Transforming RDF-star to Property Graphs: A Preliminary Analysis of Transformation Approaches

Ghadeer Abuoda[1], Daniele Dell'Aglio[1], Arthur Keen[2] and Katja Hose[1]

[1]Department of Computer Science, Aalborg University, Aalborg, Denmark
[2]ArangoDB, San Francisco, United States

Abstract

RDF and property graph models have many similarities, such as using basic graph concepts like nodes and edges. However, such models differ in their modeling approach, expressivity, serialization, and the nature of applications. RDF is the de-facto standard model for knowledge graphs on the Semantic Web and supported by a rich ecosystem for inference and processing. The property graph model, in contrast, provides advantages in scalable graph analytical tasks, such as graph matching, path analysis, and graph traversal. RDF-star extends RDF and allows capturing metadata as a first-class citizen. To tap on the advantages of alternative models, the literature proposes different ways of transforming knowledge graphs between property graphs and RDF. However, most of these approaches cannot provide complete transformations for RDF-star graphs. Hence, this paper provides a step towards transforming RDF-star graphs into property graphs. In particular, we identify different cases to evaluate transformation approaches from RDF-star to property graphs. Specifically, we categorize two classes of transformation approaches and analyze them based on the test cases. The obtained insights will form the foundation for building complete transformation approaches in the future.

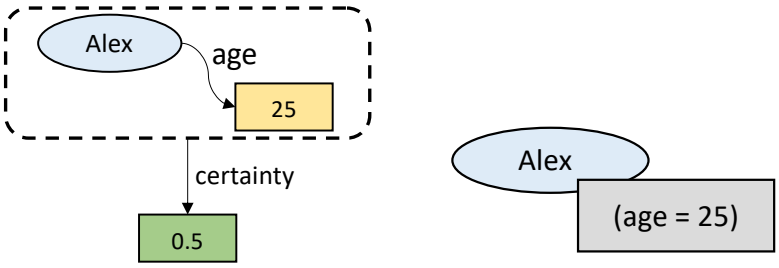
1. Introduction

The most popular models for representing knowledge graphs are: RDF[1] (Resource Description Framework) and property graphs [1] (PG). While RDF represents knowledge graphs as a set of subject-predicate-object triples, property graphs assign key-value style properties to nodes and edges.Recently, RDF-star [2] has been proposed as an extension of RDF to enable enriching RDF triples with metadata information by embedding triples in subjects or objects of other triples, which allows providing statements about statements and somewhat resembles adding properties to edges in property graphs. RDF-star is supported by a rich ecosystem of data management systems and standards, most notably systems such as Stardog, OpenLink's Virtuoso, Ontotext GraphDB, AllegroGraph, Apache Jena, and more recently also Oxigraph, but also query standards, such as SPARQL[2] and its extension SPARQL-star[3] as well as RDF Schema, which allows

QuWeDa 2022: 6th Workshop on Storing, Querying and Benchmarking Knowledge Graphs at ISWC, October 23, 2022, virtual

gsmas@cs.aau.dk (G. Abuoda); dade@cs.aau.dk (D. Dell'Aglio); arthur@arangodb.com (A. Keen); khose@cs.aau.dk (K. Hose)

[1]RDF 1.1 Primer: https://www.w3.org/TR/rdf11-primer/
[2]SPARQL 1.1 Query Language: https://www.w3.org/TR/sparql11-query/
[3]SPARQL-star Query Language: https://w3c.github.io/rdf-star/cg-spec/editors_draft.html#sparql-star



(a) RDF-star Graph **(b) A property graph**

Figure 1: Graphical Representation of Listing 1 as (a) RDF-star and (b) Property Graph

Listing 1: An example RDF-star graph in Turtle star format

```
@prefix ex: <http://example.org/> .
<<ex:Alex ex:age 25>>  ex:certainty 0.5 .
```

describing classes of RDF resources and properties[4]. In contrast, many graph database systems, such as Neo4j, TigerGraph, JanusGraph, RedisGraph, and SAP HANA are based on different variations of the property graph model [3] and different query languages [4, 5]. Unfortunately, RDF-star graphs and property graphs are not entirely compatible with one another. Although they both describe data through graphs, their underlying models and semantics are different, leading to many data interoperability issues [6, 7, 8]. Metadata or edge properties in RDF-star can be modeled as separate nodes or RDF-star triples. In contrast, edge properties can only be represented as literal key-value pairs in property graphs. In general, it is challenging to transform an RDF-star graph fully into a property graph because of the rich expressiveness of the former. The heterogeneity between the two models and their frameworks makes it necessary to study their interoperability, i.e., the ability to map one model to another for data exchange and sharing [7].

The mapping between the two models is crucial for data exchange, data integration as well as reusability of systems and tools between the frameworks. RDF-star, specifically the RDF model, is recognized as a web-native model that supports data exchange and sharing across different sources because of its formal semantics and the universal uniqueness of resources using IRIs. RDF is a common and flexible model for knowledge representation, and that is exemplified by knowledge graphs that cover a broad set of domains, such as DBpedia [9], YAGO [10], and Wikidata [11]. On the contrary, even with the wide adoption of property graph engines, property graphs lack many essential features, such as a schema language, a standard query language, standard data serialization formats, etc. Achieving interoperability and reliable transformations between the two frameworks will finally enable us to exploit the benefits of both models.

The transformation of property graphs to RDF-star has been explored recently [12], and basic transformation rules for property graphs to RDF-star were proposed [2, 13]. However, the latter does not cover all RDF-star constructs and allows for multiple alternatives.

[4]RDF Schema 1.1: https://www.w3.org/TR/rdf-schema/

Consider, for instance, the example illustrated in Figure 1(a). If we start with the triple (`ex:Alex`, `ex:age`, `25`) in Listing 1, then we could represent the RDF element (Alex) as a node in a property graph, as shown in Figure 1(b). This node would then have a property (age,25) and the RDF triple would be represented by a single node in a property graph. However, if we have a single node without an edge, we cannot represent the metadata about the original RDF triple (ex:certainty 0.5) in the property graph. Studying such cases, this paper makes the following contributions:

- We identify two alternative approaches of transformations: RDF-topology-preserving and Property-Graph transformation.
- We define a set of test cases capturing the diverse RDF-star constructs that have to be considered when transforming RDF-star to property graphs.
- Using the test cases, we systematically evaluate alternative mapping approaches and identify their shortcomings.

This paper is structured as follows: while Section 2 introduces preliminaries, Section 3 discusses related work. Section 4 presents alternative transformation approaches. Afterwards, Section 5 provides details on our test cases, which we use in Section 6 to identify and discuss shortcomings of transformation approaches. Section 7 concludes the work with an outlook for future work.

2. Preliminaries

In this section, we formally introduce RDF[1], RDF-star [2], and property graphs [1].

2.1. Resource Description Framework (RDF).

RDF is a W3C standard data model that represents information as a set of statements. Each statement denotes a typed relation between two resources.

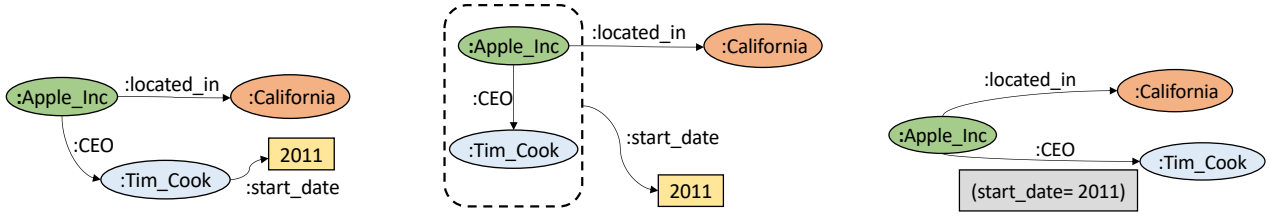
Definition 1 (RDF statement). *Let I, B and L be the disjoint sets of Internationalized Resource Identifiers (IRIs), blank nodes and literals. An* RDF statement *is a triple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$, and it indicates that s and o (*subject *and* object*, resp.) are in a relation p (*predicate*).*

In this paper, we consider two types of RDF statements that we distinguish based on whether the object is an IRI or a literal. *Object property statements* are RDF statements $(s, p, o) \in (I \cup B) \times I \times (I \cup B)$, while *datatype property statements* are RDF statements $(s, p, o) \in (I \cup B) \times I \times L$. An RDF graph containing three RDF statements is shown in Listing 2 serialized in Turtle[5], and visually in Figure 2(a) as a graph. The first two statements are object property statements. The first statement describes two resources, ex:Apple_Inc and ex:California, related by the predicate ex:located_in. The second statements indicates that ex:Apple_Inc has ex:Tim_Cook as a ex:CEO. The last statement is a datatype property statement, and it indicates that ex:Tim_Cook has the literal "2011" as the value of the ex:start_date predicate.

[5]RDF Turtle: https://www.w3.org/TR/turtle/

Listing 2: An RDF graph in Turtle format

```
@prefix ex: <http://example.org/> .
ex:Apple_Inc ex:located_in ex:California .
ex:Apple_Inc ex:CEO ex:Tim_Cook .
ex:Tim_Cook  ex:start_date 2011 .
```



(a) RDF Graph **(b) RDF-star Graph** **(c) Property Graph**

Figure 2: Example in RDF, RDF-star, and Property Graphs

2.2. RDF-star

Looking at the RDF graph in Figure 2(a), one can spot some imprecise data modelling choices: stating that Tim Cook started in 2011 is not totally correct, as he started in 2011 his role as CEO of Apple. In other words, one should associate the starting date to the statement (ex:Apple_Inc, ex:CEO, ex:Tim_Cook), as depicted in Figure 2(b). There are several ways to implement this idea in RDF, such as RDF reification [2], singleton properties [14], and named graphs [15]. However, these mechanisms have significant shortcomings [16, 17, 18].

Listing 3: An RDF-Star Graph in Turtle-Star Format

```
@prefix ex: <http://example.org/> .
  ex:Apple_Inc ex:located_in ex:California .
<<ex:Apple_Inc ex:CEO  ex:Tim_Cook>>  ex:start_date 2011 .
```

A solution to overcome such shortcomings was recently proposed by Hartig et al. with RDF-star [2, 12]. RDF-star extends RDF by letting RDF statements be subjects or objects in other statements. Listing 3 shows an RDF-star document serialized in Turtle-star [2]. The first statement is a compliant RDF statement (it appeared also in Listing 2). The second statement indicates that ex:Apple_Inc appointed ex:Tim_Cook as ex:CEO in 2011. We formally define an RDF-star statement as follows.

Definition 2 (RDF-star statement). *Let $s \in I \cup B$, $p \in I$, $o \in I \cup B \cup L$. An RDF-star statement is a triple defined recursively as:*

- *Any RDF statement (s, p, o) is an RDF-star statement;*
- *Let t and $\bar{t}$ be RDF-star statements. Then, (t, p, o), (s, p, t) and $(t, p, \bar{t})$ are RDF-star statements, also known as* asserted statement. *t and $\bar{t}$ are called* embedded *or* quoted *statements.*

2.3. Property Graphs

A property graph (PG) is a graph where nodes and edges can have multiple properties, represented as key-value pairs. Figure 2(c) illustrates the graph described in the above section as a property graph. In this case, the starting date of Tim Cook as the CEO of Apple Inc is reported as a key-value property on the :CEO edge. PGs have not a unique and standardized model; each PG engine proposes its data model. A generic PG model definition is proposed by [3].

Definition 3 (Property Graph). *Let L be the set of the labels, PN be the set of property names, and D be the set of property values. A property graph G is an edge-labeled directed multi-graph such that $G = (N, E, edge, lbl, P, \sigma)$, where:*

- *N is a set of nodes,*
- *E is a set of edges between nodes, such that $N \cap E = \emptyset$*
- *$edge : E \rightarrow (N \times N)$ is a total function that associates each edge in E with a pair of nodes in N. If $edge(e_1) = (n_1, n_2)$, n_1 is the source node and n_2 is the target node.*
- *$lbl : (N \cup E) \rightarrow \mathcal{P}(L)$ is a function that associates each edge or node with a set of labels.*
- *$\sigma : (N \cup E) \rightarrow \mathcal{P}(P)$ is a function that associates a node or edge with a non-empty set of properties P defined as a set of key-value pairs (k, v) where $k \in PN$ and $v \in D$*

To ease all approaches' output representation, we map any IRI to a distinct string representing a local name. Given I, a set of all IRIs, $localName$ is a function that maps an IRI to a string that represents the local name of an RDF resource[6]. For example, the local name for the RDF resource (http://example.com/meets) $localName$("http://example.com/meets") is "`meets`". We will use this function in the output representation in Section 6.

3. Related Work

We can distinguish between related work on converting between (i) RDF and PG and (ii) RDF-star and PG.

RDF and PG. Angles et al. [13] propose three variations of transforming RDF into PG optionally in consideration of schemas: simple, generic, and complete. The authors formally show that two of the proposed mapping approaches (generic and complete) satisfy the property of information preservation, i.e., there exist inverse mappings that allow recovering the original dataset without information loss. The evaluation in this paper (Section 6) includes the schema-independent mapping referred to as the *Generic Database Mapping* using the authors' implementation (RDF2PG[7]). Although our focus is on RDF-star (instead of RDF), we include this approach since it provides the basic formalities and implementation that can be extended to support RDF-star.

In the opposite direction, Bruyat et al. [19] propose PREC[8], a library that enables tranformation PGs into RDF graphs. The authors built a uniform graph model to describe the structure of

[6] In Neo4j, the user can configure the local name of RDF terms such as subPropertyOf, subClassOf, Class, etc.
[7] https://github.com/renzoar/rdf2pg
[8] https://bruju.github.io/PREC/

Listing 4: RDF triples

```
@prefix ex: <http://example.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
ex:book  ex:publish_date "1963-03-22"^^xsd:date .
ex:book  ex:pages        "100"^^xsd:integer .
ex:book  ex:cover        20 .
ex:book  ex:index        "55" .
```

the property graph in RDF terms. PREC uses a *context* in RDF-star format that describes the mappings between the terms used in the PG model and IRIs. The user can define a template representing the different properties and edges in the resulting RDF graph. Despite using RDF-star internally, the approach does not support mapping PGs to RDF-star graphs.

RDF-star and PG. Hartig et al. [2] propose two approaches for transforming RDF-star to PG. The first approach maps ordinary RDF triples to edges in the PG. Metadata triples are then represented as edge properties. Our analysis (Section 6) includes this approach using the authors' implementation (RDF-star Tools[9]). The second approach treats datatype and object property statements differently. The former are transformed into node properties and the latter into edges. This approach, however, is limited in mapping embedded triples, and it is not implemented in the RDF-star Tools library.

Neosemantics is a well-known project to import RDF data into Neo4J, implemented as a Neo4j plug-in[10]. The implementation was only recently extended to include an RDF-star importing feature. As we will see in Section 6 importing RDF-star into PG using this transformation is lossy and does not cover all cases.

In the other direction, Khayatbashi et al. [12] present an analysis evaluating three transformation approaches from PG to RDF, including an RDF-star approach. As a part of the study, the authors evaluated the performance of querying the generated RDF graphs in multiple triple stores. They found that there is no clear best mapping in terms of execution time; the performance of the queries over RDF and RDF-star graphs resulting from the mapping varies compared to their equivalent pure RDF representations.

4. Transformation Approaches from RDF to Property Graphs

Analyzing the approaches discussed in Section 3, we can extrapolate two principle approaches: *RDF-topology Preserving Transformation (RPT)* and *Property Graph Transformation (PGT)*. RPT tries to preserve the RDF-star graph structure by transforming each RDF statement into an edge in the PG. PGT, on the other hand, ensures that datatype property statements are mapped to node properties in the PG. In what follows, we first explain how these approaches transform RDF triples into PG and afterwards how the basic algorithms can be extended to support RDF-star.

Consider the example in Listing 4 with multiple *datatype property* statements describing the RDF resource (ex:book). Figure 3 shows graphical visualizations of the property graphs

[9] https://github.com/RDFstar/RDFstarTools
[10] Neosemantics: https://neo4j.com/labs/neosemantics/

generated by the two approaches: RPT (a) and PGT (b).

RPT, for example, converts the triple (ex:book,ex:index,55) into two nodes (ex:book) and (55), connected by an edge (ex:index). All other triples involving RDF resources, blank nodes, or literal values can be transformed in a similar way so that we obtain the PG in Figure 3(a). Algorithm 1 formalizes the RPT approach; for each triple it always creates a node for the subject (line 3) and the object (line 5) with an edge connecting them (line 12) – of course avoiding duplicate nodes for the same IRIs.

For the same example (Listing 4), PGT creates the PG in Figure 3(b) consisting of a single node respresenting the RDF resource (ex:book) with multiple properties representing property-object pairs from the RDF statements, such as (ex:index,55). Distinguishing between datatype and object property statements, this approach transforms object property statements to edges and datatype property statements to properties of the node representing the subject. Unlike RPT, the resulting PG nodes represent only RDF resources or blank nodes while literal objects will become properties. PGT is more formally sketched in Algorithm 2, which first checks the type of the statement's object (line 5) and based on that decides to either create a node (if it does not yet exist, line 6) or a property (line 13).

Algorithm 1: RPT

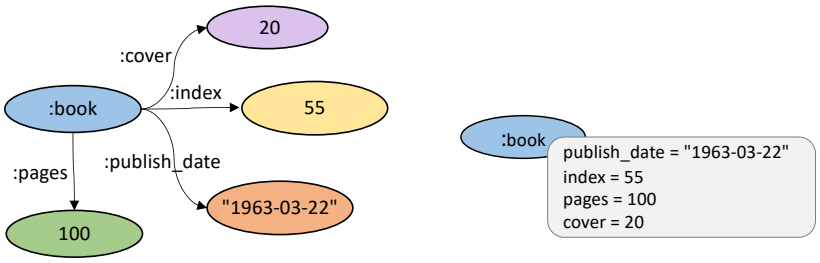
Input: A set of RDF Triples T
Output: A Property Graph $Pg = (N, E, edge, lbl, P, \sigma)$

1: $Pg \leftarrow \emptyset$
2: **for** $t \in T$, such that $t =< s, p, o >$ **do**
3: $N = N \cup \{s\}$
4: $lbl(s) =$ {"RDF resource"}
5: $N = N \cup \{o\}$
6: **if** o is an RDF resource **then**
7: $lbl(o) =$ {"RDF resource"}
8: **else**
9: $lbl(o) =$ {"Literal"}
10: **end if**
11: $E = E \cup \{e\}$
12: $edge(e) = (s, o)$
13: $lbl(e) = p$
14: **end for**
15: **return** Pg

Algorithm 2: PGT

Input: A set of RDF Triples T
Output: A Property Graph $Pg = (N, E, edge, lbl, P, \sigma)$

1: $Pg \leftarrow \emptyset$
2: **for** $t \in T$, such that $t =< s, p, o >$ **do**
3: $N = N \cup \{s\}$
4: $lbl(s) =$ {"RDF resource"}
5: **if** o is an RDF resource **then**
6: $N = N \cup \{o\}$
7: $lbl(o) =$ {"RDF resource"}
8: $E = E \cup \{e\}$
9: $edge(e) = (s, o)$
10: $lbl(e) = p$
11: **else**
12: $P = P \cup \{pr\}$
13: $pr =$ {(p, o)}
14: $\sigma(s) =$ pr
15: **end if**
16: **end for**
17: **return** Pg



(a) RPT **(b) PGT**

Figure 3: RPT and PGT transformations for the example in Listing 4

Let us now consider the RDF-star example in Listing 5, which contains an asserted triple for an embedded data property statement – the PGs obtained by applying RPT and PGT are shown in Figure 4. Algorithms 3 and 4 illustrate the main principle of mapping the embedded and asserted triples. RPT conversion for RDF-star is identical to RDF triples, then converting the asserted triple into an edge property (Algorithm 3 lines 5-8). PGT transforms the embedded triple depending on its object; if it is an RDF resource, PGT converts it to an edge. Otherwise, it converts the embedded triple into a node with a property (Algorithm 4 lines 6-11) and fails to transform the asserted triple.

Listing 5: RDF-star triples

```
@prefix ex: <http://example.org/> .
<<ex:Mark ex:age 28>> ex:certainty 1 .
```

In summary, the transformation of the triples from Listing 5 using PGT results in a PG with a single node that makes it impossible to represent the asserted triple since PGs do not support properties over other properties. In contrast, RPT transforms the embedded triple into an edge in the PG and can express the asserted triple as the edge's property. The Abstracting away from a few details (see also Section 6), the Neosemantics approach[10] basically follows PGT while RDF-star Tools[9] and RDF2PG[7] follow RPT.

Algorithm 3: RPT-star

Input: A set of RDF-star Triples T
Output: A Property Graph $Pg = (N, E, edge, lbl, P, \sigma)$

1: $Pg \leftarrow \emptyset$
2: **for** $t \in T$, such that $t =< s, p, o >$ **do**
3: **if** $\bar{t}$ is an embedded triple, such that $t =< \bar{t}, p, o >$ and $\bar{t} =< \bar{s}, \bar{p}, \bar{o} >$ **then**
4: $PgOut =$ RPT($\bar{t}$)
5: $pr =$ {(p, o)}
6: $P = P \cup \{pr\}$
7: $\sigma(e) = pr$
8: **end if**
9: **end for**
10: **return** $Pg \cup PgOut$

Algorithm 4: PGT-star

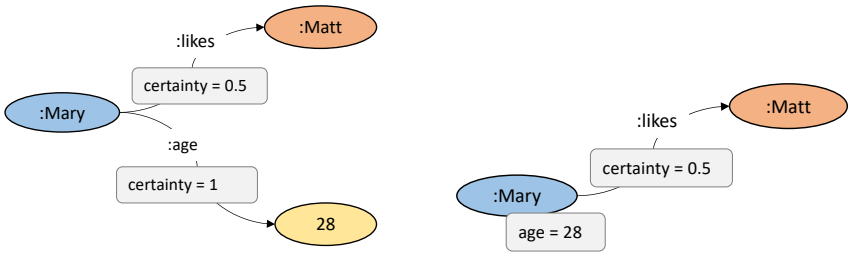
Input: A set of RDF-star Triples T
Output: A Property Graph $Pg = (N, E, edge, lbl, P, \sigma)$

1: $Pg \leftarrow \emptyset$
2: **for** $t \in T$, such that $t =< s, p, o >$ **do**
3: **if** $\bar{t}$ is an embedded triple, such that $t =< \bar{t}, p, o >$ and $\bar{t} =< \bar{s}, \bar{p}, \bar{o} >$ **then**
4: **if** $\bar{o}$ is an RDF resource **then**
5: $PgOut =$ PGT($\bar{t}$)
6: **else**
7: $PgOut =$ PGT($\bar{t}$)
8: $pr =$ {($\bar{p}$,$\bar{o}$)}
9: $P = P \cup \{pr\}$
10: $\sigma(\bar{s}) = pr$
11: **end if**
12: **end if**
13: **end for**
14: **return** $Pg \cup PgOut$

5. Test Cases

In this section, we present a systematic list of test cases that transformation approaches need to fulfill. We distinguish between basic cases that conform to small RDF graphs as well as a range of RDF-star specific test cases that challenge existing approaches – as we will see in our evaluation in Section 6. The complete list of test cases with their short titles is shown in Table 1 – subcases represent variations and bold font indicates cases discussed in more detail in this paper. For the sake of space, we present only part of these cases in this section, more details for all cases are available on our website[11]

[11] https://relweb.cs.aau.dk/rdfstar



(a) RPT **(b) PGT**

Figure 4: RPT and PGT transformations for the example in Listing 5

5.1. Standard RDF

Case 1: Standard RDF statement This case represents an *object property* statement. Both, subject and object are RDF resources. Most transformation approaches map this case to two nodes (subject and object) with an edge (the predicate) connecting them.

```
@prefix ex: <http://example.org/> .
ex:alice ex:meets ex:bob .
```

Case 2: The predicate of an RDF statement is subject in another statement Mapping an RDF statement to two nodes with the predicate as label of the edge between them leads to problems when the predicate itself is also used as a subject in another RDF statement – Case 2.1 therefore consists of the following statements:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex: <http://example.org/> .
ex:Sam ex:mentor ex:Lee .
ex:mentor rdfs:label "project supervisor" .
ex:mentor ex:name "mentor's name" .
```

Other variants of Case 2 include a predicate for a non-literal object, such as `rdf:type` and `rdfs:subPropertyOf`.

Case 3: Data types and language tags It is also important to test the support of different data types and language tags. Hence, Case 3.1, for instance, contains several *datatype property* statements involving different data types and formats for the literal objects:

```
@prefix ex: <http://example.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
ex:book  ex:publish_date "1963-03-22"^^xsd:date .
ex:book  ex:pages        "100"^^xsd:integer .
ex:book  ex:cover        20 .
ex:book  ex:index        "55" .
```

5.2. RDF-star

Case 8: Embedded object property statement in subject position As the name indicates and the following listing shows, this test case features an RDF-star statement where the subject

Table 1
Test cases for evaluating RDFstar-to-PG transformation approaches

Case	Description
	Standard RDF
1	**Standard RDF statement**
2	**The predicate of an RDF statement is subject in another statement**
2.1	Predicate as subject and literal as object
2.2	Predicate as subject and RDF resource as object
2.3	Predicate as subject and RDF property as object - *rdfs:subPropertyOf*
2.4	Predicate as subject and RDF class as object - *rdf:type*
3	**Data types and language tags**
3.1	Datatype property statements with different data types of the literal objects
3.2	Datatype property statements with different language tags of the literal objects
4	**RDF list**
5	**Blank nodes**
6	**Named graphs**
7	**Multiple types for resources - *rdf:type***
	RDF-Star
8	**Embedded *object property* statement in subject position**
9	**Embedded *datatype property* statement in subject position**
10	**Embedded *object property*statement in object position**
11	**Embedded *object property* statement in subject position and non-literal object**
11.1	Asserted statement with non-literal object
11.2	Asserted statement with non-literal object that appears in another asserted statement
12	**Embedded statement in subject position - *object property* with *rdf:type* predicate**
12.1	Asserted statement with *rdf:type* as predicate
12.2	Embedded statement with *rdf:type* as predicate
13	**Double nested RDF-star statement in subject position**
14	**Multi-valued properties**
14.1	RDF statements with same subject and predicate and different objects
14.2	RDF-star statements with the same subject and predicate and different objects
15	**Multiple instances of embedded statements in a single RDF-star graph**
15.1	Identical embedded RDF-star statements with different asserted statements
15.2	RDF statements as embedded and asserted statements in the same graph

corresponds to an embedded object property statement and the object is a literal:

```
@prefix ex: <http://example.org/> .
<<ex:alice ex:likes ex:bob>> ex:certainty 0.5 .
```

Case 9: Embedded datatype property statement in subject position Similar to the previous case we again have an RDF-star statement where the subject corresponds to an embedded statement. In contrast to the Case 8, the embedded statement in Case 9 is a datatype property statement:

```
@prefix ex: <http://example.org/> .
<<ex:Mark ex:age 28>> ex:certainty 1 .
```

Case 10: Embedded object property statement in object position Of course, RDF-star

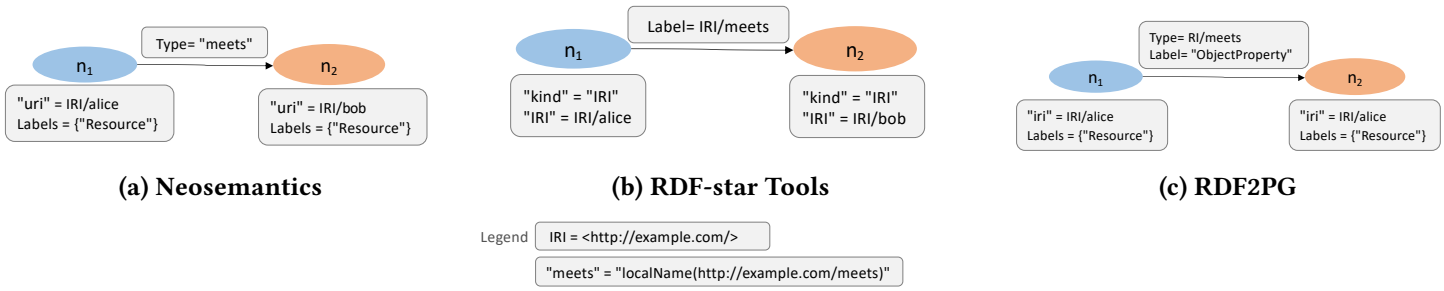
statements can also have embedded statements on object position, which is covered in this case. Similar to Case 8, the embedded statement is an *object property* statement.

```
@prefix ex: <http://example.org/> .
ex:bobhomepage ex:source <<ex:mainPage ex:writer ex:alice>> .
```

Other test cases cover other variations of asserted statements (Case 11), the usage of `rdf:type` in the embedded and asserted statements (Case 12), the double nesting of RDF-star statements (Case 13), the same RDF-star statement with different asserted statements (Case 14), and multiple occurrences of an RDF-star statement within the same graph (Case 15). As mentioned above, details can be found on our project website[11].

6. Analysis and Discussion

In this section, we use the test cases identified in Section 5 to evaluate a number of transformation approaches that we have identified in Section 3: RDF2PG[7], RDF-Star Tools[9], and Neosemantics [10] (Neo4j Community Edition version 4.3.6). The complete results and analysis can be found in the extended arxiv version of this paper [20]. As we will see and as already mentioned in Section 4, RDF-star Tools and RDF2PG follow the RDF-topology Preservation Transformation (RPT) whereas Neosemantics adopts the Property Graph Transformation (PGT).

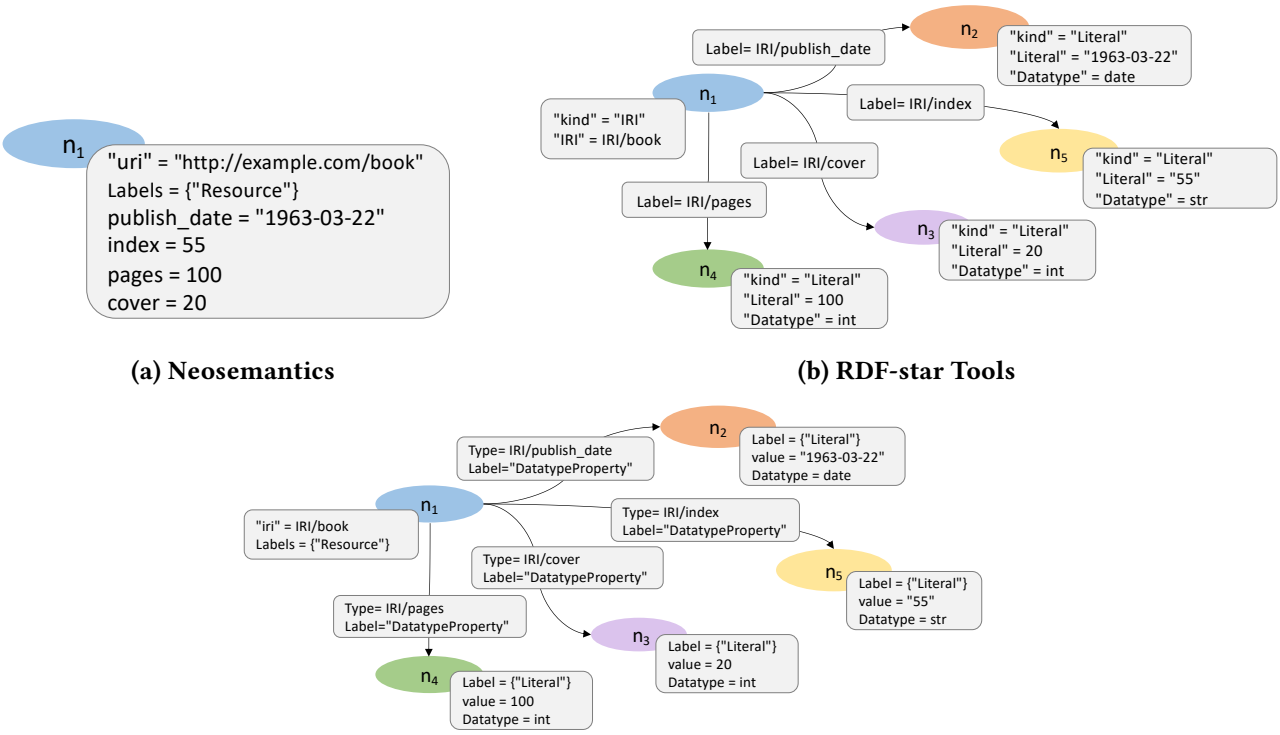


(a) Neosemantics **(b) RDF-star Tools** **(c) RDF2PG**

Figure 5: PGs obtained for Case 1

6.1. Standard RDF

At first, let us discuss our findings for Cases 1 through 7 from Table 1 targeting standard RDF statements. Case 1 corresponds to a simple RDF statement with IRIs as subject and object. Non-surprisingly, all three libraries create a PG with two nodes and one edge (see Figure 5). The main differences are in the way types, labels, and properties are handled. Both Neosemantics and RDF2PG use "Resource" as the label of the two nodes and a key value pair (key=IRI, value=IRI of the subject/object). Additionally, RDF-star Tools uses two additional properties for the nodes: (key="kind", value="IRI") and (key="IRI", value=subject/object IRI). Whereas Neosemantics and RDF2PG use the predicate from the RDF statement as the edge's type, RDF-star Tools uses the predicate as an edge label. RDF2PG additionally uses "ObjectProperty" as an additional edge label.



(a) Neosemantics **(b) RDF-star Tools**

(c) RDF2PG

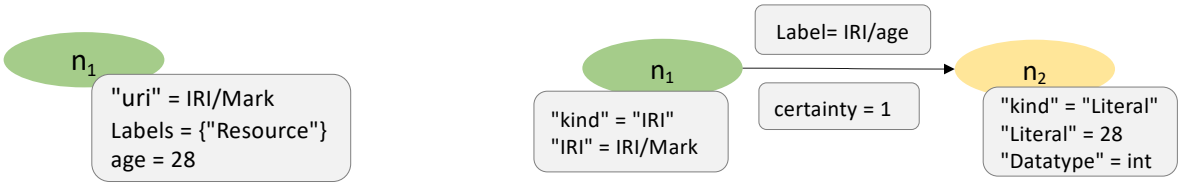
Figure 6: PGs obtained for Case 3

Case 3.1 tests the transformation of *datatype property* statements. In this case, the differences between the libraries are more evident, as depicted in Figure 6. RDF-star Tools and RDF2PG represent each literal using a separate node- The properties of the edge are similar to the ones described for Case 1 with RDF2PG using "DatatypeProperty" as edge label. Neosemantics instead, only creates a single node representing the complete set of input RDF statements; the nodes has one property for each datatype property statement in the input. Additionally, Cases 3.1 and 3.2 also test how transformation approaches support data types and language tags in RDF statements. While Neosemantics ignores them, both RDF-star Tools and RDF2PG define the nodes representing the literal objects with type literal and use the XSD schema data type as annotation.

Cases 4–6 test different RDF features, such as RDF lists (case 4), blank nodes (case 5), and named graphs (case 6). All three projects support RDF lists and blank nodes but only Neosemantics can also import named graphs.

6.2. RDF-star

Let us now discuss over findings for evaluating Cases 8 through 13 from Table 1 targeting diverse RDF-star constructs. We have observed that many of them are not supported by existing libraries. RDF2PG does not support embedded RDF-star statements, so none of these cases could



(a) Neosemantics **(b) RDF-star Tools**

Figure 7: PGs obtained for Case 9

be converted to a PG.

Neosemantics ignores RDF-star statements where: (i) the object of the asserted statement is a literal value (Case 9) or an embedded statement (Case 10), (ii) all elements of an RDF-star statement are RDF resources (Cases 11.1 and 11.2), (iii) the predicate of the embedded or asserted statement is "rdf:type" (Cases 12.1 and 12.2), or (iv) the RDF-star statement is a double embedded statement (Case 13).

RDF-star Tools supports most cases by creating nodes in the PG for literal objects. However, Cases 10 and 13 cause an error in the reading phase of the conversion: we believe that this is an implementation error, and both cases could be supported in the same manner as the others.

Among all cases, Case 9 is particularly interesting and represents a natural extension of Case 3.1. If the embedded statement (mark,age,25) is translated in PGT-style, e.g., by Neosemantics, mark is represented as a node with a property having age as key and 25 as value. It is then not straightforward how to convert the asserted statement (stating that the certainty of (mark,age,25) is 1). Hence, when running Neosemantics, we noticed that it does not transform the asserted statement completely. RDF-star Tools (RPT-style transformation), on the other hand, converts the statement in Case 9 by transforming the embedded statement to an edge with type age between two nodes representing mark and 25; the part of the asserted statement is represented as a key-value property associated to the age edge.

6.3. Edge cases

Case 2 (resource both as a predicate and a subject/object in another statement) is not supported correctly by existing transformation approaches. The result PGs include two separate and independent elements for such resources: one node and one edge property. This is different from RDF, where an IRI identifies the same resource independently from its position in an RDF statement.

Annotating RDF predicates (both datatype and object properties) is a common way to model schema information in RDF. Our finding suggests that there may be more conversion problems when looking at RDF graphs that include RDF Schema and OWL axioms. Existing tools do not have approaches to work at schema level. This investigation is beyond the scope of this paper but we plan to investigate it in our future research.

Other interesting cases are those where the RDF input contains multiple statements with the same subject and predicate but different objects. In Case 14.1, for example, contains two datatype property statements with the same subjects and predicates. Neosemantics converts

the literals ("Info_page","aau_page") into a list of strings[12] used as a value in a property of the node representing the subject of the RDF statement.

6.4. RDF-topology Preservation Transformation (RPT) and Property Graph Transformation (PGT)

Our analysis confirmed our observation that RDF-star Tools follows the RPT approach; the approach creates a node for each distinct subject and object of every RDF statement. Predicates and objects of RDF-star statements are transformed into properties with the key corresponding to the predicate and the value to the object. Neosemantics, in contrast, follows PGT approach for most of the datatype property statements, e.g., Cases 2.1, 3.1, 3.2, 11.2, and 14.1, and supports types and labels (Case 2.4, 5, 6, and 7) slightly differently. Finally, the RDF2PG project adopts the RPT approach in all cases handled.

In summary, the two lines of transformation approaches, RPT and PGT, form the core of start-of-the-art libraries. However, none of the tested libraries supports all test cases. In the end, the particular requirements for converting RDF-star to PGs vary based on the particular use cases and application-specific needs. In one use case, a user might favor one transformation approach over the other based on ontology availability, the application domain, performance, or additional application-specific needs. And driven by real-world applications, users might want to adopt a combination of the basic transformation approaches to best fit their needs.

6.5. Discussion

Ontology and schema availability Application domains, such as financial services, life science, and military, have well-established domain ontologies that are native to the RDF/RDF-star model. Such use cases often also exhibit computationally expensive queries and algorithms that are better for PGs, Hence, in such cases, RPT is preferable to convert the ontology and PGT to convert instance data to reduce the number of nodes and therefore improve runtime.

Another use case involves harvesting Linked Open Data [21, 22] where the PGT approach is usually preferable. Likewise, PGT is easier to use in combination with tabular data. However, in situations, where the ontology is essential and the RDF data is highly complex and heterogeneous, users may prefer RPT. In the end, some graph-based machine learning models also require graphs that model literal property values as nodes.

Performance and query complexity Query performance over graphs generally depends on the number of edges and nodes [23]. Therefore, the user may choose to convert an RDF/RDF-star graph into a PG using a specific transformation method to comply with performance requirements. For example, in an application for a smart home, the authors transformed the RDF graph into a PG based on a custom transformation [24]. The authors evaluated the usage of a PG generated by the custom transformation versus the PG generated by NSMTX[13]. The PG of the custom transformation had two nodes and one edge compared to 16 nodes and 15 edges for the PG generated by the NSMTX plugin. As a result, executing queries over the former graph is

[12] https://neo4j.com/docs/cypher-manual/current/syntax/values/#composite-types
[13] https://neo4j.com/nsmtx-rdf/

more efficient than over the NSMTX graph. The custom transformation eliminated many edges (i.e., relations) and converted them into properties. Since RPT tends to convert each triple into an edge, the transformation generates many nodes compared to PGT that wraps some triples as properties for nodes.

Data sharing RPT allows RDF-star asserted triples (i.e., edge attributes) to be represented as nodes in the PG, treating them as individual entities. This approach can make the graph representation more expressive for other users to understand and re-use than PGT. On the other hand, the PGT can result in many nodes with properties as literal key-value pairs, not explicit edges. This representation can be a natural choice to represent descriptions of nodes.

7. Conclusion

In this paper, we have evaluated and discussed how to transform RDF-star graphs into property graphs. To evaluate existing approaches (Neosemantics, RDF-star Tools, and RDF2PG), we have identified a number of test cases. Our analysis has shown that none of these three approaches supports all test cases and that none of them is the best for all applications. None of them considers user requirements for the transformation process. Nevertheless, existing approaches can roughly be categorized into two lines of transformation approaches: RDF-topology preserving transformation (RPT) and PG transformation (PGT). In our future work, we plan to expand our experiments to comprehensive datasets and combining RPT and PGT into a single hybrid transformation approach. Additionally, we plan to work on the query interoperability between RDF-star and property graphs.

Acknowledgments

This research was partially funded by the Danish Council for Independent Research (DFF) under grant agreement no. DFF-8048-00051B and the Poul Due Jensen Foundation.

References

[1] M. Rodriguez, P. Neubauer, Constructions from Dots and Lines, American Society for Information Science and Technology (2010) 8366.
[2] O. Hartig, Foundations of RDF* and SPARQL*:(An Alternative Approach to Statement-Level Metadata in RDF), in: AMW, 2017.
[3] D. Tomaszuk, RDF Data in Property Graph Model, in: MTSR, 2016, pp. 104–115.
[4] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, A. Taylor, Cypher: An Evolving Query Language for Property Graphs, in: SIGMOD, 2018, pp. 1433–1445.
[5] A. Deutsch, Y. Xu, M. Wu, V. E. Lee, Aggregation Support for Modern Graph Analytics in TigerGraph, in: SIGMOD, 2020, pp. 377–392.
[6] O. Hartig, Foundations to Query Labeled Property Graphs using SPARQL, in: AMAR, 2019.

[7] R. Angles, H. Thakkar, D. Tomaszuk, RDF and Property Graphs Interoperability: Status and Issues, in: AMW, 2019.
[8] O. Lassila, M. Schmidt, B. Bebee, D. Bechberger, W. Broekema, A. Khandelwal, K. Lawrence, R. Sharda, B. Thompson, Graph? yes! which one? help!, preprint arXiv:2110.13348 (2021).
[9] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, et al., DBpedia – A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia, Semantic web (2015) 167–195.
[10] F. M. Suchanek, G. Kasneci, G. Weikum, Yago: A Core of Semantic Knowledge, in: Proc. of TheWebConf, 2007, pp. 697–706.
[11] D. Vrandečić, Wikidata: A New Platform for Collaborative Data Collection, in: TheWebConf, 2012, pp. 1063–1064.
[12] S. Khayatbashi, S. Ferrada, O. Hartig, Converting Property Graphs to RDF: A Preliminary Study of the Practical Impact of Different Mappings (2022).
[13] R. Angles, H. Thakkar, D. Tomaszuk, Mapping RDF Databases to Property Graph Databases, IEEE Access (2020) 86091–86110.
[14] V. Nguyen, O. Bodenreider, A. Sheth, Don't Like RDF Reification? Making Statements about Statements Using Singleton Property, in: TheWebConf, 2014, pp. 759–770.
[15] J. J. Carroll, C. Bizer, P. Hayes, P. Stickler, Named graphs, Journal of Web Semantics (2005) 247–267.
[16] J. Frey, K. Müller, S. Hellmann, E. Rahm, M.-E. Vidal, Evaluation of Metadata Representations in RDF stores, Semantic Web (2019) 205–229.
[17] D. Hernández, L. Galárraga, K. Hose, Computing How-Provenance for SPARQL Queries via Query Rewriting, PVLDB (2021) 3389–3401.
[18] O. Pelgrin, L. Galárraga, K. Hose, Towards Fully-fledged Archiving for RDF Datasets, Semantic Web (2021) 903–925.
[19] J. Bruyat, P.-A. Champin, L. Médini, F. Laforest, PREC: semantic translation of property graphs, arXiv preprint arXiv:2110.12996 (2021).
[20] G. Abuoda, D. Dell'Aglio, A. Keen, K. Hose, Transforming RDF-star to Property Graphs: A Preliminary Analysis of Transformation Approaches - extended version, arXiv preprint arXiv:2210.05781 (2022).
[21] A. Harth, K. Hose, R. Schenkel (Eds.), Linked Data Management, Chapman and Hall/CRC, 2014.
[22] O. Hartig, K. Hose, J. F. Sequeda, Linked data management, in: Encyclopedia of Big Data Technologies, Springer, 2019.
[23] S. Das, J. Srinivasan, M. Perry, E. I. Chong, J. Banerjee, A Tale of Two Graphs: Property Graphs as RDF in Oracle, in: EDBT, 2014, pp. 762–773.
[24] N. Baken, Linked Data for Smart Homes: Comparing RDF and Labeled Property Graphs, in: LDAC2020, 2020, pp. 23–36.