleton properties proposal is that it introduces a large number of unique predicates, which is untypical for RDF data and, thus, disadvantageous for commonly-used SPARQL optimization heuristics [11]. An additional disadvantage of the proposal to use named graphs is that it inhibits an application of named graphs for other use cases.

We propose an alternative that allows for very concise queries by still remaining backwards compatible. That is, our proposal can be implemented based on any system that has been optimized for any of the other proposals. Additionally, our proposal also opens the possibility of mapping it natively to a corresponding physical storage model, which may be a foundation for novel implementations tailored to our proposal. The basis of our proposal is to extend the RDF data model with a notion of nested triples. More precisely, the extension, which we call *RDF**, allows for triples that represent metadata about another triple by directly using this other triple as their subject or object.

Example 2. Assume an extension of the Turtle syntax that implements the idea of nested triples by enclosing any embedded triple using the strings '<<' and '>>' (we call this extended syntax *Turtle** and specify it in our technical report [3]). Then, all data in Example 1 (including the metadata in Figure 1c) may be represented as follows.

```
:bob foaf:name "Bob" .
<<:bob foaf:age 23>> dct:creator <http://example.com/crawlers#c1> ;
                     dct:source <http://example.net/listing.html> .
```

Given the outlined notion of RDF* with nested triples, the crux of our proposal is to extend the SPARQL query language accordingly. That is, in the extended language, called *SPARQL**, triple patterns may also be nested, which gives users a query syntax in which accessing specific metadata about a triple is just a matter of mentioning the triple in the subject (or object) position of a metadata-related triple pattern. Apparently, the subject or object of such a metadata-related triple pattern may not only be a concrete (i.e., variable-free) triple, but it may also be another triple pattern with variables.

Example 3. By adopting the extended syntax outlined in Example 2, we may represent the query of Example 1 (cf. Figure 1b) in the following, more compact form.

```
SELECT ?x ?age ?src WHERE { <<?x foaf:age ?age>> dct:source ?src . }
```

In an ongoing research project we aim to study the trade-offs of RDF* and SPARQL*. The purpose of this paper is to present preliminary work towards such a study. In particular, we provide a formal foundation of different approaches to implement our proposal. We emphasize that RDF* and SPARQL* may be understood—and used—simply as syntactic sugar on top of RDF and SPARQL. That is, any RDF*-specific syntax (such as the aforementioned Turtle*) may be parsed directly into plain RDF data that uses any of the other RDF-based approaches for representing statement-level metadata. Then, any given SPARQL* query may be rewritten accordingly into an ordinary SPARQL query. In this paper we define formal mappings for these conversions of data and queries, and we show that the data-level mappings are *information preserving* (i.e., the resulting RDF data can be converted back to the original RDF* representation) and the query-level mappings preserve a notion of *query result equivalence*. These contributions present a foundation for implementing our proposal as a wrapper on top of any existing RDF triple store. Such an implementation does not only require a comparably small effort, but it can also benefit readily from possible optimizations that the triple store has for RDF reification (or any of the other related proposals). On the other hand, our proposal may also be implemented natively, which requires the development of techniques to execute SPARQL* queries directly on a physical storage model designed to support RDF*. For instance, the idea of nesting carries over naturally to the physical level where it may be adopted to develop a storage model that embeds physical representations of triples into one another. As a formal foundation of native implementations, in this paper we provide a query semantics of SPARQL*. In summary, we make three main contributions:

1. We introduce the RDF* data model formally, define a formal semantics of SPARQL*, and show properties related to redundancies in RDF* data (Section 2).
2. We define desirable properties of mappings that may be used to store RDF* data as ordinary RDF data and convert SPARQL* queries into SPARQL queries (Section 3).
3. We define concrete examples of such mappings that use the RDF reification vocabulary, and we show that these mappings possess the desirable properties (Section 4).

2 RDF* and SPARQL*

This section formalizes our proposal. We begin with the structural part of the RDF* data model. Thereafter, we define SPARQL* by introducing a syntax and a formal semantics. Finally, we show properties related to redundancies as are possible by our data model.

2.1 RDF*

As a basis for the following definitions, we assume pairwise disjoint sets $\mathcal{I}$ (all IRIs), $\mathcal{B}$ (blank nodes), and $\mathcal{L}$ (literals). As usual, a tuple in $(\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$ is an *RDF triple* and a set of RDF triples is an *RDF graph* [1]. RDF* extends the notion of such triples by allowing for triples that have another triple in its subject or its object position. Such nesting may be arbitrarily deep. The following definition captures this notion.

Definition 1. An **RDF* triple** is a 3-tuple that is defined recursively as follows:
1. Any RDF triple $t \in (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$ is an RDF* triple; and
2. Given RDF* triples t and t', and RDF terms $s \in (\mathcal{I} \cup \mathcal{B})$, $p \in \mathcal{I}$ and $o \in (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$, then the tuples (t, p, o), (s, p, t) and (t, p, t') are RDF* triples.

We write $\mathcal{T}^\star$ to denote the (infinite) set of all RDF* triples. Moreover, for any RDF* triple $t = (s, p, o)$ we write $\mathrm{Elmts}^+(t)$ to denote the set of all RDF terms and all RDF* triples mentioned in t; i.e., $\mathrm{Elmts}^+(t) = \{s, p, o\} \cup \{x \in \mathrm{Elmts}^+(t') \mid t' \in \{s, o\} \cap \mathcal{T}^\star\}$. Note that, if $\mathrm{Elmts}^+(t) \cap \mathcal{T}^\star \neq \emptyset$, then t represents a form of statement-level metadata and, thus, we call t a *metadata triple* (any other RDF* triple is an ordinary RDF triple).

Similar to the notion of an RDF graph, we refer to a set of RDF* triples as an *RDF* graph*. For these graphs we overload function Elmts^+, that is, for each RDF* graph $G^\star$ we define $\mathrm{Elmts}^+(G^\star) = \bigcup_{t \in G^\star} \mathrm{Elmts}^+(t)$. Furthermore, we write $\mathrm{T}^+(G^\star)$ to denote the set of all RDF* triples in an RDF* graph $G^\star$, including those that are (recursively) embedded in other RDF* triples of $G^\star$; i.e., $\mathrm{T}^+(G^\star) = G^\star \cup \big(\mathrm{Elmts}^+(G^\star) \cap \mathcal{T}^\star\big)$. Note that, since any RDF triple is an RDF* triple (case 1 in Definition 1), any RDF graph G also is an RDF* graph (for which it holds that $\mathrm{Elmts}^+(G) \cap \mathcal{T}^\star = \emptyset$ and $\mathrm{T}^+(G) = G$).

Example 4. The data given in Turtle* syntax in Example 2 can be captured formally by an RDF* graph $G^\star_{\mathsf{ex}} = \{(\mathsf{bob}, \mathsf{name}, \textit{Bob}), (t, \mathsf{creator}, \mathsf{c1}), (t, \mathsf{source}, \mathsf{listing.html})\}$, where t is the RDF* triple $(\mathsf{bob}, \mathsf{age}, \textit{23})$ (which is an ordinary RDF triple); $\textit{Bob}, \textit{23} \in \mathcal{L}$, and $\mathsf{bob}, \mathsf{name}, \mathsf{creator}, \mathsf{c1}, \mathsf{source}, \mathsf{listing.html}, \mathsf{age} \in \mathcal{I}$. Note that $\mathrm{T}^+(G^\star_{\mathsf{ex}}) = G^\star_{\mathsf{ex}} \cup \{t\}$.

2.2 SPARQL* Syntax

For the sake of conciseness, in this paper we introduce a syntax for SPARQL* that is based on Pérez et al.'s algebraic SPARQL syntax [7], and we focus only on the core concepts. For a more detailed formalization of SPARQL*, including the complete extension of the full W3C specification of SPARQL, we refer to our technical report [3].

We recall that the basic building block of SPARQL queries is a *basic graph pattern* (*BGP*); that is, a finite set of triple patterns, where every *triple pattern* is a tuple of the form $(s, p, o) \in \big(\mathcal{V} \cup \mathcal{I} \cup \mathcal{L}\big) \times \big(\mathcal{V} \cup \mathcal{I}\big) \times \big(\mathcal{V} \cup \mathcal{I} \cup \mathcal{L}\big)$ with $\mathcal{V}$ being a set of query variables that is disjoint from $\mathcal{I}$, $\mathcal{B}$, and $\mathcal{L}$, respectively. SPARQL* extends these patterns by adding the possibility to nest triple patterns within one another.

Definition 2. A **triple* pattern** is a 3-tuple that is defined recursively as follows:
1. Any triple pattern $t \in \big(\mathcal{V} \cup \mathcal{I} \cup \mathcal{L}\big) \times \big(\mathcal{V} \cup \mathcal{I}\big) \times \big(\mathcal{V} \cup \mathcal{I} \cup \mathcal{L}\big)$ is a triple* pattern; and
2. Given two triple* patterns tp and tp', and $s \in (\mathcal{V} \cup \mathcal{I} \cup \mathcal{L})$, $p \in (\mathcal{V} \cup \mathcal{I})$ and $o \in (\mathcal{V} \cup \mathcal{I} \cup \mathcal{L})$, then (tp, p, o), (s, p, tp) and (tp, p, tp') are triple* patterns.

Similar to the notion of a BGP, we call any finite set of triple* patterns a *BGP** (note that by this definition, every ordinary BGP is also a BGP*). While our technical report defines more expressive types of SPARQL* queries [3], in this paper we focus on the BGP* fragment of SPARQL* and, thus, define a *SPARQL* query* to be a pair (W, B) that consists of a finite set W of variables (i.e., $W \subseteq \mathcal{V}$) and a BGP* B. Likewise, in this paper, an ordinary *SPARQL query* is a pair (W, B) with $W \subseteq \mathcal{V}$ and B being a BGP.

Hereafter, we write $\mathcal{TP}^\star$ to denote the (infinite) set of all triple* patterns. Moreover, in the same spirit as they are used for RDF* triples and RDF* graphs, we overload functions Elmts^+ and T^+ for our query patterns: For every triple* pattern $tp = (s, p, o)$ we have $\mathrm{Elmts}^+(tp) = \{s, p, o\} \cup \{x \in \mathrm{Elmts}^+(tp') \,|\, tp' \in \{s, o\} \cap \mathcal{TP}^\star\}$, and for every BGP* B it is $\mathrm{Elmts}^+(B) = \bigcup_{tp \in B} \mathrm{Elmts}^+(tp)$ and $\mathrm{T}^+(B) = B \cup \big(\mathrm{Elmts}^+(B) \cap \mathcal{TP}^\star\big)$.

Example 5. The SPARQL* query in Example 3 can be captured formally by the pair $Q_{\mathsf{ex}} = (\{\textit{?x}, \textit{?age}, \textit{?src}\}, B_{\mathsf{ex}})$ with a BGP* $B_{\mathsf{ex}} = \{((\textit{?x}, \mathsf{age}, \textit{?age}), \mathsf{source}, \textit{?src})\}$. Note that B_{ex} consists of a single triple* pattern, whereas $\mathrm{T}^+(B_{\mathsf{ex}})$ consist of two: $\mathrm{T}^+(B_{\mathsf{ex}}) = \{\, ((\textit{?x}, \mathsf{age}, \textit{?age}), \mathsf{source}, \textit{?src}),\ (\textit{?x}, \mathsf{age}, \textit{?age})\,\}$.

2.3 SPARQL* Semantics

Before defining a query semantics of SPARQL*, we recall that the semantics of SPARQL is defined based on the notion of *solution mapping*s [2,7], that is, partial mappings $\mu\colon \mathcal{V} \to \big(\mathcal{I} \cup \mathcal{B} \cup \mathcal{L}\big)$. For SPARQL* we extend this notion to so-called *solution* mappings* that may bind variables not only to IRIs, blank nodes, or literals, but also to RDF* triples. Hence, a *solution* mapping* η is a partial mapping $\eta : \mathcal{V} \to \big(\mathcal{T}^\star \cup \mathcal{I} \cup \mathcal{B} \cup \mathcal{L}\big)$.

Given a solution* mapping η and a BGP* B, we write $\eta[B]$ to denote the BGP* obtained from B by replacing the variables in B according to η (variables that are not in $\mathrm{dom}(\eta)$ are not replaced). Now we are ready to define the following evaluation function that formalizes a (set) semantics of SPARQL* queries over RDF* graphs.

Definition 3. Let Q be a SPARQL* query (W, B). The **evaluation** of Q over an RDF* graph $G^\star$, denoted by $[\![Q]\!]_{G^\star}$, is a set of solution* mappings that is defined as follows:

$$[\![Q]\!]_{G^\star} = \big\{ \eta_{|W} \,\big|\, \mathrm{dom}(\eta) = \mathrm{vars}(B) \text{ and } \eta[B] \subseteq \mathrm{T}^+(G^\star) \big\},$$

where $\eta_{|W}$ is the solution* mapping that is the restriction of η to the variables in W, and $\mathrm{vars}(B)$ denotes the set of all variables mentioned in B, i.e., $\mathrm{vars}(B) = \mathrm{Elmts}^+(B) \cap \mathcal{V}$.

Example 6. For Q_{ex} (cf. Example 5) over G_{ex} (cf. Example 4) we have $[\![Q_{\mathsf{ex}}]\!]_{G_{\mathsf{ex}}} = \{\eta_1\}$ with $\eta_1(\textit{?x}) = \mathsf{bob}$, $\eta_1(\textit{?age}) = \textit{23}$, and $\eta_1(\textit{?src}) = \mathsf{listing.html}$. Consider another query, $Q_{\mathsf{ex2}} = (\{\textit{?t}, \textit{?src}\}, B_{\mathsf{ex2}})$ with $B_{\mathsf{ex2}} = \{(\textit{?t}, \mathsf{source}, \textit{?src})\}$. Then, $[\![Q_{\mathsf{ex2}}]\!]_{G_{\mathsf{ex}}} = \{\eta_2\}$ with $\eta_2(\textit{?t}) = t$ and $\eta_2(\textit{?src}) = \mathsf{listing.html}$ (with the RDF* triple t as given in Example 4).

Note that the evaluation function in Definition 3 is a direct extension of the evaluation function that Pérez et al. have defined for ordinary SPARQL queries [7]. That is, for a SPARQL* query whose BGP* is an ordinary BGP and an RDF* graph G that is an ordinary RDF graph (i.e., $\mathrm{T}^+(G) = G$), both functions return the same result (the solution* mappings returned as per Definition 3 are ordinary solution mappings in this case).

2.4 Redundancy in RDF* Graphs

We conclude the formal introduction of RDF* and SPARQL* by highlighting a noteworthy characteristic of the RDF* data model. Consider an RDF* graph $G^\star$ that contains an RDF* triple $t \in G^\star$ such that t is a metadata triple and, thus, mentions another RDF*

triple t'; i.e., $t' \in \mathrm{Elmts}^{+}(t)$. Additionally, t' may also be an element of $G^\star$ (in addition to being contained indirectly in $G^\star$ due to its containment in t), or t' may not be contained directly in $G^\star$. We consider both options ($t' \in G^\star$ and $t' \notin G^\star$) to be equivalent in terms of the information content carried by $G^\star$. This understanding is reflected in how we define the SPARQL* query semantics in Definition 3 (observe that the definition imposes the condition that $\eta[B] \subseteq \mathrm{T}^{+}(G^\star)$ rather than $\eta[B] \subseteq G^\star$). As a consequence, for the result of any query over $G^\star$, it makes no difference whether $t' \in G^\star$ or $t' \notin G^\star$. Hence, having t' contained directly in $G^\star$ is a form of redundancy. In the remainder of this section we show that implementation techniques for our proposal are free to choose how to deal with such redundancies in order to achieve performance gains.

Informally, we say that an RDF* graph that does not contain such redundancies is *redundancy free*; on the other hand, adding such redundancies will eventually result in a *redundancy-complete* graph. The following definition captures these notions formally.

Definition 4. An RDF* triple t' is **redundant in** an RDF* graph $G^\star$ if $t' \in G^\star$ and there exists another RDF* triple $t \in G^\star$ such that $t' \in \mathrm{Elmts}^{+}(t)$. An RDF* graph $G^\star$ is **redundancy free** if there does not exist any RDF* triple that is redundant in $G^\star$. An RDF* graph $G^\star$ is **redundancy complete** if for every RDF* triple $t \in G^\star$, every RDF* triple $t' \in (\mathrm{Elmts}^{+}(t') \cap \mathcal{T}^\star)$ is redundant in $G^\star$ (hence, $t' \in G^\star$).

Note that RDF* graphs that are ordinary RDF graphs are both redundancy free and redundancy complete. Every other RDF* graph may be augmented with redundant triples and, similarly, redundancy may be removed. The following definition captures such operations and Proposition 1 below shows properties of the resulting RDF* graphs.

Definition 5. Let $G^\star$ be an RDF* graph. The **redundancy augmentation** of $G^\star$ is an RDF* graph $G^\star_{\mathsf{aug}}$ that is defined as $G^\star_{\mathsf{aug}} = \mathrm{T}^{+}(G^\star)$. The **redundancy elimination** of $G^\star$ is an RDF* graph $G^\star_{\mathsf{el}}$ that is defined as $G^\star_{\mathsf{el}} = \{t \in G^\star \mid t \text{ is not redundant in } G^\star\}$.

Proposition 1. *Let $G^\star$ be an RDF* graph, and $G^\star_{\mathsf{aug}}$ and $G^\star_{\mathsf{el}}$ be the redundancy augmentation and redundancy elimination of $G^\star$, respectively. The following properties hold:*
1. *$G^\star_{\mathsf{aug}}$ is redundancy complete, and $G^\star_{\mathsf{el}}$ is redundancy free.*
2. *If $G^\star$ is redundancy complete, then $G^\star_{\mathsf{aug}} = G^\star$.*
3. *If $G^\star$ is redundancy free, then $G^\star_{\mathsf{el}} = G^\star$.*
4. *For every SPARQL* query Q, it holds that $[\![Q]\!]_{G^\star_{\mathsf{aug}}} = [\![Q]\!]_{G^\star}$ and $[\![Q]\!]_{G^\star_{\mathsf{el}}} = [\![Q]\!]_{G^\star}$.*

Proof. The properties in Proposition 1 follow trivially from Definitions 3–5, where the crux of the fourth property is that $\mathrm{T}^{+}(G^\star) = \mathrm{T}^{+}(G^\star_{\mathsf{aug}}) = \mathrm{T}^{+}(G^\star_{\mathsf{el}})$.

The fourth property in Proposition 1 is perhaps the most relevant in practice because approaches to implement our proposal may leverage the highlighted query result equivalences. For instance, a physical storage model for RDF* might be designed to explicitly capture all redundant triples if this allows for more efficient query execution techniques.

3 Desirable Properties of RDF*-to-RDF Mappings

While the previous section presents all the necessary formal foundations to implement our proposal natively, the remainder of this paper provides the additional foundations needed for wrapper-based implementations. We begin with general definitions that capture desirable properties of the mappings that a wrapper may use to convert data and queries. The next section shall then introduce specific mappings.

As indicated in Section 1, we assume that every wrapper uses two mappings. The first mapping is a function that maps every RDF* graph to an ordinary RDF graph. Hereafter, we refer to such a function as an *RDF*-to-RDF mapping*. As is typical for such kind of mappings [10], we consider it desirable for an RDF*-to-RDF mapping to be *information preserving*; informally, this property requires the existence of an inverse mapping [10]. This inverse mapping can then be used to reconstruct the original RDF* graph from the resulting RDF graph. The following formal definition captures a notion of information preservation that shall allow us to indicate that specific RDF*-to-RDF mappings have this property only for designated subsets of all possible RDF* graphs.

Definition 6. Let $\mathcal{G}^\star$ be a (possibly infinite) set of RDF* graphs. An RDF*-to-RDF mapping m is **information preserving for** $\mathcal{G}^\star$ if there exists a computable mapping m' from RDF graphs to RDF* graphs such that for every $G^\star \in \mathcal{G}^\star$ it holds that $m'(m(G^\star)) = G^\star$.

The second mapping used by wrapper-based implementations of our proposal is concerned with rewriting SPARQL* queries into ordinary SPARQL queries. Hence, we define such *SPARQL*-to-SPARQL mappings* to be functions that map every SPARQL* query to a SPARQL query. Informally, given a SPARQL query resulting from the application of such a mapping and an RDF graph produced by a corresponding RDF*-to-RDF mapping, it is desirable that evaluating the SPARQL query over the RDF graph returns a query result that is equivalent to the result of evaluating the original SPARQL* query over the original RDF* graph. However, true equivalence cannot be achieved in all cases because evaluating a SPARQL* query may result in solution* mappings that map variables to RDF* triples, which is impossible with ordinary solution mappings (as obtained by evaluating ordinary SPARQL queries). To accommodate for this difference we shall define a notion of query result equivalence that is based on reducing solution* mappings to ordinary solution mappings. To define this reduction formally we assume an auxiliary mapping that maps every RDF* triple to a distinct IRI or blank node:

Definition 7. A **triple-to-ID mapping** id is an injective function $id \colon \mathcal{T}^\star \rightarrow (\mathcal{I} \cup \mathcal{B})$.

Definition 8. Let id be a triple-to-ID mapping. Then, the id-specific **reduction** of a solution* mapping η, denoted by $\mathsf{reduce}^{id}(\eta)$, is a solution mapping μ such that $\mathrm{dom}(\mu) = \mathrm{dom}(\eta)$ and for each variable $?v \in \mathrm{dom}(\eta)$ it holds that $\mu(?v) = id(\eta(?v))$ if $\eta(?v)$ is an RDF* triple and $\mu(?v) = \eta(?v)$ if $\eta(?v)$ is not an RDF* triple. Moreover, the id-specific **reduction** of a set Ω of solution* mappings, denoted by $\mathsf{reduce}^{id}(\Omega)$, is a set of solution mappings defined by $\mathsf{reduce}^{id}(\Omega) = \bigcup_{\eta \in \Omega} \mathsf{reduce}^{id}(\eta)$.

Example 7. Let t be the RDF* triple $(\mathsf{bob}, \mathsf{age}, \mathit{23})$ in Example 4. Assume a triple-to-ID mapping id_{ex} s.t. $id_{\mathsf{ex}}(t) = b$ where b is an arbitrary blank node. Then, the id_{ex}-specific reduction of solution* mapping η_2, as given in Example 6, is a solution mapping μ_2 with $\mu_2(\mathit{?src}) = \mathsf{listing.html}$ and $\mu_2(\mathit{?t}) = b$ (recall from Example 6 that we have $\eta_2(\mathit{?t}) = t$).

We now have all the necessary to define the notion of query result preservation. Similar to our definition of information preservation, we define this property with the possibility to indicate limitations on the scope for which the property is supposed to hold.

Definition 9. Let $\mathcal{G}^\star$ and $\mathcal{Q}^\star$ be (possibly infinite) sets of RDF* graphs and of SPARQL* queries, respectively, dm be an RDF*-to-RDF mapping, and id be a triple-to-ID mapping. A SPARQL*-to-SPARQL mapping qm is **query result preserving for** $\mathcal{G}^\star$ **and** $\mathcal{Q}^\star$ **under** dm **and** id if for every SPARQL* query $Q \in \mathcal{Q}^\star$ and every RDF* graph $G^\star \in \mathcal{G}^\star$, it holds that $[\![qm(Q)]\!]_{dm(G^\star)} = \mathsf{reduce}^{id}([\![Q]\!]_{G^\star})$.

4 Mapping RDF* to Standard RDF Reification

This section provides the foundations of creating wrappers that use RDF reification [4] to implement RDF* and SPARQL* on top of any existing RDF triple store. More specifically, we define a suitable RDF*-to-RDF mapping and a corresponding SPARQL*-to-SPARQL mapping, and we show that these mappings possess the desirable properties.

The idea of the RDF*-to-RDF mapping is to flatten RDF* triples by using the RDF reification vocabulary such that, for instance, the RDF* data in Example 2 would be mapped to the RDF data in both Figures 1a and 1c. As a basis for defining this mapping we use the notion of the triple-to-ID mapping (cf. Definition 7) and introduce another auxiliary concept; namely, a notion of *reducing* an RDF* triple to an ordinary RDF triple by applying a given triple-to-ID mapping to the subject and object of the RDF* triple.

Definition 10. Given a triple-to-ID mapping id, the id-specific **reduction** of an RDF* triple $t = (x_1, x_2, x_3)$, denoted by $\mathsf{reduce}^{id}(t)$, is an RDF triple (x'_1, x'_2, x'_3) s.t. $x'_2 = x_2$ and for each $i \in \{1, 3\}$ we have that $x'_i = id(x_i)$ if x_i is an RDF* triple and else $x'_i = x_i$.

Example 8. Consider the triple-to-ID mapping id_{ex} as in Example 7 and RDF* triple t as in Example 4. The id_{ex}-specific reduction of RDF* triple $(t, \mathsf{source}, \mathsf{listing.html})$ is the RDF triple $(b, \mathsf{source}, \mathsf{listing.html})$, whereas $\mathsf{reduce}^{id_{\mathsf{ex}}}(t)$ simply is t itself.

We now define the RDF*-to-RDF mapping. Informally, for any given RDF* graph, this mapping returns an RDF graph that consists of two subgraphs. The first subgraph contains RDF triples that are the reductions of all RDF* triples in the given RDF* graph (including the triples that are nested in some metadata triples). The second subgraph contains RDF triples that are added to reify every RDF* triple that is nested in some metadata triple in the given RDF* graph. The formal definition is given as follows.

Definition 11. Let id be a triple-to-ID mapping. The id-specific **standard RDF representation of RDF*** is an RDF*-to-RDF mapping m that, for every RDF* graph $G^\star$, gives:

$$m(G^\star) \;=\; \big\{\mathsf{reduce}^{id}(t) \,|\, t \in \mathrm{T}^{+}(G^\star)\big\} \;\cup\; \textstyle\bigcup_{t \in (\mathrm{Elmts}^{+}(G^\star) \cap \mathcal{T}^\star)} \mathrm{reif}^{id}(t),$$

where for every RDF* triple t, with $\mathsf{reduce}^{id}(t) = (s, p, o)$, we have:

$$\begin{aligned}\mathrm{reif}^{id}(t) = \big\{\, & (id(t), \mathsf{rdf{:}type}, \mathsf{rdf{:}Statement}), \quad (id(t), \mathsf{rdf{:}subject}, s), \\ & (id(t), \mathsf{rdf{:}predicate}, p), \quad (id(t), \mathsf{rdf{:}object}, o) \,\big\}.\end{aligned}$$

Example 9. Given the triple-to-ID mapping id_{ex} in Example 8, it is trivial to see that the standard RDF representation of our example RDF* graph $G^\star_{\mathsf{ex}}$ (cf. Example 4) is the RDF graph that may be serialized in Turtle as given in both Figures 1a and 1c together.

We note that the triple-to-ID mapping used in Definition 11 may map triples to IRIs or blank nodes that are already used in the given RDF* graph. If this is the case, that is, if the image of the triple-to-ID mapping overlaps with the set of IRIs and blank nodes in the RDF* graph, we say that the triple-to-ID mapping and the RDF* graph are in conflict. Formally, a triple-to-ID mapping id is *in conflict* with an RDF* graph $G^\star$ if there exists an RDF* triple $t \in \mathrm{T}^{+}(G^\star)$ such that $id(t) \in \mathrm{Elmts}^{+}(G^\star)$. From now on we assume triple-to-ID mappings that are *not* in conflict with any given RDF* graph.

Now we can show that the mapping in Definition 11 possesses the information preservation property as long as the mapping is applied only to redundancy-complete RDF* graphs that do not use the RDF reification vocabulary (i.e., for such a graph $G^\star$ it must hold that $\mathrm{Elmts}^{+}(G^\star) \cap \{\mathsf{rdf{:}Statement}, \mathsf{rdf{:}subject}, \mathsf{rdf{:}predicate}, \mathsf{rdf{:}object}\} = \emptyset$).

Theorem 1. *Let id be a triple-to-ID mapping and $\mathcal{G}^\star$ be a (possibly infinite) set of RDF* graphs such that for every $G^\star \in \mathcal{G}^\star$ it holds that (i) id is not in conflict with $G^\star$, (ii) $G^\star$ is redundancy complete, and (iii) $G^\star$ does not use the RDF reification vocabulary. The id-specific standard RDF representation of RDF* is information preserving for $\mathcal{G}^\star$.*

Proof (Sketch). To prove Theorem 1 we define a mapping m' as required by Definition 6. To this end, for every RDF graph G, let $\mathfrak{R}(G)$ to be the set of all 5-tuples (x, s, p, o, R) with $x, s, p, o \in (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$ and $R \subseteq G$ such that

$R = \{(x, \mathsf{rdf{:}type}, \mathsf{rdf{:}Statement}), (x, \mathsf{rdf{:}subject}, s), (x, \mathsf{rdf{:}predicate}, p), (x, \mathsf{rdf{:}object}, o)\}$.

Furthermore, let $N(G) = \{t \in G \,|\, t \notin R \text{ for all } (x, s, p, o, R) \in \mathfrak{R}(G)\}$. Note that if G is the id-specific standard RDF representation of some $G^\star \in \mathcal{G}^\star$, then it holds that $N(G)$ does not use the RDF reification vocabulary and for every $(x, s, p, o, R) \in \mathfrak{R}(G)$ we have that x is in the image of mapping id and x is unique. These properties follow from the fact that id is injective and $G^\star$ does not use the RDF reification vocabulary. Due to the latter property (the uniqueness of all x), we may turn $\mathfrak{R}(G)$ into a mapping that maps every x to a triple with the corresponding s, p, and o, respectively. Formally: let $\mathsf{dereif}^{\mathfrak{R}(G)}$ be a mapping with $\mathrm{dom}(\mathsf{dereif}^{\mathfrak{R}(G)}) = \{x \,|\, (x, s, p, o, R) \in \mathfrak{R}(G)\}$ such that for every $(x, s, p, o, R) \in \mathfrak{R}(G)$ we have that $\mathsf{dereif}^{\mathfrak{R}(G)}(x) = (s, p, o)$. As our last preliminary, we introduce a recursively-defined mapping $\mathsf{reverse}^{\mathfrak{R}(G)}$ that maps every RDF triple $t = (x_1, x_2, x_3)$ to an RDF* triple $t^\star = (x_1^\star, x_2^\star, x_3^\star)$ such that for each $i \in \{1, 2, 3\}$ we have that $x_i^\star = \mathsf{reverse}^{\mathfrak{R}(G)}\big(\mathsf{dereif}^{\mathfrak{R}(G)}(x_i)\big)$ if $x_i \in \mathrm{dom}(\mathsf{dereif}^{\mathfrak{R}(G)})$ and $x_i^\star = x_i$ if $x_i \notin \mathrm{dom}(\mathsf{dereif}^{\mathfrak{R}(G)})$. Now we have everything to define m' for every RDF graph G: $m'(G) = \{\mathsf{reverse}^{\mathfrak{R}(G)}(t) \,|\, t \in N(G)\}$. The crux of showing that m' is computable is the fact that, for any RDF graph G, the set $\mathfrak{R}(G)$ can be generated by one pass over G. The crux of showing the correctness of m' is that the mapping $\mathsf{reverse}^{\mathfrak{R}(G)}$ is the reverse operation of our notion of reducing RDF* triples (cf. Definition 10). $\square$

We now turn to the SPARQL*-to-SPARQL mapping. The idea of this mapping is to use the RDF reification vocabulary in the same manner as done by the RDF*-to-RDF mapping in Definition 11. For instance, the SPARQL* query in Example 3 would be mapped to the SPARQL query in Figure 1b. Since this paper focuses only on the BGP* fragment of SPARQL*, we also define the mapping only for this fragment. However, since this fragment is the basic building block of any other, more expressive fragment, it is easy to extend the mapping to other fragments by using a recursive definition.

To define the mapping we first need two auxiliary mappings whose purpose is similar to that of the triple-to-ID mapping and the reduction of RDF* triples, respectively.

Definition 12. A **TP*-to-var mapping** var is an injective function $var\colon \mathcal{TP}^\star \rightarrow \mathcal{V}$.

Definition 13. Let var be a TP*-to-var mapping. The var-specific **reduction** of a triple* pattern $tp^\star = (x_1, x_2, x_3)$, denoted by $\mathsf{reduce}^{var}(tp^\star)$, is a triple pattern (x_1', x_2', x_3') s.t. for all $i \in \{1, 2, 3\}$ we have that $x_i' = var(x_i)$ if x_i is a triple* pattern and else $x_i' = x_i$.

Example 10. Assume a TP*-to-var mapping var_{ex} such that $var_{\mathsf{ex}}(tp) = \textit{?r}$ for the triple* pattern $tp = (\textit{?x}, \mathsf{age}, \textit{?age})$. Then, the var_{ex}-specific reduction of the triple* pattern in Example 5, $((\textit{?x}, \mathsf{age}, \textit{?age}), \mathsf{source}, \textit{?src})$, is the triple pattern $(\textit{?r}, \mathsf{source}, \textit{?src})$.

For TP*-to-var mappings we also consider a notion of conflict—in this case w.r.t. a BGP*. That is, a TP*-to-var mapping var is *in conflict* with a BGP* B if there exists a triple* pattern $tp \in \mathrm{T}^{+}(B)$ such that $var(tp) \in \mathrm{Elmts}^{+}(B)$. As for the triple-to-ID mappings, hereafter, we assume only TP*-to-var mappings that are *not* in conflict with any given BGP*. Now, we are ready to define the SPARQL*-to-SPARQL mapping.

Definition 14. Let var be a TP*-to-var mapping. Then, the var-specific **standard RDF representation of SPARQL*** is a SPARQL*-to-SPARQL mapping m that, for every SPARQL* query $Q^\star = (W^\star, B^\star)$, is defined by $m(Q^\star) = (W, B)$ such that $W = W^\star$ and

$$B = \left\{ \mathsf{reduce}^{var}(tp^\star) \,|\, tp^\star \in \mathrm{T}^{+}(B^\star) \right\} \cup \bigcup_{tp^\star \in (\mathrm{Elmts}^{+}(B^\star) \cap \mathcal{TP}^\star)} \mathrm{reif}^{var}(tp^\star),$$

where for every triple* pattern $tp^\star$, with $\mathsf{reduce}^{var}(tp^\star) = (s, p, o)$, we have:

$$\mathrm{reif}^{var}(tp^\star) = \Big\{ \, (var(tp^\star), \mathsf{rdf{:}type}, \mathsf{rdf{:}Statement}), \quad (var(tp^\star), \mathsf{rdf{:}subject}, s), \quad (var(tp^\star), \mathsf{rdf{:}predicate}, p), \quad (var(tp^\star), \mathsf{rdf{:}object}, o) \, \Big\}.$$

The following result shows that the standard RDF representation of SPARQL* (Definition 14) is a query result preserving mapping for RDF* graphs and SPARQL* queries that do not use the RDF reification vocabulary (for queries with their BGP* B this means it must hold that $\mathrm{Elmts}^{+}(B) \cap \{\mathsf{rdf{:}Statement}, \mathsf{rdf{:}subject}, \mathsf{rdf{:}predicate}, \mathsf{rdf{:}object}\} = \emptyset$).

Theorem 2. *Let var be a TP*-to-var mapping, let $\mathcal{Q}^\star$ be a set of SPARQL* queries such that for every $(W, B) \in \mathcal{Q}^\star$ it holds that (i) var is not in conflict with B and (ii) B does not use the RDF reification vocabulary, and let $\mathcal{G}^\star$ be a set of RDF* graphs such that no $G^\star \in \mathcal{G}^\star$ uses the RDF reification vocabulary. Then, the var-specific standard RDF representation of SPARQL* is query result preserving for $\mathcal{G}^\star$ and $\mathcal{Q}^\star$ under any triple-to-ID mapping id and the id-specific standard RDF representation of RDF*.*

Proof (Sketch). Theorem 2 can be shown by an induction on the set B of triple* patterns in the queries in $\mathcal{Q}^\star$. Then, the crux of the equivalence of query results is the analogy of Definitions 11 and 14 and the restriction of Theorem 2 to cases in which the RDF*-to-RDF mapping used is an id-specific standard RDF representation of RDF*. Note that, in contrast to Theorem 1, Theorem 2 does not need to require redundancy completeness for the graphs in $\mathcal{G}^\star$ because of Property 4 in Proposition 1. □

5 Outlook

This paper provides the formal foundations of a new proposal to represent statement-level metadata and related queries in the RDF context. We consider establishing these foundations as the basis to systematically study the trade-offs of our proposal. Consequently, our future work is to conduct such a study, which will take multiple directions: First, we aim to investigate how appealing SPARQL* queries are to users in comparison to the corresponding SPARQL queries that would have to be written for the other approaches to statement-level metadata in RDF (i.e., standard RDF reification, singleton properties, and singleton named graphs). Second, we want to understand the *practical* consequences of executing SPARQL* queries based on a wrapper that employs the mappings defined in this paper. In fact, we are interested not only in wrappers for RDF reification but also for singleton properties and named graphs. To this end, the work in this paper has to be extended by defining (information preserving and query result preserving) mappings for these two approaches, which can easily be done by adapting the definitions and the results in Section 4. Third, we want to investigate approaches to implement our proposal natively. A particularly interesting idea in this context is to carry over the notion of nested triples to the physical level. Finally, as another direction for more fundamental work, we aim to compare RDF* to notions of nested relations.

References

1. R. Cyganiak, D. Wood, M. Lanthaler, G. Klyne, J. J. Carroll, and B. McBride. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, Feb. 2014.
2. S. Harris, A. Seaborne, and E. Prud'hommeaux. SPARQL 1.1 Query Language. W3C Recommendation, Mar. 2013.
3. O. Hartig and B. Thompson. Foundations of an Alternative Approach to Reification in RDF. *CoRR*, abs/1406.3399, 2014. Online: http://arxiv.org/abs/1406.3399.
4. P. J. Hayes and P. F. Patel-Schneider. RDF 1.1 Semantics. W3C Recommendation, Feb. 2014.
5. D. Hernández, A. Hogan, and M. Krötzsch. Reifying RDF: What Works Well With Wikidata? In *Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS)*, 2015.
6. V. Nguyen, O. Bodenreider, and A. P. Sheth. Don't like RDF Reification? Making Statements about Statements Using Singleton Property. In *Proceedings of the 23rd International World Wide Web Conference (WWW)*, 2014.
7. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and Complexity of SPARQL. *ACM Transactions on Database Systems*, 34(3), 2009.
8. E. Prud'hommeaux and G. Carothers. RDF 1.1 Turtle. W3C Recommendation, Feb. 2014.
9. I. Robinson, J. Webber, and E. Eifrém. *Graph Databases*. O'Reilly Media, 2013.
10. J. Sequeda, M. Arenas, and D. P. Miranker. On Directly Mapping Relational Databases to RDF and OWL. In *Proceedings of the 21st World Wide Web Conference (WWW)*, 2012.
11. P. Tsialiamanis, L. Sidirourgos, I. Fundulaki, V. Christophides, and P. Boncz. Heuristics-based Query Optimisation for SPARQL. In *Proceedings of the 15th International Conference on Extending Database Technology (EDBT)*.

Acknowledgments

The author would like to thank a number of persons for providing feedback on RDF* and SPARQL*, as well as on earlier versions of this document. In alphabetical order, these persons are: Bryan Thompson, Cosmin Basca, Grant Weddell, Juan Sequeda, Kavitha Srinivas, Maria-Esther Vidal, Michael Schmidt, Mike Personick, Peter Boncz, Tamer Özsu, and the three anonymous reviewers of AMW 2017. The author's work has been funded partially by the CENIIT program at Linköping University (project no. 17.05).