Preserving Information Content in RDF Using Bounded Homomorphisms

Audun Stolpe and Martin G. Skjæveland

Department of Informatics, University of Oslo, Norway
{audus,martige}@ifi.uio.no

Abstract. The topic of study in the present paper is the class of RDF homomorphisms that substitute one predicate for another throughout a set of RDF triples, on the condition that the predicate in question is not also a subject or object. These maps turn out to be suitable for reasoning about similarities in information content between two or more RDF graphs. As such they are very useful e.g. for migrating data from one RDF vocabulary to another. In this paper we address a particular instance of this problem and try to provide an answer to the question of when we are licensed to say that data is being transformed, reused or merged in a non-distortive manner. We place this problem in the context of RDF and Linked Data, and study the problem in relation to SPARQL construct queries.

1 Introduction

As yet, the World Wide Web shows a bias towards getting the information to flow, at the expense of maintaining the integrity of the circulated information. Maintaining integrity is usually recognised as a very real and increasingly acute need, though. Take public sector information: open public sector information is a valuable national resource, and there is widespread agreement that dissemination promotes transparent and accountable government, improves quality of service, and in general serves to maintain a well-informed public. Yet, whilst the political pressure for reusable public sector information is building momentum, as witnessed e.g. by the European Public Sector Information Directive of 2003, governments as suppliers and authoritative sources of information on the Web must nevertheless acknowledge the challenges related to maintaining the primary nature of its information. This points to a general tension between two fundamental requirements of the data-oriented Web: Keep the data freely flowing, but shepherd the data into sanctioned use. In the present paper we shall place this problem in the context of RDF and Linked Data, and study it in relation to SPARQL construct queries.

Example 1. The Cultural Heritage Management Office in Oslo is the City of Oslo's adviser on questions of cultural conservation of architecturally and culturally valuable buildings, sites and environments. It maintains a list of protected buildings, known as 'the yellow list', which has been transformed to RDF and published as Linked Data [11]. A small excerpt is given below:

E. Simperl et al. (Eds.): ESWC 2012, LNCS 7295, pp. 72–86, 2012.

```
<http://sws.ifi.uio.no/gulliste/kulturminne/208/5/6643335/597618>
  rdf:type gul:Kontor ;
  hvor:gateadresse "Akersgata 44" ;   geo:long "10.749" ;
  hvor:postnummer  "0180" ;           geo:lat  "59.916" .
```

Note that there is no explicit representation of city or country, and no grouping of similar data. Suppose now that we wish to lift all available information about culturally valuable buildings in Norway to the national level. We do so by adding Oslo and Norway as parts of the address data. Also, we add types to buildings by linking to the relevant class from the CIDOC CRM standard for cultural heritage information (`http://www.cidoc-crm.org/`). For heuristic purposes we also group geographical information and address information respectively around suitably typed nodes:

```
CONSTRUCT{ ?x rdf:type ?y, cidoc:E25.Man-Made_Feature;
     vcard:adr [ rdf:type vcard:Address;
                 vcard:street-address ?street;  vcard:zip-code ?code;
                 vcard:locality geonames:3143242; # Oslo
                 vcard:country-name "Norway"@en ] ;
     vcard:geo [ rdf:type geo:Point;  geo:lat ?lat;  geo:long ?long ] }
WHERE{ ?x rdf:type ?y;
          hvor:gateadresse ?street;  hvor:postnummer ?code;
          geo:lat ?lat;  geo:long ?long . }
```

The structural change to the data caused by the construct query is rather thoroughgoing and extensive. Yet, there is still a principled relationship between structural elements of the two graphs, e.g. the property `hvor:gateadresse` morphs into the sequence of properties `vcard:adr`, `vcard:street-address`. Moreover, the pairs of resources that are linked by `hvor:gateadresse` in the former graph remain linked by `vcard:adr`, `vcard:street-address` in the latter graph, and no other pairs are similarly related. Indeed, the transformation can easily be seen to be systematic in the sense that all pairs related in the same manner in the source graph are transformed uniformly in terms of the same structural element in the target graph. It is also non-distortive in the sense that no other pair of resources are so related. Contrast with the case in which we replace `hvor:postnummer` with `vcard:locality`, whilst keeping everything else as-is. We would then not be able to distinguish between cities and zip-codes in the target graph, and would in that sense have distorted the information from the source.

The purpose of the present paper is to give these intuitions mathematical content. That is, we wish to formulate a criterion to sort conservative from non-conservative ways of transforming data. Since we take construct queries as our paradigm of RDF transformation, this means sorting conservative from non-conservative construct queries. It is important to note that the uniformity and non-distortiveness criteria we shall propose are purely structural, and do not heed the semantics of the vocabulary elements involved. To the question 'what makes the chain `vcard:adr`, `vcard:zip-code` an adequate representation of `hvor:gateadresse`?' the only answer is 'because somebody wishes it to be so'. What our criteria have to offer is thus nothing more than a clear notion of what

you are rationally committed to, in terms of the structure of the target graph, once you have made your choice of representatives for elements of the source graph. We will do so by studying a class of RDF homomorphisms that substitutes one edge for another throughout a set of RDF triples, subject to the condition that the edge in question is not also a vertex.

The paper is organised as follows: Section 2 defines the general concept of an RDF homomorphism, and distinguishes the subset of conditional edge-substitutions mentioned above. We shall call them p-maps. Section 3 recapitulates the basic syntax and semantics of the SPARQL query language. Section 4 defines the notion of a bounded p-map and argues that it gives an adequate criterion of conservativeness. Section 5 generalises the conservativeness criterion to handle more sophisticated construct queries, e.g. that of Example 1. Section 6 presents essential results on the computational properties of computing p-maps, whilst Section 7 closes with a summary and a few pointers to future lines of research.

Related Work. Our homomorphisms differ from those of [1, 2] which essentially rename blank nodes in order to mimic the semantics of RDF as defined in [6]. To the best of our knowledge, our particular notion of RDF homomorphism, and the use of it, is novel. Considered as an embedding of one graph into another a p-map can be viewed in two different ways which, although they amount to the same formally, give rather different gestalts to the central issue. Looked at from one angle, our problem (i.e. embedding a source into a target) resembles data exchange: Given one source of data marked up in one way, one wants to migrate the data to some target repository in a way that conforms to the target's schema. Yet, it differs from the problem studied in [4] in that our setup takes the target to be fixed and possibly non-empty. Looked at from another angle, the problem concerns how to *extend* an RDF graph conservatively. More specifically, it concerns the problem of how to ensure that a transformation of source data into a target repository does not interfere with the assertive content of the source. Yet, it is unlike logic-based conservative extensions [5, 7, 8] in that the logical vocabulary is being replaced as the source is 'extended' into the target. As such bounded p-maps may also have a role to play in *data fusion*, which is defined as "the process of fusing multiple records representing the same real-world object into a single, consistent, and clean representation" [3].

2 RDF Graphs and RDF Homomorphisms

Let U, B and L be pairwise disjoint infinite sets of *URI references*, *blank nodes* and *literals*, respectively, and let $\mathcal{U}$ denote the union of these sets. An *RDF triple* is a member of $\mathcal{T} := (U \cup B) \times U \times \mathcal{U}$. We shall write RDF triples as $\langle a, p, b\rangle$ and say that a and b are *vertexes*, and p the *edge* of the triple. An *RDF graph* is a finite set of RDF triples. The *vocabulary* of an RDF graph G is the set $\mathcal{U}_G = V_G \cup E_G$, where V_G is the set of vertexes and E_G the set of edges in G. Note that V_G and E_G need not be disjoint—a matter of some importance as we shall see later. $\pi_i(t)$ denotes the i-th element of the tuple or sequence t.

Definition 1. *An* RDF homomorphism *of RDF graph G to RDF graph H is a function $h : \mathcal{U}_G \longrightarrow \mathcal{U}_H$ which induces a function $h : G \longrightarrow H$ such that $h(\langle a, p, b\rangle) = \langle h(a), h(p), h(b)\rangle \in H$.*

Definition 2. *A* p-map *$h : G \longrightarrow H$ is an RDF homomorphism $h : G \longrightarrow H$ where $h(u) = u$ for all $u \in V_G$.*

Thus, a p-map is an RDF homomorphism in which the only elements that are allowed to vary are edges: If $h : G \longrightarrow H$ is an RDF homomorphism between RDF graphs G and H, then $h(g) \in H$ for all triples $g \in G$, while if h is a p-map, then $\langle a, h(p), b\rangle \in H$ for all triples $\langle a, p, b\rangle \in G$. This is a natural class of homomorphisms to study for our purposes since edges are typically vocabulary elements, while vertexes contain the "real" data. Note, though, that the definition of a p-map is not without subtleties, given that a single element in an RDF graph may be both a vertex and an edge:

Proposition 1. *Let h be a p-map of G and let $\langle a, p, b\rangle$ be any arbitrarily chosen triple in G. Then $h(\langle a, p, b\rangle) = \langle a, p, b\rangle$ whenever $p \in V_G$.*

3 Syntax and Semantics of SPARQL

To make this paper reasonably self-contained, we introduce a minimum of SPARQL syntax and semantics, considering only the select-project-join fragment. For a complete exposition of SPARQL, consult e.g. [1, 9, 10].

Assume the existence of a set of variables $\mathcal{V}$ disjoint from $\mathcal{U}$. A SPARQL graph pattern is defined recursively as follows:

Definition 3. *A* SPARQL graph pattern *S is either a triple pattern t in $(\mathcal{V} \cup U) \times (\mathcal{V} \cup U) \times (\mathcal{V} \cup U \cup L)$, or a* conjunction *$S_1 \& S_2$ of SPARQL graph patterns.*

According to this definition SPARQL graph patterns do not contain blank nodes. As shown in [1] it is easy to extend the definition in this respect, but as blank nodes behave like variables in select queries, we shall not care to do so. We use $var(S)$ to denote the set of variables occurring in a set of triples S, and $var_p(S)$ to denote those occurring as edges, i.e. in the second element of triples.

Definition 4. *A* conjunctive SPARQL query*, or just 'select query', is a pair $\langle S, \boldsymbol{x}\rangle$, where S is a SPARQL graph pattern and $\boldsymbol{x}$ a subset of $var(S)$.*

A *variable mapping* is a partial function $\mu : \mathcal{V} \longrightarrow \mathcal{U}$ that extends in the natural way to a function on triples which respects RDF triple grammar, i.e. $\mu(S) \subseteq \mathcal{T}$ for any triple pattern S. The domain of a function f is denoted $dom(f)$, and the range by $ran(f)$. The semantics of the select-project-join fragment of select and construct SPARQL queries can now be given by the following series of definitions, modelled after [1, 9]:

Definition 5. *μ_1 and μ_2 are* compatible variable mappings *if for every $x \in dom(\mu_1) \cap dom(\mu_2)$ we have $\mu_1(x) = \mu_2(x)$.*

Definition 6. *Let Ω_1 and Ω_2 be sets of variable mappings. We define the* join *of Ω_1 and Ω_2 as $\Omega_1 \bowtie \Omega_2 := \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2, \mu_1 \text{ and } \mu_2 \text{ compatible}\}$.*

Definition 7. *The* evaluation of S over an RDF graph G, *written $[\![S]\!]_G$, is*

1. *$\{\mu \mid dom(\mu) = var(t)$ and $\mu(t) \in G\}$, if S is a triple pattern t,*
2. *$[\![S_1]\!]_G \bowtie [\![S_2]\!]_G$, if S is a conjunction $S_1 \& S_2$*

Definition 8. *The* answer to a query $\langle S, \boldsymbol{x}\rangle$ over a graph G, *written $\langle S, \boldsymbol{x}\rangle(G)$, is the set $\{\mu(\boldsymbol{x}) \mid \mu \in [\![S]\!]_G\}$.*

Definition 9. *A* SPARQL template *is a SPARQL graph pattern in which blank nodes may occur as vertexes.*

Definition 10. *Let C be a SPARQL template, S a SPARQL graph pattern, G an RDF graph, and $blank(T)$ be the set of blank nodes occurring in a set of triples T. We define the set of* renaming functions *$\{\rho_\mu \mid \mu \in [\![S]\!]_G\}$ relative to C and S as follows:*

- *for every ρ_μ, $dom(\rho_\mu) = blank(C)$ and $ran(\rho_\mu) \subseteq (B \setminus blank(G))$,*
- *every ρ_μ is injective, and*
- *for all $\mu_1, \mu_2 \in [\![S]\!]_G$, if $\mu_1 \neq \mu_2$, then $ran(\rho_{\mu_1}) \neq ran(\rho_{\mu_2})$.*

Definition 11. *A* SPARQL construct query*, or just 'construct query', is a pair $\langle C, S\rangle$, where C is a SPARQL template and S a SPARQL graph pattern such that $var(C) \subseteq var(S)$. The answer to a construct query $\langle C, S\rangle$ over an RDF graph G, written $\langle C, S\rangle(G)$, is the RDF graph $\cup_{\mu \in [\![S]\!]_G}(\mu(\rho_\mu(C))$.*

We end this section with a lemma that links the principal notions introduced so far. It shows, essentially, that answers to queries and evaluations of SPARQL patterns are interchangeable idioms for talking about transformations of RDF graphs:

Lemma 1. *Let G and H be RDF graphs, and h any function from $\mathcal{U}_G$ to $\mathcal{U}_H$. Then,*

1. *$\langle S, \boldsymbol{x}\rangle(G) \subseteq \langle h(S), \boldsymbol{x}\rangle(H)$ iff $[\![S]\!]_G \subseteq [\![h(S)]\!]_H$.*
2. *$\langle h(S), \boldsymbol{x}\rangle(H) \subseteq \langle S, \boldsymbol{x}\rangle(G)$ iff $[\![h(S)]\!]_H \subseteq [\![S]\!]_G$.*

Proof. The claim follows immediately from Definition 8 and the fact that $dom(h) \cap \mathcal{V} = \emptyset$, whence $var(S) = var(h(S))$. □

4 Degrees of Conservativeness

Having assembled the requisite preliminaries, we turn to the problem of analysing the notion of a conservative construct query. We shall limit the analysis in this section to the simple case where the query transforms RDF triples to RDF triples. Let G be any RDF graph. As a tentative characterisation we may say that a construct query is conservative if applied to G it evaluates to a graph

that conservatively transforms the sub-graph of G that matches the pattern in the `SELECT` clause. This pushes the question back to what it means for an RDF graph to be a conservative transformation of another. As the reader may suspect already, we shall take the existence of a particular kind of p-map between two graphs to provide a sufficient condition. As homomorphisms, p-maps in general reflect the structure of the source in the target. A simple consequence of this is that queries over the source can be translated to queries over the target without loss of tuples in the result set:

Theorem 1. *Let G and H be RDF graphs, $h : G \longrightarrow H$ a p-map, and S a SPARQL pattern such that $var_p(S) = \emptyset$. Then $\langle S, \boldsymbol{x} \rangle (G) \subseteq \langle h(S), \boldsymbol{x} \rangle (H)$.*

The existence of a p-map between the source and the target graph may thus be taken to account for the systematicity of a construct query, as alluded to in Section 1. It does not account for non-distortiveness for which we also need to reflect the structure of the result back into the source. We shall consider three ways of doing that, represented by *bounds* on p-maps:

Definition 12. *A p-map $h : G \longrightarrow H$ is* bounded*, and called a* p1-, p2- or p3-map*, respectively, if it satisfies one of the following conditions; for all $a, p, b \in \mathcal{U}$:*

$$\langle a, h(p), b \rangle \in H \Rightarrow \langle a, p, b \rangle \in G \tag{p1}$$

$$\langle a, h(p), h(b) \rangle \in H \text{ or } \langle h(a), h(p), b \rangle \in H \Rightarrow \langle a, p, b \rangle \in G \tag{p2}$$

$$\langle h(a), h(p), h(b) \rangle \in H \Rightarrow \langle a, p, b \rangle \in G \tag{p3}$$

As we shall see, each bound reflects a different aspect of the structure of the target in the source. It is easy to check that (p1) is strictly stronger than (p2), and that (p2) is strictly stronger than (p3). To be sure, there are other bounds, but these are particularly simple and natural. We shall need the following lemma:

Lemma 2. *If $var_p(t) = \emptyset$ and $\langle t, \boldsymbol{x} \rangle (G) \neq \emptyset$ for a triple pattern t, then $\pi_2(h(t)) = h(\pi_2(t))$ for any p-map h.*

Turning now to condition (p1) we obtain the converse of Theorem 1:

Theorem 2. *If $\langle S, \boldsymbol{x} \rangle (G) \neq \emptyset$, $var_p(S) = \emptyset$ and h is a p1-map $h : G \longrightarrow H$, then $\langle h(S), \boldsymbol{x} \rangle (H) \subseteq \langle S, \boldsymbol{x} \rangle (G)$.*

Proof. Assume the conditions of the theorem hold. By Lemma 1 it suffices to show that $[\![h(S)]\!]_H \subseteq [\![S]\!]_G$. The proof proceeds by induction on the complexity of S. For the base case, suppose S is a triple pattern t and that $\mu \in [\![h(t)]\!]_H$. By Definition 7(1) it follows that $\mu(h(t)) \in H$. By the suppositions of the theorem we have $var_p(t) = \emptyset$ and $\langle t, \boldsymbol{x} \rangle (G) \neq \emptyset$, whence Lemma 2 yields $\pi_2(h(t)) = h(\pi_2(t)) = h(p)$ for some $p \in \pi_2(G)$. Therefore $\mu(h(t)) = \langle a, h(p), b \rangle \in H$ for some a, b, whence $\langle a, p, b \rangle \in G$ since h is a p1-map. By Definition 7(1) $dom(\mu) = var(h(t)) = var(t)$, so $\mu \in [\![t]\!]_G$ as desired. For the induction step, assume the property holds for simpler patterns, and consider $S = S_b \& S_c$ such that the suppositions of the theorem hold, and such that $\mu \in [\![h(S_b \& S_c)]\!]_H$. It is

easy to check that $[\![h(S_b \& S_c)]\!]_H = [\![h(S_b) \& h(S_c)]\!]_H$, whence $\mu \in [\![h(S_b)]\!]_H \bowtie [\![h(S_c)]\!]_H$, by Definition 7(2). It follows from Definition 6 that $\mu = \mu_b \cup \mu_c$ for compatible μ_b and μ_c such that $\mu_b \in [\![h(S_b)]\!]_H$ and $\mu_c \in [\![h(S_c)]\!]_H$. Now, since $\langle S_b \& S_c, \boldsymbol{x} \rangle (G) \neq \emptyset$ and $var_p(S_b \& S_c) = \emptyset$, by the supposition of the case, we have $\langle S_b, \boldsymbol{y} \rangle (G) \neq \emptyset$ and $var_p(S_b) = \emptyset$ for $\boldsymbol{y}$ such that $y_i \in dom(\mu_b)$ for all $y_i \in \boldsymbol{y}$ and similarly for S_c. Therefore the induction hypothesis applies, so $\mu_b \in [\![S_b]\!]_H$ and $\mu_c \in [\![S_c]\!]_H$ by Lemma 1. We have already assumed that μ_b and μ_c are compatible, so $\mu_b \cup \mu_c \in [\![S_b]\!]_G \bowtie [\![S_c]\!]_G = [\![S_b \& S_c]\!]_G$ by Definition 7(2). Since $\mu_b \cup \mu_c = \mu$, we are done. □

Theorem 1 and Theorem 2 show that p1-maps induces a transformation between RDF graphs that is exact in the sense that the diagram in Figure 1 commutes. That is, whatever answer a query Q yields over G, $h(Q)$ yields precisely the same answer over H. Interestingly, the converse is also true, if a function induces an, in this sense, exact transformation between graphs, then it is a p1-map:

Theorem 3. *Let h be any function from $\mathcal{U}$ to itself. If for all SPARQL patterns S we have $\langle S, \boldsymbol{x} \rangle (G) = \langle h(S), \boldsymbol{x} \rangle (H)$, then h is a p1-map of G to H.*

Proof. The proof is by induction on the complexity of S. The induction step is easy, so we show only the base case where S is a triple pattern t. Suppose that h is not a homomorphism between G and H. Then there is a triple $\langle a, p, b \rangle \in G$ such that $\langle h(a), h(p), h(b) \rangle \notin H$. Let $t := \langle x, p, y \rangle$ and $\boldsymbol{x} := \langle x, y \rangle$. Then $h(t) = \langle x, h(p), y \rangle$, and $\langle a, b \rangle \in \langle t, \boldsymbol{x} \rangle (G) \setminus \langle h(t), \boldsymbol{x} \rangle (H)$. We therefore have $\langle t, \boldsymbol{x} \rangle (G) \not\subseteq \langle h(t), \boldsymbol{x} \rangle (H)$. Suppose next that h does not satisfy (p1). Then there is a triple $\langle a, h(p), b \rangle \in H$ such that $\langle a, p, b \rangle \notin G$, and $t := \langle x, p, y \rangle$ separates G and H by a similar argument. □

The class of p1-maps thus completely characterises the pairs of graphs for which there is an exact triple-to-triple translation of select queries from one to the other. Note that exactness here does not mean that the source and target are isomorphic. The target may contain more information in the form of triples, as long as these triples do not have source edges that map to them. Indeed, a p1-map need not even be injective:

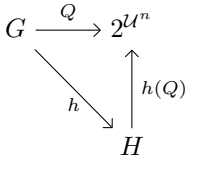


Fig. 1.

Example 2. Assume we have the following RDF graphs: $G := \{\langle a, p, b \rangle, \langle a, q, b \rangle\}$, $H_1 := \{\langle a, r, b \rangle\}$ and $H_2 := \{\langle a, r, b \rangle, \langle c, s, d \rangle\}$. Then $\{p \mapsto r, q \mapsto r\}$ is a p1-map of G to H_1, and of G to H_2, given that h is the identity on vertexes.

Characterisation results similar to Theorem 2 and Theorem 3 are easily forthcoming for p2- and p3-maps as well. The proofs are reruns with minor modifications of that for p1-maps.

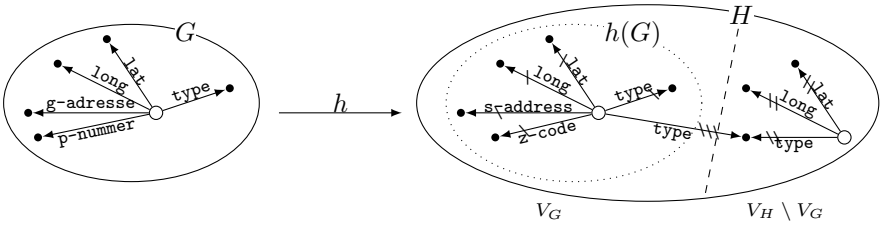
Theorem 4. *Let h be any function from $\mathcal{U}$ to itself and suppose $\langle S, \boldsymbol{x} \rangle (G) \neq \emptyset$ and $var_p(S) = \emptyset$. Then, $h : G \longrightarrow H$ is a p2-map iff $\boldsymbol{u} \in \langle h(S), \boldsymbol{x} \rangle (H) \setminus \langle S, \boldsymbol{x} \rangle (G)$ implies $u \notin \mathcal{U}_G$ for* any *$u \in \boldsymbol{u}$, and $h : G \longrightarrow H$ is a p3-map iff $\boldsymbol{u} \in \langle h(S), \boldsymbol{x} \rangle (H) \setminus \langle S, \boldsymbol{x} \rangle (G)$ implies $u \notin \mathcal{U}_G$ for* some *$u \in \boldsymbol{u}$.*

Different bounds may be used to exercise different levels of control depending on the nature or interpretation of an edge. More specifically, different predicates may be restricted in different ways depending on the intended interpretation of those predicates: p1-maps are suitable for that part of a data-set to which one would wish to remain absolutely faithful, typically the domain-specific information that is collected and managed by the issuer of the data-set. p2-maps are suitable for situations where you would want to merge domain-specific knowledge from two different sources whilst keeping the information from each of the sources unchanged. They are more forgiving than p1-maps in the sense that they allow a relation to grow as long as every added pair relates new objects only. Finally, p3-maps would typically be applied to vocabulary elements that are most aptly considered as part of the logical or general-purpose vocabulary. For instance, applied to `rdf:type`, they allow types to be added to source elements as long as those types are not already used in the source. In other words, p3-maps allow additional typing as long as the added types do not occur in the source.

Example 3. Let G be the excerpt of triples listed in Example 1 and assume it is transformed into the following target H:

```
<http://sws.ifi.uio.no/gulliste/kulturminne/208/5/6643335/597618>
  rdf:type gul:Kontor, cidoc:E25.Man-Made_Feature ;
  vcard:street-address "Akersgata 44" ;   geo:long "10.749" ;
  vcard:zip-code "0180" ;                 geo:lat  "59.916" .
<http://sws.ifi.uio.no/gulliste/kulturminne/999/2/6644406/596768>
  rdf:type cidoc:E25.Man-Made_Feature ;
  geo:long "10.731" ;   geo:lat  "59.926" .
```

The map of the source into the target shows the features of bounded p-maps given in the preceding paragraph. The edges `hvor:gateaddresse` and `hvor:postnummer` are mapped to respectively `vcard:street-address` and `vcard:zip-code` under the (p1) bound, indicating that these edges relate the exact same data as their counterparts in the source. The edges `geo:lat` and `geo:long` are mapped to themselves under bound (p2), meaning that new relationship may be added as long as they relate only new data elements, i.e. elements not originating from the source. The edge `rdf:type` is also mapped to itself under a (p3) bound allowing new types to be added to the buildings in the yellow list (and new buildings be given old types). The transformation is illustrated below:



Marked arrows, $\nrightarrow$, $\nrightarrow$ and $\nrightarrow$, represent triples satisfying bound (p1), (p2) and (p3), respectively. The set of target vertexes is partitioned into two sets, V_G and $V_H \setminus V_G$, illustrated by the dashed line.

Conservativeness (in our sense) is preserved by composition:

Theorem 5. *Let h_1 be a p-map of G to H that satisfies a bound p_i, and h_2 a p-map of H to I that satisfies a bound p_j, and suppose p_i logically entails p_j. Then $h_2 \circ h_1$ is a p-map that satisfies p_j.*

As the theorem shows, a composition of two bounded p-maps will satisfy the weakest of the two bounds. Since they are both conservative in the above-mentioned sense, we can claim that the use of bounded p-maps counteract cumulative error in iterated data transformation.

Note that SPARQL graph patterns and SPARQL templates are similar to RDF graphs in the sense that they too are sets of triples. Thus, we may extend the notion of a p-map accordingly by including variables in the domain and letting the p-map be the identity on those variables. Clearly, if h is a p-map of the latter sort, then we have $var(t) = var(h(t))$ for any triple pattern t. Moreover, for triple patterns where no edge is a variable, μ and h commute: $\mu(h(t)) = h(\mu(t))$. This allows us to prove the following result:

Theorem 6. *Let $\langle C, S\rangle$ be a construct query, where C contains no variables as edges. If h is a p-map of S to C which is bounded by one of (p1)–(p3), then h is a p-map under the same bound of $\langle S, S\rangle(G)$ to $\langle C, S\rangle(G)$.*

Proof. In the limiting case that $[\![S]\!]_G = \emptyset$, we have $\langle C, S\rangle(G) = \emptyset$ as well, whence the theorem holds vacuously. For the principal case where $[\![S]\!]_G \neq \emptyset$ suppose $g \in \langle S, S\rangle(G) = \cup_{\mu\in[\![S]\!]_G}(\mu(\rho_\mu(S))$. By Definition 3 we have that S does not contain blank nodes, so $g \in \cup_{\mu\in[\![S]\!]_G}(\mu(S))$. It follows that $g = \mu(t)$ for a triple pattern t in S and some $\mu \in [\![S]\!]_G$. By assumption, h is a p-map of S to C, whence $h(t)$ is a triple pattern in C, and since $var(h(t)) = var(t)$, it follows that $\mu(h(t)) \in \cup_{\mu\in[\![S]\!]_G}(\mu(\rho_\mu(C))$. It remains to show that $h(g) = \mu(h(t))$. Since $g = \mu(t)$ it suffices to show that $h(\mu(t)) = \mu(h(t))$, which is just the commutativity of μ and h. The relationships between the different graphs are illustrated in Figure 2. Now assume that h from S to C is restricted by a bound (p1)–(p3), indicated by (p) in the figure. Then for every $t \in C$ there is a $t' \in S$ such that the bound holds. For all μ such that $\mu(t) \in \langle C, S\rangle(G)$ we have $\mu(t') \in \langle S, S\rangle(G)$, but then the p-map h must be restricted by the same bound as between $\langle S, S\rangle(G)$ and $\langle C, S\rangle(G)$. □

Thus, if there is a bounded p-map from the `WHERE` block to the `CONSTRUCT` block in a construct query, then any sub-graph that matches the former can be p-mapped with the same bound into the result of the query. By the properties of bounded p-maps, therefore, we are licensed to say that the construct query is a conservative transformation.

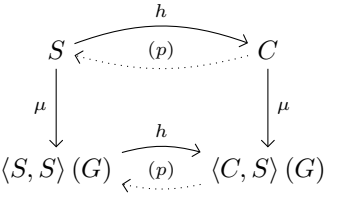


Fig. 2.

5 Generalising the Conservativeness Criterion

We take ourselves to have demonstrated that a bounded p-map is an interesting kind of structure-preserving map for the purpose of maintaining information content across repeated transformation of RDF data. Needless to say triple-to-triple transformations are very restrictive. As Example 1 shows, many construct queries seem to have a legitimate claim to conservativity even though they fall outside of this class. The purpose of the present section is therefore to put the concept of a p-map to more creative use and expand the class of RDF graphs that we recognise as conservative in relation to others. More specifically, we shall generalise the notion of a p-map from a triple-to-triple transformation to a sub-graph to sub-graph transformation, no longer requiring that pairs of vertexes be consistently and non-distortively related by triples—only that they be so related by sub-graphs. The point is to have a transformation that is conservative in much the same sense as a bounded p-map is. That is, the transformation should satisfy the property that if a sub-graph of H is expressed purely in terms of representatives of structural elements of G, then it reflects an actual sub-graph of G. We shall illustrate this approach by considering a function that maps *paths* in G to paths in H. This particular choice is motivated by Example 1 and similar ones which show that many construct queries can indeed be considered as transformations mapping triples to paths, or paths to paths more generally.

Definition 13. *A* walk *is a non-empty sequence α of triples $\alpha := \langle g_1, g_2, \ldots, g_n \rangle$ such that $\pi_3(g_i) = \pi_1(g_{i+1})$ for $1 \leq i < n$. We shall let $len(\alpha) = n$ denote the* length *of α, whilst $px(\alpha) := \pi_1(g_1)$ and $dt(\alpha) := \pi_3(g_n)$ will denote the* proximal *and* distal *vertexes of α, respectively. A* cycle *is a walk α where $dt(\alpha) = px(\alpha)$. A* path *is a walk where no proper segment forms a cycle.*

Note that we only consider finite paths. The set of paths in an RDF graph G, denoted $G^{\uparrow}$, is thus finite too. Cycles are allowed, as long as they do not contain smaller cycles, and triples are considered as unary paths (or unary cycles if the vertexes are the same). We next introduce two equivalence relations on paths:

Definition 14. *Let α and β be paths. We define the equivalence relations $=_V$ and $=_E$ on paths as*

1. *$\alpha =_V \beta$ iff $px(\alpha) = px(\beta)$ and $dt(\alpha) = dt(\beta)$*
2. *$\alpha =_E \beta$ iff α equals β, except that the respective proximal and distal vertexes of α and β may differ.*

In the case where α is a path and g a triple we shall abuse this notation slightly, writing $\alpha =_E g$ to mean $\alpha =_E \langle g \rangle$, and similarly for $=_V$.

Clearly, if two paths are both $=_V$-equivalent and $=_E$-equivalent, then they are the same path. Neither relation factors blank nodes into the notion of sub-graph equivalence. This is an obvious further development which we shall comment on below. By the use of the relations $=_E$ and $=_V$ it is possible to impose restrictions that intuitively constrain the transformation of paths to paths in the same way that a bound constrains the transformation of triples to triples.

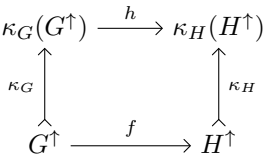


Fig. 3.

Consider the diagram in Figure 3. Here κ_G and κ_H are functions that return the closure under composition—a notion yet to be defined—of the two RDF graphs G and H. We shall say that a bound on maps of paths corresponds to a bound on p-maps if for every f, $G^\uparrow$ and $H^\uparrow$ where $f : G^\uparrow \longrightarrow H^\uparrow$, there is an $h : \kappa_G(G^\uparrow) \longrightarrow \kappa_H(H^\uparrow)$ such that f satisfies the path map bound iff h satisfies the p-map bound. If that is the case, then a consistent relation between triples in the top row is reflected by a consistent relation between paths in the lower row, whence f may be used *in loco parentis* for h to measure the conservativeness of H wrt. G. To that end, we next define the closure under composition of an RDF graph, and state a few properties of this operation:

Definition 15. *A* composition function κ *for an RDF graph* G *is a function of type* $\kappa : G^\uparrow \longrightarrow \mathcal{T}$ *such that*

1. $\alpha =_V \kappa(\alpha)$,
2. $\alpha =_E \beta$ *iff* $\kappa(\alpha) =_E \kappa(\beta)$,
3. $\pi_2(\kappa(\alpha)) = \pi_2(\alpha)$, *if* $len(\alpha) = 1$, *otherwise* $\pi_2(\kappa(\alpha)) \notin \mathcal{U}_G$.

According to this definition, a composition function is such that every non-unary path in G is correlated with a triple whose edge is new to G. Essentially for this reason, composition functions always exist:

Lemma 3. *There is a composition function for every RDF graph* G.

A composition function for G extends G, and puts $G^\uparrow$ and $\kappa(G^\uparrow)$ in one-to-one correspondence, as one would expect:

Proposition 2. *If* κ *is a composition function for* G, *then* $G \subseteq \kappa(G^\uparrow)$.

Lemma 4. *A composition function* κ *for* G *is a bijection between* $G^\uparrow$ *and* $\kappa(G^\uparrow)$.

Turning to path-maps, that is, to functions of type $f : \mathcal{T}^\uparrow \longrightarrow \mathcal{T}^\uparrow$, we are not interested in all path-maps, only those that can be used to 'emulate' p-maps. For want of a better name we shall call them c-maps:

Definition 16. *A* c-map *is a path-map* f *where:*

1. *the relation* $\{(g, g') \mid (\langle g \rangle, \langle g' \rangle) \in f\}$ *is a p-map,*
2. $\alpha =_V f(\alpha)$,
3. *if* $\alpha =_E \beta$, *then* $f(\alpha) =_E f(\beta)$,
4. $len(\alpha) \leq len(f(\alpha))$.

The class of c-maps thus consists of those path-maps that behave like a p-map on unary paths (1), are sensitive to $=_V$- and $=_E$-equivalence (2, 3), and never truncate paths (4). The next theorem shows that every c-map of $G^\uparrow$ to $H^\uparrow$ induces a p-map of $\kappa_G(G)$ to $\kappa_H(H)$, for some κ_G and κ_H:

Lemma 5. *Let κ_G, κ_H be composition functions for RDF graphs G and H, respectively. Then every path-map $f : G^\uparrow \longrightarrow H^\uparrow$ induces a p-map h_f of $\kappa_G(G^\uparrow)$ to $\kappa_H(H^\uparrow)$, defined by letting h_f be the identity on vertexes and putting $h_f(\pi_2(\kappa_G(\alpha))) = \pi_2(\kappa_H(f(\alpha)))$.*

We now generalise the p-map bounds of Definition 12 to bounds on c-maps:

Definition 17. *Suppose α and β are paths in the RDF graphs G and H, respectively. We shall say that a c-map $f : G^\uparrow \longrightarrow H^\uparrow$ is respectively a c1-, c2- or c3-map if one of the following conditions holds:*

$$f(\alpha) =_E \beta \quad \Rightarrow \quad f(\gamma) = \beta \text{ for some } \gamma \in G^\uparrow \quad \text{(c1)}$$
$$f(\alpha) =_E \beta \text{ and } \{px(\beta), dt(\beta)\} \cap V_G \neq \emptyset \quad \Rightarrow \quad f(\gamma) = \beta \text{ for some } \gamma \in G^\uparrow \quad \text{(c2)}$$
$$f(\alpha) =_E \beta \text{ and } \{px(\beta), dt(\beta)\} \subseteq V_G \quad \Rightarrow \quad f(\gamma) = \beta \text{ for some } \gamma \in G^\uparrow \quad \text{(c3)}$$

That these bounds are in fact generalisations of those of Definition 12 is established by the following theorem:

Theorem 7. *Let f be a c-map of RDF graph G to RDF graph H, κ_G and κ_H composition functions for G and H, respectively, and h_f the induced p-map of $\kappa_G(G^\uparrow)$ to $\kappa_H(H^\uparrow)$. Then f is a cn-map iff h_f is a pn-map, for $\mathsf{n} = 1, 2$ or 3.*

Proof. Suppose f is a c3-map and assume that there is a triple $g := \langle a, h_f(p), b\rangle \in \kappa_H(H^\uparrow)$ where $a, b \in V_G$. We need to show that $\langle a, p, b\rangle \in \kappa_G(G^\uparrow)$. Given that κ_G and κ_H are surjective by Lemma 4, let $\alpha \in G^\uparrow$ and $\beta \in H^\uparrow$ be such that $\pi_2(\kappa_G(\alpha)) = p$ and $\kappa_H(\beta) = g$. By the definition of h_f given in Lemma 5, $f(\alpha) =_E \beta$, so by bound (c3) there is a $\gamma \in G^\uparrow$ where $f(\gamma) = \beta$. We have $g =_V \beta =_V \gamma$ by Definition 16 and Definition 15, and $\gamma =_E \alpha$ by Definition 16, since $f(\alpha) =_E \beta$. This means that $\kappa_G(\gamma) = \langle a, p, b\rangle$. For the converse direction, suppose h_f is a p3-map and assume, for some $\alpha \in G^\uparrow$ and $\beta \in H^\uparrow$, that $f(\alpha) =_E \beta$ and $a, b \in V_G$, where $a := px(\beta)$ and $b := dt(\beta)$. By the definition of h_f we have $h_f(\pi_2(\kappa_G(\alpha))) = \pi_2(\kappa_H(\beta))$, so let $\kappa_H(\beta) := \langle a, h_f(p), b\rangle$. Since h_f satisfies (p1), we have $\langle a, p, b\rangle \in \kappa_G(G^\uparrow)$. By Lemma 4, there is a $\gamma \in G^\uparrow$ such that $\kappa_G(\gamma) = \langle a, p, b\rangle$. Since $\kappa_G(\gamma) =_E \kappa_G(\alpha)$, we have $f(\gamma) =_E f(\alpha)$, and given that $\gamma =_V f(\gamma) =_V \beta$, we arrive at $f(\gamma) = \beta$. It is easy to adjust the membership of $a, b, px(\beta), dt(\beta)$ in V_G in the proof and confirm the claim for two other pair of corresponding bounds. □

Expanding the class of conservative construct queries to also handle paths requires the following generalisation of Theorem 6:

Theorem 8. *Let $\langle C, S\rangle$ be a construct query, where C contains no blank nodes and no variables as edges. If f is a cn-map of S to C, then f is a cn-map of $\langle S, S\rangle(G)$ to $\langle C, S\rangle(G)$, for $\mathsf{n} = 1, 2, 3$.*

Proof (Sketch). Let α be a chain in S and let f be a c-map of C to S. If $f(\alpha)$ contains blank nodes then, due to the relabelling function ρ, $f(\alpha)$ may be instantiated differently for each pair of objects that matches the vertexes

of α. This means we may not have $\mu(\rho_\mu(f(\alpha))) =_E \mu'(\rho_{\mu'}(f(\alpha)))$, whence f is not a c-map of $\langle S, S\rangle(G)$ to $\langle C, S\rangle(G)$. In the absence of blank nodes in C this situation cannot arise, ρ becomes redundant, and the proof becomes a straightforward generalisation of that for Theorem 6. □

As this proof-sketch is designed to show, extra care is required when the construct query C contains blank nodes—as it does for instance in Example 1. However, the preceding lemmata and theorems lay out all the essential steps. More specifically, all that is needed in order to accommodate Example 1 and similar ones, is to substitute equivalence classes of paths for paths throughout, where equivalence is equality up to relabelling of blank nodes. The verification of this claim is a rerun with minor modifications, and has therefore been left out.

6 Computational Properties

The problem of deciding whether there exists a homomorphism between two (standard) graphs is well-known to be NP-complete. Since p-maps are more restricted than generic graph homomorphisms, identifying p-maps between RDF graphs is an easier task. In fact it can be done in polynomial time, the verification of which is supported by the following lemmata:

Lemma 6. *Let h_1 and h_2 be p-maps of G_1 and G_2 respectively to H. Then $h_1 \cup h_2$ is a p-map of $G_1 \cup G_2$ to H if $h_1(u) = h_2(u)$ for all $u \in dom(h_1) \cap dom(h_2)$.*

Lemma 7. *If h_1, h_2 are bounded p-maps such that $h_1(u) = h_2(u)$ for all $u \in dom(h_1) \cap dom(h_2)$. Then $h_1 \cup h_2$ is a bounded p-map satisfying the weaker of the two bounds.*

According to Lemma 6 the task of finding a p-map of G to H can be reduced to the task of finding a set of p-maps of *sub-graphs* of G into H that are compatible wrt. to shared domain elements. Lemma 7 then tells us that to check whether the resulting p-map is bounded by some bound pn, it suffices to check whether each of the smaller maps is. This procedure, each step of which is clearly polynomial, does not require any backtracking, whence:

Theorem 9. *Given two RDF graphs G and H, finding a p-map $h : G \longrightarrow H$, bounded or not, is a problem polynomial in the size of G and H.*

Proof (Sketch). For any RDF graphs G and H, fix the set V_G of nodes occurring as vertexes in G. Then for each $p \in E_G$ construct a p-map of $G_p := \{\langle a, p', b\rangle \in G \mid p' = p\}$ into H. This amounts to iterating through the edges of H and finding one, say q, such that i) $\langle a, p, b\rangle \in G_p \rightarrow \langle a, q, b\rangle \in H_q$ and ii) if $p \in V_G$ then $p = q$. Lemma 6 tells us that the union of these maps is a p-map of G to H, i.e. no choice of q for p is a wrong choice. There is therefore no need for backtracking, whence a p-map can be computed in polynomial time. To check whether it satisfies a given bound pn, it suffices by Lemma 7 to check that each of the maps h_p of G_p to H_q does. That is, for each element $\langle a, q, b\rangle \in H_q \setminus G_p$ check that the required triple is in G_p. This is clearly a polynomial check. □

```
Input  : RDF graphs G and H, bound pn.
Output: A p-map h bounded by pn, or ⊥ if none exists.
h := {⟨a, a⟩} for a ∈ V_G;
for p ∈ E_G
    if p ∈ V_G
        if ⟨a, p, b⟩ ∈ G → ⟨a, p, b⟩ ∈ H
            for ⟨c, p, d⟩ ∈ H \ G
                if not CheckBound(⟨c, p, d⟩ , pn) return ⊥;
        else return ⊥;
    else
        bool found := false;;
        for  q ∈ E_H
            if not found and ⟨a, p, b⟩ ∈ G → ⟨a, q, b⟩ ∈ H
                for ⟨c, q, d⟩ ∈ H \ G
                    if not CheckBound(⟨c, q, d⟩ , pn)  break;
                h := h ∪ {⟨p, q⟩};;
                found := true;
        if not found return ⊥;
return h;
```

Algorithm 1. Computing a bounded p-map if one exists. The check for compliance with the bounds is encapsulated in a boolean subroutine `CheckBound`.

For c-maps the situation is more complex. Since the composition of an RDF graph may be exponentially larger than the graph itself, the problem is no longer polynomial. More precisely, if G is an RDF graph and κ a composition function for G, then $|\kappa(G^{\uparrow})| \leq \sum_{n=1}^{|V_G|} |E_G| \times n!$. Yet, this is not a problem for any realistically sized construct query. An experimental application is up and running at `http://sws.ifi.uio.no/MapperDan/`. Mapper Dan takes two RDF graphs or a construct query as input, lets the user specify which bounds to apply to which predicates, and checks whether there is a map under the given bound between the two graphs or between the `WHERE` and `CONSTRUCT` block of the construct query. In the cases where a bound is violated Mapper Dan offers guidance, if possible, as to how to obtain a stratified map which satisfies the bounds. A map can be used to translate the source RDF data to the target vocabulary, produce a construct query which reflects the map, or to rewrite SPARQL queries.

7 Conclusion and Future Work

This paper provides a structural criterion that separates conservative from non-conservative uses of SPARQL construct queries. Conservativity is here measured against the 'asserted content' of the underlying source, which is required to be preserved by the possible change of vocabulary induced by the construct clause. Our problem led us to consider a class of RDF homomorphisms (p-maps) the existence of which guarantees that the source and target interlock in a reciprocal simulation. Viewed as functions from triples to triples, p-maps are computable in

polynomial time. The complexity increases with more complex graph patterns. The class of p-maps has other applications besides that described here, e.g as the basis for a more refined notion of RDF merging. As of today merging is based on the method of taking unions modulo the standardising apart of blank nodes. If one also wants a uniform representation of the data thus collected this method is too crude. What one would want, rather, is a way of transforming the data by swapping vocabulary elements whilst, as far as it goes, preserving the information content of all the involved sources (this is not easily achieved by subsuming a set of properties or types under a common super-type in an ontology). Such a merge procedure may turn out to be an important prerequisite for truly RESTful write operations on the web of linked data.

Acknowledgements. This research was partially funded by the Norwegian Research Council through the Semicolon II project (`http://www.semicolon.no/`).

References

1. Arenas, M., Gutierrez, C., Pérez, J.: Foundations of RDF Databases. In: Tessaris, S., Franconi, E., Eiter, T., Gutierrez, C., Handschuh, S., Rousset, M.-C., Schmidt, R.A. (eds.) Reasoning Web. LNCS, vol. 5689, pp. 158–204. Springer, Heidelberg (2009)
2. Baget, J.-F.: RDF Entailment as a Graph Homomorphism. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 82–96. Springer, Heidelberg (2005)
3. Bleiholder, J., Naumann, F.: Data fusion. ACM Computing Surveys 41(1) (2008)
4. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: semantics and query answering. Theoretical Computer Science 336, 89–124 (2005)
5. Ghilardi, S., Lutz, C., Wolter, F.: Did I Damage my Ontology? A Case for Conservative Extensions in Description Logics. In: Proc. of the 10th Int. Conference on Principles of Knowledge Representation and Reasoning, KR 2006 (2006)
6. Hayes, P.: RDF Semantics. W3C Recommendation, W3C (2004), `http://www.w3.org/TR/rdf-mt/`
7. Hutter, D.: Some Remarks on the Annotation %cons (1999)
8. Makinson, D.C.: Logical Friendliness and Sympathy in Logic. In: Logica Universalis. Birkhäuser, Basel (2005)
9. Pérez, J., Arenas, M., Gutierrez, C.: Semantics of SPARQL. Technical report, Universidad de Chile, TR/DCC-2006-17 (2006)
10. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation, W3C (2008), `http://www.w3.org/TR/rdf-sparql-query/`
11. Stolpe, A., Skjæveland, M.G.: From Spreadsheets to 5-star Linked Data in the Cultural Heritage Domain: A Case Study of the Yellow List. In: Norsk Informatikkonferanse (NIK 2011), Tapir, pp. 13–24 (2011)