PREC: semantic translation of property graphs

Julian Bruyat[1], Pierre-Antoine Champin[1,2], Lionel Médini[1], and Frédérique Laforest[1]

[1] Université de Lyon, INSALyon, UCBL, LIRIS CNRS UMR 5205, Lyon, France
{firstname.lastname}@liris.cnrs.fr
[2] W3C / ERCIM, Sophia Antipolis, France

Abstract. Converting property graphs to RDF graphs allows to enhance the interoperability of knowledge graphs. But existing tools perform the same conversion for every graph, regardless of its content. In this paper, we propose PREC, a user-configured conversion of property graphs to RDF graphs to better capture the semantics of the content.

Keywords: RDF · Property graph · Conversion

1 Introduction

Property graphs (PGs) are a family of graph models where nodes and edges can have properties. They are widely used in industry. RDF is a model where each node and relation has only one label. RDF provides better syntactic and semantic interoperability, being a W3C standard.

In this paper, we describe our PG-to-RDF Experimental Converter (PREC)[3] that aims to convert PGs to RDF while semantically enriching the contained data. Unlike other approaches, PREC allows users to explicitly specify this enrichment. We first remind what PGs and RDF graphs are, and review some of the existing tools for converting graphs from one model to the other. We then describe the translation process of PREC. We finally discuss the proposed solution.

2 Definitions and state of the art

In this section, after introducing property graphs and RDF graphs, we review some of the work that already exists to bridge the gap between them.

[3] https://github.com/BruJu/PREC

Acknowledgement: This work was partly supported by the EC within the H2020 Program under grant agreement 825333 (MOSAICrOWN).

2.1 Different types of graphs

Property graphs Property graphs are not a unique model: each vendor (Neo4j, Amazon Neptune, OrientDB...) proposes its own set of features for its property graph model. We focus on a model covering a large majority of PG implementations: in this paper we define a PG as a graph with nodes and edges where each edge has a source node, a destination node and one label. A label is a string that describes the relation between the two linked nodes. Nodes can have any number of labels, including none. Nodes and edges can also have properties: a list of key-value pairs, where the key is a string and the value can be a string, a number or a list thereof. Properties can also have properties, called meta-properties.

Resource Description Framework graphs (RDF) The RDF graph model [12] is a W3C standard. An RDF graph is defined as a set of $(subject, predicate, object)$ triples. The terms composing a triple can either be IRIs (Internationalized Resource Identifier), blank nodes (except for predicates) or literals (for objects only).

Unlike PGs, RDF has a default semantic: each triple of the RDF graph is deemed true. Removing any triple does not change the truth value of other triples (any RDF graph must entail any subset of itself). Moreover, IRIs are assumed to have a shared semantics, defined by their issuer.

2.2 Existing syntax

For an end user, the most notable difference between the RDF model and the PG model is the ability of the latter to add properties to the edges of the graph. Since the creation of RDF, many methods to add annotations to triples have been proposed. The standard RDF reification [8] consists in creating a node that represents the triple, and is linked to the subject, predicate and object with `rdf:subject`, `rdf:predicate` and `rdf:object`. Other common methods include singleton properties [9] or even using named graphs. *A tale of two graphs* [4] explores these different methods to translate PG edge properties to RDF. Hartig and Thompson proposed RDF-star [6]: an extension of the RDF model to allow the use of triples as the subject or the object of other triples, which emulates the ability of PGs to add properties to edges.

2.3 Existing converters

Most of the existing work studies how to expose an RDF Graph with a PG API. G2GML [2] uses SPARQL queries to populate a PG. rdf2neo [1] populates a PG to have an API that the authors consider easier to use than the usual RDF APIs. Gremlinator [10] exposes a Gremlin endpoint to query an RDF graph.

To the best of our knowledge, only two systems allow to convert from PG to RDF. Neosemantics[4] enables users to convert in both directions. Moreover it

[4] https://github.com/neo4j-labs/neosemantics

aims at enriching the PG model with RDF features, *e.g.* inference and model validation with SHACL. Tomaszuk et al. [11] developed the property graph ontology (PGO) implemented in the graphConv tool. It provides a proper way to describe PGs by using RDF. The authors proved the reverse transformation is also possible. But neither Neosemantics nor graphConv are configurable: the same conversion is applied regardless of the information stored in the PG. While NeoSemantics creates a contextual node when populating a PG from an RDF graph to be able to convert back to the same RDF graph, this node is not intended to be created or modified by the user.

2.4 Mapping languages

Tools already exist to produce RDF triples from another source. R2RML [3] enables a SQL to RDF conversion through a mapping described by the user. The mapping is described thanks to IRIs that are templated with the SQL table fields. This mapping has been further expanded to other formats, like JSON, with RML [5]. Another noteworthy tool is JSON-LD [7], which lets the user write a context in JSON to produce RDF graphs from a data JSON file.

3 PREC

In this paper, we propose a user-configured PG-to-RDF translation: instead of using the PG as the sole input, PREC also uses a *context* inspired from JSON-LD. Such a context describes the mapping between the terms used in the PG and IRIs, and templates to represent the different properties and edges in the resulting RDF graph. This enables to better capture and translate the implicit semantics contained in the PG. To do so, PREC performs a two-steps translation as shown in Figure 1 and described in the following sections.

3.1 PREC-0: describing the PG in RDF

To tame the great diversity of PG models, we propose to work on a uniform graph model. This uniform model maps each PG to an RDF graph that describes its structure, similarly to the ones produced by graphConv [11].

We built our own model of RDF graph that can describe the nodes, edges, labels and properties of any PG. Most terms used in our model are from the RDF and RDFS namespaces. We also use terms from the property graph ontology. Our own PREC namespace defines some extra terms for property typing. Unlike graphConv, we support meta-properties, and we create a blank node for each label, instead of directly using a literal to represent them.

The PREC-0 mapping from a PG to its description in RDF is reversible: we can rebuild the nodes and the edges from their descriptions. As a blank node is built to identify every node, edge, label and property of the PG, we ensure that no information is lost in the process.

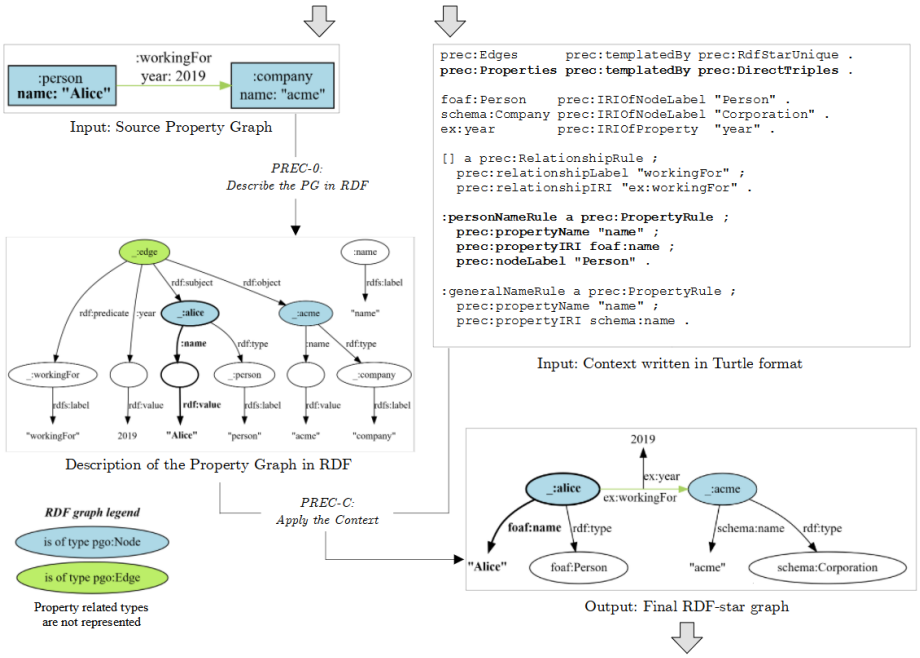

Fig. 1. General overview of PREC.

3.2 PREC-C: applying the context

The originality of our work is the ability to apply a context. A context elicits the semantics of the labels and property names of the PG by mapping them to IRIs and describing how to represent them in RDF.

The context vocabulary[5] A PREC context is an RDF-star graph describing a set of rules. Rules mainly concern node labels, edges and properties.

An example of rule is the `:personNameRule` shown in the context of Figure 1. This rule is applied to the properties with the key "name", only for nodes that hold the label "Person". Each such property is mapped to the IRI `foaf:name` and directly modelled as a triple. In the same example, the name of Alice, a person, is captured by the rule and transformed. The name of acme, a company, is transformed by the other more general rule `:generalNameRule`.

Templating As our goal is to let users describe how to model the PG components, destination templates are not actually hard coded in PREC-C. Listing 1 shows how the `prec:RdfStarUnique` template (used in the example of Figure 1) is defined[6]. In a similar fashion, users can build their own templates.

[5] https://bruy.at/prec

[6] The pvar namespace contains placeholders for the PG elements matched by the rules.

Listing 1. An edge template that maps the PG edge to one RDF triple.

```
prec:RdfStarUnique a prec:EdgeTemplate ; prec:composedOf
    # The edge as an asserted triple
    << pvar:source pvar:edgeIRI pvar:destination >> ,
    # Keep the information that it was an edge in the PG
    << << pvar:source pvar:edgeIRI pvar:destination >> a pgo:Edge >> ,
    # Assign properties to the embedded triple
    << << pvar:source pvar:edgeIRI pvar:destination >>
           pvar:propertyPredicate pvar:propertyObject >> .
```

Design Choices for the Context Vocabulary The context is described in RDF-star. The context vocabulary aims to be agnostic of implementation details, such as the structure of the initial description of the PG in RDF, and the actual implementation of the rules:

- Rules in contexts refer to the labels and property names used in the PG. An end user does not need to know the implementation details of PREC-0 to write a context and use PREC.
- Contexts are declarative rather than imperative: the order in which rules are declared is not relevant, making contexts easier to write and maintain.
- Instead of the declaration order, rules are executed in an order that depends on their type : first meta-property rules, then property rules, then edge rules. It avoids harmful interactions as the production of the latter depends on the production of the former.

4 Preliminary functional evaluation

In terms of PG-to-RDF graph conversion, PREC provides the same options as the two existing tools.

Neosemantics Reproducing the behaviour of Neosemantics to convert Neo4j graphs to RDF is trivial. This can be achieved by keeping the first two lines of the context in Figure 1, *i.e.* modeling every property with the `prec:DirectTriples` template and every edge with the `prec:RdfStarUnique` template.

PG Ontology PREC is also able to produce graphs that follows the PG Ontology [11]. To do so, the user needs to provide a context with explicit templates that describe how nodes, edges and properties are represented[7].

5 Challenges and perspectives

In this paper, we introduced the main motivations behind PREC: a semantic translation from PG to RDF. It is performed through a two-steps conversion: the first step converts a PG to an RDF graph that describes its structure by using a common model. The second one uses a context provided by the user

[7] *e.g.* https://gist.github.com/BruJu/21ef88497217da5e2f46b0eea7cf7cac

that guides the conversion from an RDF description of the PG structure to an idiomatic RDF graph capturing the implicit semantics of the PG.

We are now considering several perspectives for this work:

- *Information loss*: We intend to detect information loss due to the chosen RDF representation of the properties and edges of the PG.
- *Reversibility*: Users may want to revert the process, to go back from RDF to the original PG. As the destination template is free, we have to carefully detect which rule each triple of the graph comes from to build back the graph.
- *Other RDF transformation tools*: the second step of our approach transforms the RDF graph produced by PREC-0 into another RDF graph, for which existing tools (*e.g.* SPARQL CONSTRUCT) could also be used. We plan to study to which point such existing tools could be reused and refocused to the specific needs of PG translation to RDF.

References

1. Brandizi, M., Singh, A., Hassani-Pak, K.: Getting the best of linked data and property graphs: rdf2neo and the knetminer use case. In: SWAT4LS (2018)
2. Chiba, H., Yamanaka, R., Matsumoto, S.: G2gml: Graph to graph mapping language for bridging rdf and property graphs. In: International Semantic Web Conference. pp. 160–175. Springer (2020)
3. Cyganiak, R., Sundara, S., Das, S.: R2RML: RDB to RDF mapping language. W3C recommendation, W3C (Sep 2012), https://www.w3.org/TR/2012/REC-r2rml-20120927/
4. Das, S., Srinivasan, J., Perry, M., Chong, E.I., Banerjee, J.: A tale of two graphs: Property graphs as rdf in oracle. In: EDBT. pp. 762–773 (2014)
5. Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: Rml: a generic language for integrated rdf mappings of heterogeneous data. In: Ldow (2014)
6. Hartig, O., Thompson, B.: Foundations of an alternative approach to reification in RDF. CoRR (2014), http://arxiv.org/abs/1406.3399
7. Kellogg, G., Champin, P.A., Longley, D.: Json-ld 1.1–a json-based serialization for linked data. W3C recommendation, W3C (2020), https://www.w3.org/TR/json-ld11/
8. Manola, F., Miller, E.: RDF primer. W3C recommendation, W3C (Feb 2004), https://www.w3.org/TR/2004/REC-rdf-primer-20040210/
9. Nguyen, V., Bodenreider, O., Sheth, A.: Don't like rdf reification? making statements about statements using singleton property. In: Proceedings of the 23rd international conference on World wide web. pp. 759–770 (2014)
10. Thakkar, H., Punjani, D., Lehmann, J., Auer, S.: Two for one: Querying property graph databases using sparql via gremlinator. In: Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA). pp. 1–5 (2018)
11. Tomaszuk, D., Angles, R., Thakkar, H.: Pgo: Describing property graphs in rdf. IEEE Access 8 pp. 118355–118369 (2020)
12. Wood, D., Lanthaler, M., Cyganiak, R.: RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C (Feb 2014), https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/