Flexible RML-Based Mapping of Property Graphs to RDF

Ali Elhalawati[1], Anastasia Dimou[1], Olaf Hartig[2] and Daniel Hernández[3]

[1]KU Leuven, Leuven, Belgium
[2]Linköping University, Linköping, Sweden
[3]University of Stuttgart, Stuttgart, Germany

Abstract

RDF graphs and (Labeled) Property Graphs (PGs) have emerged as data models for representing graph databases. Given the differences between the two models, ensuring interoperability between them has become essential, to leverage the strengths of both models. Various approaches have been proposed to map PGs to RDF graphs. However, these approaches differ in terms of structure, representation, size of the generated RDF graph, and degree of configuration provided to the user, making direct comparisons challenging. While declarative methods prevailed to construct RDF graphs from other data formats, the mapping languages proposed for such transformations have not been considered so far for mapping PGs to RDF graphs. In this work, we provide a representation of PG-to-RDF approaches through templates described using RML, a mapping language to construct RDF graphs from heterogeneous data. We show that all considered PG-to-RDF approaches can be represented in RML and, by having a uniform representation of them, we can compare them showcasing their differences. Finally, we show that not only can RML be used to capture PG-to-RDF mappings, but it actually offers more expressive power than the considered PG-to-RDF approaches.

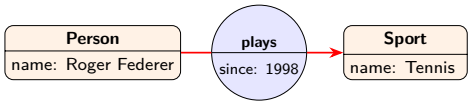

Figure 1: A Representation of a PG with a real-life scenario.

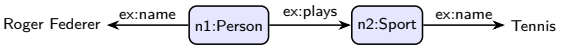

Figure 2: An RDF graph with a real-life scenario.

1. Introduction

Graph databases are powerful representation methods to model data with relationships and dependencies across various fields [1, 2]. In graph databases, nodes normally represent entities, while edges capture their relationships. The most prominent and widely adopted forms of graph databases today are *Property Graphs* (PGs) [2] and *RDF Graphs* [3].

Assuming the reader's familiarity with the concepts and terminologies of PGs and RDF graphs [4], Fig. 1 shows a PG with two nodes, labeled "`Person`" and "`Sport`", an edge with the label "`plays`", node properties as `name:` "`Roger Federer`", and an edge property `since:` "1998". Fig. 2 illustrates an RDF graph with triples having two subjects `n1:Person` and `n2:Sport`, Predicates as `ex:plays` and `ex:name`, and objects as `n2:Sport` and "`Tennis`".

RDF-star [5] is an extension of RDF where the subject and the object can be triples, i.e. an RDF triple can be treated as a node within the graph. For example, the triple `n1:Person ex:plays n2:Sport` in Fig. 2 can be the subject to the triple `«n1:Person ex:plays n2:Sport» ex:since` "1998".

PGs and RDF graphs have representational and structural differences [6]. RDF graphs do not have edges, properties, or labels, and are not multigraphs as PGs. On the other hand, RDF graphs rely on IRIs to have global unique identifiers for RDF components, unlike PGs that rely on IDs to uniquely identify PG components only inside the scope of the PG.

Since PGs and RDF graphs differ, the choice between them depends on the use case. However, to leverage the strengths of both, several approaches have been proposed to enable their interoperability. Several works exist in the literature on RDF-to-PG mapping [7, 8, 9], as well as PG-to-RDF mapping [10, 11, 12, 13, 14, 15, 16]. Some of the PG-to-RDF approaches provide *direct mappings* to convert PGs to RDF automatically without user intervention [10, 11, 12], while other approaches offer configuration on the RDF terms in the output RDF graph [13], or even generate an RDF graph for a specific PG sub-graph [14, 15].

Despite the existence of several PG-to-RDF mapping approaches, it is hard to compare them directly, since each approach differs in terms of the structure, representation, size of the generated RDF graph, the degree of configuration provided to the user, and assumes different data formats for the input PG. In addition, these approaches impose restrictions on the generated RDF terms in their mappings. For example, all the PG-to-RDF approaches do not allow combining more than one PG component in one RDF term in the generated RDF graph, and restrict property values in a PG to be literals in the generated RDF graph.

Several declarative mapping languages were proposed to construct RDF graphs from heterogeneous data sources [17]. However, these mapping languages are not considered so far for mapping PGs to RDF graphs. The RDF Mapping Language (RML) is a declarative mapping language used to generate RDF graphs from heterogeneous data sources [18, 19], by extending the W3C recommended R2RML [20] to heterogeneous data sources. RML has been extended to *RML-star* [21], enabling declarative mappings to

Published in the Proceedings of the Workshops of the EDBT/ICDT 2025 Joint Conference (March 25-28, 2025), Barcelona, Spain
ali.elhalawati@kuleuven.be (A. Elhalawati); anastasia.dimou@kuleuven.be (A. Dimou); olaf.hartig@liu.se (O. Hartig); daniel.hernandez@ki.uni-stuttgart.de (D. Hernández)
0000-0003-1457-0031 (A. Elhalawati); 0000-0003-2138-7972 (A. Dimou); 0000-0002-1741-2090 (O. Hartig); 0000-0002-7896-0875 (D. Hernández)

generate RDF-star graphs.

In this work, we provide a uniform method for comparing the existing PG-to-RDF approaches using RML and show that RML can map PGs to RDF graphs. We accomplish this by representing each considered PG-to-RDF mapping approach as a template of RML mappings and providing a uniform input format for the PG across all considered approaches. These RML templates require the user to define only the property keys and values to be mapped from the PG. This uniform representation of the most prominent PG-to-RDF mapping approaches enables direct comparisons, highlighting their differences. In addition, we highlight the limitations of the considered PG-to-RDF approaches in generating RDF graphs and showcase that not only RML supports mapping PG-to-RDF, but it also offers more expressive power and configuration than the considered PG-to-RDF approaches.

2. Preliminaries

In this section, we revisit the formal definition of PGs and briefly summarize RML. We refrain from formally defining RDF graphs as it is unnecessary for this work and instead, refer to the RDF specification for details [3]. We adopt the formal definition of PGs from [6, 12, 4], as follows.

Definition 1. *A PG is the tuple* $(N, E, \rho, \lambda, \sigma)$ *such that* N *is a finite set of nodes and* E *is a finite set of edges where* $N \cap E = \emptyset$. $\rho : E \to (N \times N)$ *is the edge construction total function.* $\lambda : (N \cup E) \to L$ *is a total function that maps nodes and edges to labels in* L. $\sigma : (N \cup E) \times P \to \mathcal{VA}$ *is a partial function that takes a node/edge and a property from* P *and assigns them to a property value from* $\mathcal{VA}$.

Example 1. *The formal definition of PGs captures the PG in Fig. 1 as follows:*

$$N = \{n1, n2\},\ \lambda(n1) = \text{“}\texttt{Person}\text{”},\ \lambda(n2) = \text{“}\texttt{Sport}\text{”},$$
$$E = \{e1\},\ \lambda(e1) = \text{“}\texttt{plays}\text{”},\ \rho(e1) = (n1, n2),$$
$$\sigma(n1, name) = \text{“}\texttt{Roger Federer}\text{”},$$
$$\sigma(n2, name) = \text{“}\texttt{Tennis}\text{”},\ \sigma(e1, since) = \text{“1998”}$$

An *RML Mapping Document* $\mathcal{M}$ describes how to generate RDF graphs from input data sources. $\mathcal{M}$ is formed from a set of *Triples Maps* that define how to generate triples in an RDF graph. A triples map has zero or one *Logical Source*, one *Subject Map*, zero or more *Predicate-Object Maps*, and zero or more *Graph Maps*. The logical source specifies the input data source. The subject map defines how to create subjects in triples. A predicate-object map consists of one or more *Predicate Map* that defines the rule generating the predicates in RDF triples. Additionally, a predicate map is accompanied by one or more *Object Map* that defines how to generate objects in triples. Graph maps can be specified in subject maps and object maps to group the generated triples in named graphs.

To support the extension of RDF with RDF-star, Delva et al. introduce an extension of RML, *RML-Star* [21], which enables the generation of RDF-star triples. An RML-Star $\mathcal{M}$ may include *Triples Maps*, *Asserted Triples Maps*, and *Non-Asserted Triples Maps*. The RDF triples constructed by the asserted triples maps are also generated in the output graph, unlike the non-asserted triples maps where the constructed RDF triples are only referenced inside RDF-star triples. Asserted and non-asserted triples maps contain *Star Maps* for subject and object maps. Each Star Map has a *Quoted Triples Map* referencing another asserted or non-asserted triples map.

3. RML Templates for PG-to-RDF

Several works map PGs to RDF graphs, we review these works and provide their corresponding RML templated mappings necessary for generating RDF graphs on a running example. These templated mappings only require the user to specify the property keys and values to be mapped from their PG. In our work, Neo4j serves as the database system for managing PGs. Neo4j enables users to query the stored PGs using the Cypher query language. The query results are then exported to a JSON using the *APOC* [22] plugin in Neo4j. We present the templated RML mappings in a YARRRML serialization [23]. The full YARRRML and RML templates of each considered approach are available in a dedicated GitHub repo (https://github.com/dtai-kg/PG-to-RDF-RML-Templates).

3.1. The Oracle Approaches

Das et al. [10] introduced three PG-to-RDF direct mappings for Oracle databases, which differ in the mapping of edge properties: reification-based (RF), subproperty-based (SP), and named graph-based (NG). None of them covers node labels. Predefined transformation functions **IRI** and **Literal**, are used to transfer PG elements to IRIs and literals in the output RDF graph.

RF Mapping Let $PG = (N, E, \rho, \lambda, \sigma)$ be a PG, an RDF graph is generated from PG as follows:

1. For every $e \in E$ with $\rho(e) = (n1, n2)$ and $\lambda(e) = l$, triples are generated as follows:

   IRI(e) `rdf:subject` **IRI**($n1$).
   IRI(e) `rdf:predicate` **IRI**(l).
   IRI(e) `rdf:object` **IRI**($n2$).
   IRI($n1$) **IRI**(l) **IRI**($n2$).

2. For every $x \in N \cup E$, where x is an edge or a node with $\sigma(x, p) = v$ where $p \in P$ and $v \in \mathcal{VA}$, a triple is generated following the construction **IRI**(x) **IRI**(p) **Literal**(v).

Example 2. *Applying the RF mapping to the PG in Fig. 1 yields an RDF graph with the following triples:*

```
pg:e1 rdf:subject pg:n1.
pg:e1 rdf:predicate pg:r/plays.
pg:e1 rdf:object pg:n2.
pg:n1 pg:r/plays pg:n2.
pg:n1 pg:k/name "Roger Federer".
pg:n2 pg:k/name "Tennis".
pg:e1 pg:k/since "1998".
```

For the RF mapping, the template RML mappings to generate the RDF graph in Ex. 2 are the following:

Listing 1: The RF Template Mappings

```
TM1:
 s: pg:e$(id)
 po:
   - [rdf:predicate, pg:r/$(label)]
   - [rdf:subject, pg:n$(start.id)]
   - [rdf:object, pg:n$(end.id)]
   - [pg:k/since, $(properties.since)]
TM2:
 s: pg:n$(start.id)
 po:
   - [pg:r/$(label), pg:n$(end.id)]
   - [pg:k/name, $(start.properties.name)]
TM3:
 s: pg:n$(end.id)
 po:
   - [pg:k/name, $(end.properties.name)]
```

TM1 generates the triples where the edge is the subject, while TM2 and TM3 generate the triples where the source node and the target node are subjects.

SP Mapping For $PG = (N, E, \rho, \lambda, \sigma)$, the corresponding RDF graph is generated such that for every $e \in E$ where $\rho(e) = (n1, n2)$ and $\lambda(e) = l$ triples are generated as follows:

IRI(e) `rdfs:subPropertyOf` **IRI**(l).
IRI($n1$) **IRI**(l) **IRI**($n2$).
IRI($n1$) **IRI**(e) **IRI**($n2$).

For $x \in N \cup E$ with $\sigma(x, p) = v$, the triples are generated following step 2 in RF mapping.

Example 3. *On the PG of Fig. 1, SP maps edges and their labels as follows:*

```
pg:e1 rdfs:subPropertyOf pg:r/plays.
pg:n1 pg:e1 pg:n2.
```

SP overlaps with the RF in the generated RDF graph for the nodes and properties. However, they differ in the RDF graph for the edges and their labels by relying on the subPropertyOf relationship to express the edges instead of reification. This requires slight modifications to the template in Listing 1, the modified template is available in our dedicated GitHub repo.

NG Mapping For $PG = (N, E, \rho, \lambda, \sigma)$, an RDF graph is generated following the NG mapping as follows: (i) For every $e \in E$ where $\rho(e) = (n1, n2)$ and $\lambda(e) = l$ a triple is generated and grouped in a named graph following the construction **IRI**($n1$) **IRI**(l) **IRI**($n2$) **IRI**(e), (ii) for every $\sigma(e, p) = v$, we group the edge properties in named graphs following the construction **IRI**(e) **IRI**(p) **Literal**(v) **IRI**(e), (iii) for every $n \in N$, the triples are generated as step 2 in RF mapping.

Example 4. *On the PG of Fig. 1, the NG mapping groups edges, labels, and properties in named graphs:*

```
pg:n1 pg:r/plays pg:n2 pg:e1.
pg:e1 pg:k/since "1998" pg:e1.
```

NG mapping is similar to RF and SP in generating RDF graphs for PG nodes and node properties. However, it differs in generating RDF graphs for edges, edge properties, and edge labels. Slight modifications are required for the template in Listing 1, where edge relations, properties, and labels are placed in named graphs categorized by the edges.

3.2. Singleton Property Graph Approach

Nguyen et al. [11] propose a direct PG-to-RDF mapping through an intermediate unified graph model called *Singleton Property Graph* (SPG). SPGs capture the structures of both PGs and RDF graphs. To transform PGs into SPGs, each node and edge in the PG is mapped to a node in the SPG. Furthermore, two edges are introduced in the SPG: one connects the edge's created node to the source node, and the other links the edge's node to the destination node. The same non-configurable transformation functions **IRI** and **Literal** defined in the Oracle approaches are assumed. Also, three unique IRIs $\mathbf{IRI}_{in}, \mathbf{IRI}_{out}, \mathbf{IRI}_{label}$ are used to describe the edge relation and node/edge labels in the output RDF graph. The PG is mapped to an SPG, which is mapped to an RDF graph as follows: Let $PG = (N, E, \rho, \lambda, \sigma)$ be a PG, for every $n \in N$ and $\lambda(n) = ln$ a triple is generated following the construction **IRI**(n) $\mathbf{IRI}_{label}$ **Literal**(ln). For every $e \in E$ with $\rho(e) = (n1, n2)$ and $\lambda(e) = le$ triples are generated as follows:

IRI($n1$) $\mathbf{IRI}_{in}$ **IRI**(e).
IRI(e) $\mathbf{IRI}_{out}$ **IRI**($n2$).
IRI(e) $\mathbf{IRI}_{label}$ **Literal**(le).

For $x \in N \cup E$, the triples are generated as step 2 in RF mapping.

Example 5. *Applying the SPG mapping to the PG in Fig. 1 yields an RDF graph as follows:*

```
ex:n1 rdfs:label "Person".
ex:n1 ex:name "Roger Federer".
ex:n2 rdfs:label "Sport".
ex:n2 ex:name "Tennis".
ex:e1 rdfs:label "plays".
ex:e1 ex:since "1998".
ex:n1 ex:nodeToedge ex:e1.
ex:e1 ex:EdgeTonode ex:e2.
```

SPG does not differentiate between edge and node labels. To address this, a variation of the SPG mapping was proposed [6], where the unique IRI $\mathbf{IRI}_{label}$ is split into two distinct IRIs $\mathbf{IRI}_{elabel}$ and $\mathbf{IRI}_{nlabel}$ for edge and node labels respectively.

We showcase the template RML mappings by generating the RDF graph in Ex. 5 as follows:

Listing 2: The SPG Template Mappings

```
TM1:
  s: ex:n$(id)
  po:
    - [rdfs:label, $(labels)]
    - [ex:name, $(properties.name)]
TM2:
  s: ex:n$(start.id)
  po:
    - [ex:nodeToedge, ex:e$(id)]
TM3:
  s: ex:e$(id)
  po:
    - [rdfs:label, $(label)]
    - [ex:edgeTonode, ex:n$(end.id)]
    - [ex:since, $(properties.since)]
```

TM1 generates RDF triples that describe the properties of nodes in the graph. TM2 generates triples that link the source node to the edge. TM3 generates triples that describe the edge's properties and label and connect the edge to the target node.

3.3. RDF-Star-Based Approach

Hartig [13] proposes an RDF-star-based mapping approach that converts PGs to RDF-star graphs by describing edge properties using RDF-star triples, where the triple connecting the edge label to the source node and the target node is used as the subject in these RDF-star triples. This approach enables users to choose patterns for generating IRIs that denote edge labels and property keys, and the desired RDF term in the resulting RDF-star graph for certain PG components. The functions **IRI**, **Literal**, ϕ and γ take a PG component C and transfer it to an RDF term such that **IRI**(C) and **Literal**(C) generate an IRI and a literal. $\phi(C)$ generates either a literal or an IRI, and $\gamma(C)$ generates a blank node or an IRI. The functions **IRI**, ϕ, and γ are *configurable*, i.e. the user can define the IRIs to be used, as well as the desired RDF term in ϕ and γ. In addition, an IRI $\mathbf{IRI}_{label}$ is used to describe the presence of a node label.

Let $PG = (N, E, \rho, \lambda, \sigma)$ be a PG, for every $n \in N$ where $\lambda(n) = ln$ a triple is generated following the construction $\gamma(n)$ $\mathbf{IRI}_{label}$ $\phi(ln)$. For every $n \in N$ with $\sigma(n, p) = v$, a triple is generated following the construction $\gamma(n)$ **IRI**(p) **literal**(v). For every $e \in E$ where $\rho(e) = (n1, n2)$ and $\lambda(e) = le$, a triple is generated following the construction $\gamma(n1)$ **IRI**(le) $\gamma(n2)$. For every $e \in E$ where $\rho(e) = (n1, n2)$, $\lambda(e) = le$ and $\sigma(e, p') = v'$, the triple describing the edge is reused as a subject to generate the RDF-star triple «$\gamma(n1)$ **IRI**(le) $\gamma(n2)$» **IRI**(p') **literal**(v').

Example 6. *We show how to generate the RDF-star triples for edge properties on the PG in Fig. 1:*

```
ex:n1 ex:plays ex:n2.
<<ex:n1 ex:plays ex:n2>> ex:since "1998".
```

In this approach, the edge triple uses the edge label as a predicate, raising the question of what happens if edges have no label. This approach overlaps with the SPG approach in producing node labels and properties. The difference lies in expressing the edge relations, labels, and properties using RDF-star. The mappings in Listing 2 are modified with RML-star mappings to generate the desired triples. We provide this modified template in our GitHub repo.

3.4. PGO Approach

Tomaszuk et al. proposed the *Property Graph Ontology* (PGO), an OWL ontology designed for describing PGs in RDF [12]. Users are restricted to using the predefined ontology terms provided by PGO and can only integrate these terms with a limited set of other ontologies specified in [12]. The same non-configurable transformation functions **IRI** and **Literal** described in the oracle approach are used to convert a PG component to an IRI or literal.

Let $PG = (N, E, \rho, \lambda, \sigma)$ be a PG, each node/edge property mapping $\sigma(x, p) = v$ is assumed to have a unique identifier $\sigma_{x,p,v}$. For every $n \in N$, a triple is generated following the construction **IRI**(n) `rdf:type pgo:Node`. For each $n \in N$ with $\sigma(n, p) = v$, a triple is generated following the construction **IRI**(n) `pgo:hasNodeProperty` **IRI**$(\sigma_{n,p,v})$. For every $e \in E$ where $\rho(e) = (n1, n2)$, triples are generated as follows:

IRI(e) `rdf:type pgo:Edge.`
IRI(e) `pgo:startNode` **IRI**$(n1)$`.`
IRI(e) `pgo:endtNode` **IRI**$(n2)$`.`

For each $e \in E$ with $\sigma(e, p) = v$ a triple is generated following the construction **IRI**(e) `pgo:hasEdgeProperty` **IRI**$(\sigma_{e,p,v})$. For $x \in N \cup E$ such that $\sigma(x, p) = v$ the following triples are generated:

IRI$(\sigma_{x,p,v})$ `rdf:type pgo:Property.`
IRI$(\sigma_{x,p,v})$ `pgo:key` **Literal**(p)`.`
IRI$(\sigma_{x,p,v})$ `pgo:value` **Literal**(v)`.`

For $x \in N \cup E$ where $\lambda(x) = l$, the triple **IRI**(x) `pgo:label` **Literal**(l) is generated.

Example 7. *We show a sample of the PGO mapping on the PG in Fig. 1:*

```
ex:n1 rdf:type pgo:Node.
ex:n1 pgo:label "Person".
ex:n1 pgo:hasNodeProperty ex:p1.
ex:n2 rdf:type pgo:Node.
ex:p1 rdf:type  pgo:Property.
ex:p1 pgo:key "name".
ex:p1 pgo:value "Roger Federer".
ex:e1 rdf:type pgo:Edge.
ex:e1 pgo:startNode ex:n1.
ex:e1 pgo:endNode ex:n2.
ex:e1 pgo:label "plays".
ex:e1 pgo:hasEdgeProperty ex:p2.
ex:p2 rdf:type  pgo:Property.
ex:p2 pgo:key "since".
ex:p2 pgo:value "1998".
```

Besides relying on specific ontology terms, the PGO translation generates a large RDF graph even for a small PG, as repeated triples are generated for PG components to comply with the ontology. To adapt to the assumption of having unique property IDs, we attach the property value to the node/edge and use it as a unique ID for each property. In addition, to correctly construct triples specialized to nodes/edges, we use functions in RML to check the type of the PG component. The template RML mappings to generate the RDF graph in Ex. 7 are as follows.

Listing 3: The PGO Template Mappings

```
TM1:                                                    1
  s: ex:n$(id)                                          2
  condition:                                            3
   function: idlab-fn:equal                             4
   parameters:                                          5
    - [grel:valueParameter, $(type), s]                 6
    - [grel:valueParameter2, "node", o]                 7
  po:                                                   8
  - [a, pgo:Node]                                       9
  - [pgo:label, $(labels)]                              10
  - [pgo:hasNodeProperty,                               11
       ex:p_$(properties.name)_$(id)]                   12
TM2:                                                    13
  s: ex:e$(id)                                          14
  condition:                                            15
   function: idlab-fn:equal                             16
   parameters:                                          17
    - [grel:valueParameter, $(type), s]                 18
    - [grel:valueParameter2,                            19
                  "relationship", o]                    20
  po:                                                   21
  - [a, pgo:Edge]                                       22
  - [pgo:hasEdgeProperty,                               23
      ex:p_$(properties.since)_$(id)]                   24
TM3:                                                    25
  s: ex:p_$(properties.name)_$(id)                      26
```

```
 po:                                          27
 - [a, pgo:property]                          28
 - [pgo:key, "name"]                          29
 - [pgo:value, $(properties.name)]            30
TM4:                                          31
 s: ex:p_$(properties.since)_$(id)            32
 po:                                          33
 - [a, pgo:property]                          34
 - [pgo:key, "since"]                         35
 - [pgo:value, $(properties.since)]           36
TM5:                                          37
   s: ex:e$(id)                               38
   po:                                        39
   - [pgo:startNode, ex:n$(start.id)]         40
   - [pgo:endNode, ex:n$(end.id)]             41
   - [pgo:label, $(label)]                    42
```

TM1 generates the triples that identify the node, attach the node to its label, and create the unique properties IDs for each node. TM2 does the same as TM1 but for edges. TM3 and 4 construct the triples to describe the node/edge properties. TM5 generates the triples that describe the edge label, and connect the edge to its source and target nodes.

3.5. PRSC and PREC Approaches

Bruyat [24] introduces two user-defined mappings that convert PGs to RDF PRSC [15] and PREC [14].

PRSC [15] is defined as mapping rules written in RDF-star [25]. PRSC mapping rules are divided into target and production parts. The target specifies the intended component in the PG based on its type (node or edge), label, and property names. The labels and property names of the target node/edge must be specified since omitting them implies that they do not exist in the original PG. In the production part of the rule, the user specifies the desired output in the RDF-star graph. Let $PG = (N, E, \rho, \lambda, \sigma)$ be a PG, PRSC assumes the following variables:

- `pvar:self` represents $n \in N$ or $e \in E$ in the form of a blank node.
- `pvar:source` represents the source node $n1$ of any edge e with $\rho(e) = (n1, n2)$ as a blank node.
- `pvar:destination` represents the target node $n2$ for any edge e with $\rho(e) = (n1, n2)$ in the form of a blank node.
- `prec:valueOf(p)` is a literal representing the value v for $\sigma(x, p) = v$ where $x \in N \cup E$, p has to be specified.

In the production part of every rule, users can combine these variables with any desired constant IRI to define the intended output in the RDF-star graph.

Example 8. *Applying this PRSC rule on the PG in Fig. 1 selects edges with a specific label and property:*

```
_:playsRule a prec:PRSCEdgeRule;
prec:label "plays";
prec:propertyKey "since";
prec:produces
<<<<pvar:source ex:plays pvar:destination>>>>
            ex:since "since"^^prec:valueOf.
```

As a result of this rule, the RDF-star triple `<<_:n1 ex:plays _:n2>> ex:since "1998"` *is generated.*

It is observed that, in any PRSC rule, if a user intends to target a specific node or edge x, they are required to specify all the property names and labels of x, even if these are not intended to be included in the production part of the rule. This requirement can be tedious, particularly for property graphs with numerous properties and labels.

PREC [14] follows the same structure as PRSC but introduces some differences. In the target part of PREC rules, the target can be a property instead of only a node/edge. Moreover, unlike PRSC, omitting certain property names of the target node or edge does not imply their non-existence. PREC is deemed in [24] as a low-level mapping that is "not user-friendly" and "difficult" as it requires the user to be very familiar with the language and learn an extensive vocabulary to utilize it effectively. Another limitation of PREC is that it sometimes produces unexpected triples specifically in cases where different nodes/edges share the same property names. In addition, PREC cannot deal with nodes/edges having empty labels.

Despite the high degree of configuration, PREC and PRSC share some limitations. Both systems convert the entire PG to an RDF graph following the PGO ontology [12], this proved some scalability issues as mentioned in [15]. Both systems enforce certain RDF term types for each PG component, e.g., nodes/edges are always blanks and property values are always literals. Property values are enforced to be used only as objects in the output RDF graph, and it is impossible to combine different PG components in one RDF term.

PRSC Solution Let m_{PRSC} be a set of mappings in PRSC format, every mapping rule r in m_{PRSC} has a target **t** and a production **prod**. **t** has a type node or edge, a possible-empty label l, and a possible-empty set of property names P. **prod** represents an RDF-star graph where every triple in **prod** can contain user-defined constant IRIs, another triple, or a PRSC-defined variable dv. For every r in m_{PRSC}, if **t** has a type node, then for any $dv \in$ **prod**, $dv \in \{pvar{:}self,\ prec{:}valueOf(p)\}$ where p is a property name in the PG. Otherwise, if **t** has a type edge, then $dv \in \{pvar{:}self, pvar{:}source, pvar{:}destination, prec{:}valueOf(p)\}$.

For every r in m_{PRSC}, let $\mathcal{C}(r) \rightarrow cq$ be a function that constructs a cypher query cq for t in r. We show how to construct cq based on the content of t as follows:

- if t has type node, a non-empty label l, and $P = \{p_1, p_2, \ldots p_j\}$ where $j \geq 0$:

  ```
  MATCH (n:l) WHERE n.p1 IS NOT NULL AND
      n.p2 IS NOT NULL ... AND n.pj IS NOT
       NULL AND size(keys(n)) = j RETURN
      (n)
  ```

- if t has type node, an empty label, and $P = \{p_1, p_2, \ldots p_j\}$ where $j \geq 0$:

  ```
  MATCH (n) WHERE size(labels(n)) = 0 AND
       n.p1 IS NOT NULL AND n.p2 IS NOT
      NULL ... AND n.pj IS NOT NULL AND
      size(keys(n)) = j RETURN (n)
  ```

- if t has type edge, a non-empty label l, and $P = \{p_1, p_2, \ldots p_j\}$ where $j \geq 0$:

  ```
  MATCH (n)-[r:l]->(q) WHERE n.p1 IS NOT
      NULL AND n.p2 IS NOT NULL ... AND n.
      pj IS NOT NULL AND size(keys(r)) =
      j RETURN (r)
  ```

PG comp.	RF	SP	NG	SPG	star	PGO
nodes	0	0	0	1	1	1
node labels	0	0	0	1	1	1
edges	4	3	1 q	3	1	3
edge label	1	1	1 q	1	1	1
node prop.	1	1	1	1	1	4
edge prop.	1	1	1 q	1	1*	4

Table 1
Minimum no. of triples per PG component

The $size(keys())$ condition ensures only a specific number of properties as PRSC operates. After constructing cq, cq can be exported to a JSON $JSON_{cq}$ with any PG database system, where $JSON_{cq}$ contains all the necessary components to generate the corresponding RDF-star graph. $JSON_{cq}$ is then provided to RML mappings as an input data source and combined with the user-defined constant IRIs in **prod** to generate the RDF-star graph intended by r.

3.6. Discussion

We compare the considered PG-to-RDF approaches in terms of the no. of triples per PG component (Table 1). We notice that PGO generates the most triples for properties, while Oracle generates the most triples for edges. RDF-star constructs RDF-star triples (*) when mapping edge properties, while NG generates named graphs (q) when mapping edges, edge labels, and edge properties. PRSC is not mentioned in Table 1 as it allows configuring the no. of triples.

We observe that all the approaches considered have limitations. All the approaches limit every PG component to certain RDF term types, e.g., all the approaches restrict the property values to be literal in the generated RDF graph. Most approaches lack configuration. Only the PRSC approach allows configuration in the content and size of the generated RDF/RDF-star graphs, but this does not include properties. The RDF-star approach also allows configuration in the generated RDF graph concerning the IRI patterns and the RDF terms for certain PG components. None of the approaches supports using more than one PG component within a single IRI, e.g., it is not possible to generate an RDF term containing an IRI with both the node and its label from a PG, which means that each PG component must be mapped to a single RDF term. None of the approaches supports combining the PG with other data sources for RDF construction if the user is interested in joining data across different data sources. None of the approaches supports having triples in the object of an RDF-star triple.

3.7. RML as a Flexible Solution

Besides covering all the PG-to-RDF translations considered, RML overcomes all their limitations by providing full control over the RDF terms, the desired output (RDF or RDF star), and the ability to combine multiple components from the input data. We showcase in Listing 4 RML-star mappings that generate a custom RDF graph for portions of the PG in Fig. 1. This RDF graph is impossible to generate with any of the considered PG-to-RDF approaches and shows the great mapping flexibility of RML. The full version of the Listing is provided in the dedicated GitHub repo.

Listing 4: Highly Flexible RML Mappings

```
TM1:
  s: ex:$(labels)/$(properties.name)
  po:
   - [ex:nodeID, $(id)]
TM2:
  s: ex:$(start.properties.name)
  po:
   - [ex:$(label),
        $(end.properties.name)]
TM3:
  s: ex:since
  po:
   - p: ex:$(properties.since)
     o:
      - quoted: TM2
```

Applying the RML mappings in Listing 4 on the PG in Fig. 1 results in the following RDF-star graph:

```
<ex:Person/Roger%20Federer> ex:nodeID  “1”.
<ex:Sport/Tennis> ex:nodeID  “2”.
<ex:Roger%20Federer> ex:plays “Tennis”.
ex:since ex:1998 <<ex:Roger%20Federer ex:
    plays “Tennis”>>.
```

This RDF graph generated from Listing 4 shows that RML can flexibly map PG components to RDF. TM1 generates the first and second triples, where the object is the node, and the subject is an IRI combining the property value and label. Such triples are not achievable by any of the considered PG-to-RDF approaches, as they force the property value to be an object literal. They also restrict PG nodes to be IRIs or blank nodes and do not allow mapping multiple PG components as a single IRI. TM2 generates the third triple, which is used as an object in TM3 to generate the fourth RDF-star triple. None of the considered approaches allows using triples in the object of an RDF-star triple.

4. Conclusions

In this work, we presented RML as a uniform and flexible solution for mapping PGs to RDF. We studied the most prominent PG-to-RDF translations and provided templated RML mappings that show RML's capability in performing these translations. For future work, we plan to conduct a thorough performance comparison between the considered PG-to-RDF approaches in generating RDF graphs with RML.

Acknowledgments

Hartig's contributions to this work were funded by Vetenskapsrådet (the Swedish Research Council, project reg. no. 2019-05655). Dimou and Elhalawati's contributions to this research were partially supported by Flanders Make, the strategic research centre for the manufacturing industry, and the Flemish Government under the “Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen” program. Hernández's contributions to the work were funded by the German Research Foundation, DFG (GA SFB-1574-471687386).

References

[1] Tian, Yuanyuan, The World of Graph Databases from An Industry Perspective, SIGMOD Rec. 51 (2022). doi:10.1145/3582302.3582320.

[2] Robinson, Ian and Webber, Jim and Eifrem, Emil, Graph Databases: New Opportunities for Connected Data, 2nd ed., O'Reilly Media, Inc., 2015.

[3] R. Cyganiak, D. Wood, M. Lanthaler, RDF 1.1 Concepts and Abstract Syntax, Recommendation, World Wide Web Consortium (W3C), 2014. URL: http://www.w3.org/TR/rdf11-concepts/.

[4] Angles, Renzo and Arenas, Marcelo and Barceló, Pablo and Hogan, Aidan and Reutter, Juan and Vrgoč, Domagoj, Foundations of Modern Query Languages for Graph Databases, ACM Comput. Surv. 50 (2017). doi:10.1145/3104031.

[5] Hartig, Olaf and Champin, Pierre-Antoine and Kellogg, Gregg and Seaborne, Andy, RDF-star and SPARQL-star, W3C Final Community Group Report, 2021. URL: https://w3c.github.io/rdf-star/cg-spec/2021-12-17.html.

[6] Khayatbashi, Shahrzad and Ferrada, Sebastián and Hartig, Olaf, Converting property graphs to RDF: a preliminary study of the practical impact of different mappings, in: Proceedings of the 5th ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA), ACM, 2022. doi:10.1145/3534540.3534695.

[7] R. Angles, H. Thakkar, D. Tomaszuk, Mapping RDF Databases to Property Graph Databases, IEEE Access 8 (2020). doi:10.1109/ACCESS.2020.2993117.

[8] D. Tomaszuk, RDF Data in Property Graph Model, in: Metadata and Semantics Research, Springer International Publishing, 2016. doi:10.1007/978-3-319-49157-8_9.

[9] G. Abuoda, D. Dell'Aglio, A. Keen, K. Hose, Transforming rdf-star to property graphs: A preliminary analysis of transformation approaches, in: Proceedings of the QuWeDa 2022: 6th Workshop on Storing, Querying and Benchmarking Knowledge Graphs co-located with ISWC, CEUR-WS.org, 2022. URL: https://ceur-ws.org/Vol-3279/paper2.pdf.

[10] Das, Souripriya and Srinivasan, Jagannathan and Perry, Matthew and Chong, Eugene Inseok and Banerjee, Jayanta, A Tale of Two Graphs: Property Graphs as RDF in Oracle., in: Proceedings of the 17th International Conference on Extending Database Technology, EDBT, OpenProceedings.org, 2014. doi:10.5441/002/edbt.2014.82.

[11] Nguyen, Vinh and Yip, Hong Yung and Thakkar, Harsh and Li, Qingliang and Bolton, Evan and Bodenreider, Olivier, Singleton Property Graph: Adding A Semantic Web Abstraction Layer to Graph Databases., in: Proceedings of the Blockchain enabled Semantic Web Workshop (BlockSW) and Contextualized Knowledge Graphs (CKG) Workshop co-located with ISWC, CEUR-WS.org, 2019. URL: https://ceur-ws.org/Vol-2599/CKG2019_paper_4.pdf.

[12] Tomaszuk, Dominik and Angles, Renzo and Thakkar, Harsh, PGO: Describing Property Graphs in RDF, IEEE Access 8 (2020). doi:10.1109/ACCESS.2020.3002018.

[13] Hartig, Olaf, Foundations to Query Labeled Property Graphs using SPARQL, in: Joint Proceedings of the 1st International Workshop On Semantics For Transport and the 1st International Workshop on Approaches for Making Data Interoperable co-located with SEMANTiCS, CEUR-WS.org, 2019. URL: https://ceur-ws.org/Vol-2447/paper3.pdf.

[14] Bruyat, Julian and Champin, Pierre-Antoine and Médini, Lionel and Laforest, Frederique, PREC: semantic translation of property graphs, in: 1st workshop on Squaring the Circles on Graphs, SEMANTiCS, 2021. URL: https://hal.science/hal-03407785v1.

[15] Bruyat, Julian and Champin, Pierre-Antoine and Médini, Lionel and Laforest, Frederique, PRSC: From PG to RDF and back, using schemas, Semantic Web (2024). doi:10.3233/SW-243675.

[16] NeoSemantics, Accessed: 2025-01-06. URL: https://github.com/neo4j-labs/neosemantics.

[17] D. Van Assche, T. Delva, G. Haesendonck, P. Heyvaert, B. De Meester, A. Dimou, Declarative RDF graph generation from heterogeneous (semi-)structured data: A systematic literature review, Journal of Web Semantics (2023). doi:https://doi.org/10.1016/j.websem.2022.100753.

[18] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, R. Van de Walle, RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data, in: Proceedings of the 7th Workshop on Linked Data on the Web, CEUR, 2014. URL: http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf.

[19] A. Iglesias-Molina, D. Van Assche, J. Arenas-Guerrero, B. De Meester, C. Debruyne, S. Jozashoori, P. Maria, F. Michel, D. Chaves-Fraga, A. Dimou, The RML Ontology: A Community-Driven Modular Redesign After a Decade of Experience in Mapping Heterogeneous Data to RDF, in: International Semantic Web Conference, Springer, 2023. doi:10.1007/978-3-031-47243-5_9.

[20] S. Das, S. Sundara, R. Cyganiak, R2RML: RDB to RDF Mapping Language, Working Group Recommendation, 2012. URL: http://www.w3.org/TR/r2rml/.

[21] T. Delva, J. Arenas-Guerrero, A. Iglesias-Molina, O. Corcho, D. Chaves-Fraga, A. Dimou, RML-star: A declarative mapping language for RDF-star generation, 2021. URL: https://ceur-ws.org/Vol-2980/paper374.pdf.

[22] APOC, Accessed: 2025-01-06. URL: https://neo4j.com/docs/apoc/current/.

[23] YARRRML, Accessed: 2025-01-06. URL: https://rml.io/yarrrml/.

[24] Bruyat, Julian, From property graphs to knowledge graphs, Theses, INSA Lyon, 2024. URL: https://hal.science/tel-04772451.

[25] D. Beckett, T. Berners-Lee, E. Prud'hommeaux, G. Carothers, RDF 1.1 Turtle – Terse RDF Triple Language, Recommendation, World Wide Web Consortium (W3C), 2014. URL: http://www.w3.org/TR/turtle/.