Concept Lattices of RDF Graphs

Jens Kötters

Abstract. The concept lattice of an RDF graph is defined. The intents are described by graph patterns rather than sets of attributes, a view that is supported by the fact that RDF data is essentially a graph. A simple formalization by triple graphs defines pattern closures as connected components of graph products. The patterns correspond to conjunctive queries, generalization of properties is supported. The relevance of the notion of connectedness is discussed, showing the impact of ontological considerations on the process of concept formation. Finally, definitions are given which allow the construction of pattern closures of concept graphs, which formalize Conceptual Graphs.

Keywords: RDF Schema, Pattern Concepts, Conceptual Graphs, Navigation

1 Introduction

The Resource Description Framework (RDF) is an extensible standard for knowledge representation which relies on the use of simple sentences, each consisting of a subject, a predicate and an object. In RDF terminology, such a sentence is called a *triple*; a collection of triples is called an *RDF graph*, and the entities which occur as subjects, predicates or objects of triples (i.e.,the things being talked about) are called *resources*. Figure 1 shows an RDF graph in the text-based Turtle notation [5] (namespace declarations are omitted). Figure 2 shows the same RDF graph, using a standard graphical notation (see e.g. [9]). Each triple is represented by an arc. In the abundance of such data, query languages, most notably SPARQL [1], can be used to gain access to the information contained. Querying alone is however of limited use to anyone who needs information but does not know specifically what to look for, or how to ask for it, or maybe whether such information can be found at all. In such cases, concept lattices can in principle guide the exploration of the data source, as they support interactive and data-aware query modification. In connection with SPARQL, this has already been demonstrated in [7]. The current paper is also motivated by data exploration, but deliberately limits the query language to conjunctive queries. In this case, the entire search space is obtained as a concept lattice in which each intent is described by a single *most specific query* (MSQ).

Conjunctive queries can be represented as graphs, and entailment is described by graph homomorphisms [6]. Figure 3 shows a graph representing a conjunctive query – "Someone who spoke about a sailor (in the subject)" – and its SPARQL translation below, which can always be obtained in a straightforward way. Graph queries are modeled after RDF graphs (Fig.2), so that solutions are

Local Proceedings also appeared in ISBN 978-84-606-7410-8, Universidad de Málaga (Dept. Matemática Aplicada), Spain.

```
ex:Frank   ex:loves     ex:Mary .
ex:Frank   rdf:type     ex:Sailor .
ex:Frank   rdf:type     ex:Englishman .
ex:Tom     ex:loves     ex:Mary .
ex:Tom     rdf:type     ex:Gentleman .
ex:Tom     ex:dislikes  ex:Frank .
ex:Frank   ex:dislikes  ex:Tom .
ex:Tom     ex:name      "Tom" .
ex:Tom     ex:said      ex:S1 .
ex:S1      rdf:subj     ex:Tom .
ex:S1      rdf:pred     ex:name .
ex:S1      rdf:obj      "Tom" .
ex:Frank   ex:jested    ex:S2.
ex:S2      rdf:subj     ex:Frank .
ex:S2      rdf:pred     ex:likes .
ex:S2      rdf:obj      ex:Lobscouse .

ex:Liam    ex:loves     ex:Aideen .
ex:Aideen  ex:loves     ex:Liam .
ex:Rowan   ex:loves     ex:Aideen .
ex:Rowan   ex:hates     ex:Liam .
ex:Aideen  ex:said      ex:S3 .
ex:S3      rdf:subj     ex:Rowan .
ex:S3      rdf:pred     ex:likes .
ex:S3      rdf:obj      ex:Aideen .
```

Fig. 1. Sample RDF graph in Turtle notation

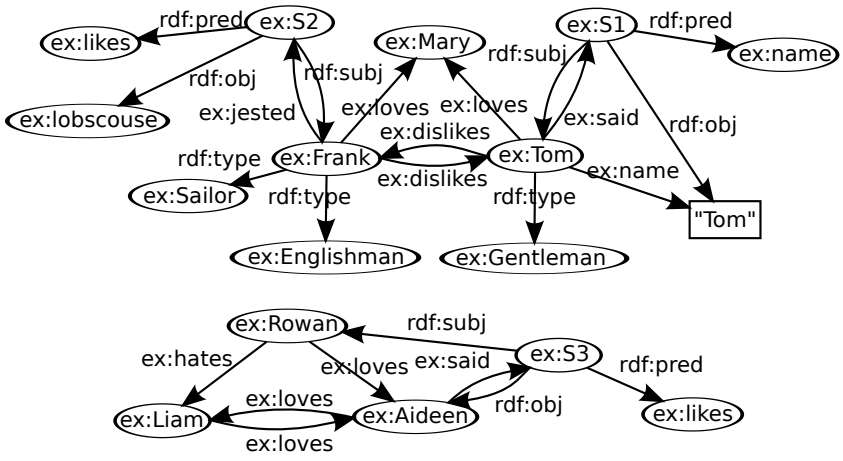
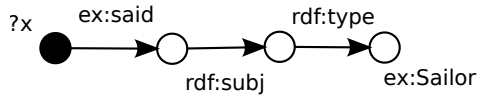


Fig. 2. Drawing of the RDF graph of Fig.1

described by homomorphisms, as well. But there are some differences to RDF graphs. Firstly, all nodes (and all arcs) represent variables. The subject(s) of the query are distinguished in some way (in Fig. 3, a black node represents the single subject), whereas all other variables are implicitly understood as being existentially quantified. Finally, the labels are attributes of a formal context $\mathbb{K}$ (for the example, see Fig. 4), and thus represent formal concepts of its concept lattice $\underline{\mathcal{B}}(\mathbb{K})$. In practice, the formal context may encode background knowledge from an RDF Schema file like the one in Fig. 5. The query vocabulary is restricted to the attributes of $\mathbb{K}$, the name `Frank` e.g. may not be used in a query for the given example.




```
SELECT ?x WHERE { ?x ex:said ?y. ?y rdf:subj ?z. ?z rdf:type ex:Sailor. }
```

Fig. 3. Conjunctive query as a graph (above) and in SPARQL (below)

For the rest of the paper, we shall refer to queries as patterns. Sect. 2 introduces triple graphs as formalizations of queries without distinguished variables (they correspond to SPARQL `WHERE`-clauses), and a query in one distinguished variable is then represented by the pair (x, G) consisting of a triple graph G and a vertex x of G. The case of queries in several variables is not treated in this paper, but has been treated in [11] in connection with relational structures. Section 3 shows how the search space for conjunctive queries over an RDF graph is obtained as a concept lattice, where concept intents are patterns rather than sets of attributes.

Lattices of *pattern concepts* were introduced in [8], and although the theory presented there does not make specific assumptions of what a pattern is, it seems clear that graphs were meant to play a prominent role. In fact, the paper already presents an example application where patterns are chemical compounds and compares them by labeled graph homomorphisms. While chemical compounds are described by connected graphs, their supremum in terms of these homomorphisms is not necessarily connected (*ibid.*, pp. 139-140). Another suggestion made in [8] is to use Conceptual Graphs (CGs) as patterns. If graph patterns are used to describe situations (where several objects combine to a whole by means of the way they are related to each other), we would probably expect them to be connected. But if we formalize CGs using the same kind of labeled graphs that was used for chemical compounds, the set of patterns is generally not closed under suprema because, as we have seen, the supremum operation does not preserve connectedness.

Properties and Classes	name	type	loves	likes	hates	dislikes	jested	said	subj	pred	obj	Englishman	Gentleman	Man	Sailor	Person	"Tom"	lobscouse	Resource
name	×																		×
type		×																	×
loves			×	×															×
likes				×															×
hates					×	×													×
dislikes						×													×
jested							×	×											×
said								×											×
subj									×										×
pred										×									×
obj											×								×
Englishman												×		×		×			×
Gentleman													×	×		×			×
Man														×		×			×
Sailor															×	×			×
Person																×			×
"Tom"																	×		×
lobscouse																		×	×
Resource																			×

Fig. 4. Formal context $\mathbb{K}$ of properties and classes

```
ex:Englishman  rdfs:subClassOf     ex:Man
ex:Gentleman   rdfs:subClassOf     ex:Man
ex:Sailor      rdfs:subClassOf     ex:Person
ex:Man         rdfs:subClassOf     ex:Person
ex:loves       rdfs:subPropertyOf  ex:likes
ex:hates       rdfs:subPropertyOf  ex:dislikes
ex:jested      rdfs:subPropertyOf  ex:said
ex:loves       rdfs:domain         ex:Person
ex:loves       rdfs:range          ex:Person
```

Fig. 5. An RDFS ontology

The situation changes if graphs have a distinguished element. The supremum is a graph product, again with a distinguished element, and although the product of connected graphs is generally not connected, it makes sense to identify the supremum with the connected component which holds the distinguished variable. The relevance of connectedness is discussed in Sect. 4. In Sect. 5, the approach is applied to concept graphs.

2 Triple Graphs

A *triple graph* over $\mathbb{K}$ is a triple (V, T, κ), where V is a set of vertices, $T \subseteq V \times V \times V$ and $\kappa : V \to \underline{\mathcal{B}}(\mathbb{K})$.

A morphism $\varphi : (V_1, T_1, \kappa_1) \to (V_2, T_2, \kappa_2)$ of triple graphs is a map $\varphi : V_1 \to V_2$ with

$$(x, y, z) \in T_1 \Rightarrow (\varphi x, \varphi y, \varphi z) \in T_2, \tag{1}$$

$$\kappa_1(x) \geq \kappa_2(\varphi x) \tag{2}$$

for all $x, y, z \in V_1$.

The product of a family of triple graphs $G_i = (V_i, T_i, \kappa_i)$, $i \in I$, is the triple graph

$$\prod_{i \in I} G_i := (\bigtimes_{i \in I} V_i, \{\ t^{\mathrm{T}} \mid t \in \bigtimes_{i \in I} T_i\}, \bigvee_{i \in I} \kappa_i), \tag{3}$$

where $t^{\mathrm{T}} := ((t_{i0})_{i \in I}, (t_{i1})_{i \in I}, (t_{i2})_{i \in I})$ and $(\bigvee_{i \in I} \kappa_i)(v) := \bigvee_{i \in I} \kappa_i(v_i)$.

Finally, an RDF graph with the set T of triples is represented by the triple graph (R, T, γ^*), where R is the set of resources (properties and classes are duplicated so that they appear in only one triple) and

$$\gamma^*(g) := \begin{cases} (g'', g') \text{ if } g \in G, \\ \top := (G, G') \text{ otherwise} \end{cases} . \tag{4}$$

A *pattern* is a pair (v, H), where $H =: (V, T, \kappa)$ is a triple graph and $v \in V$. A *pattern morphism* $\varphi : (v_1, H_1) \to (v_2, H_2)$ is a morphism $\varphi : H_1 \to H_2$ with $\varphi(v_1) = v_2$. A *solution* is a pattern in the data graph. Pattern morphisms can thus describe solutions as well as entailment.

The product of a family of patterns is given by

$$\prod_{i \in I} (v_i, H_i) := ((v_i)_{i \in I}, \prod_{i \in I} H_i). \tag{5}$$

The definitions above follow a category theoretical approach. Queries (minus distinguished variable(s)) and the data source have representations in the same category (using Δ for the data source), and morphisms $\lambda : G \to \Delta$ can be understood as solutions of the respective query. Query entailment is then described by morphisms, and if the category has products, then so has the category of structured arrows (these are the pairs (ν, G), cf. [2]), which model queries with distinguished variables. The pattern closures (MSQs) arise as products from the set of possible solutions (λ, Δ). For convenience, we write (x, G) instead of (ν, G) if the domain of ν is a one-element set (x being the image of that element).

3 Example RDF Graph and Concept Lattice

In this section, we will walk through the process of generating a concept lattice for the RDF graph in Fig. 1. To keep the example small (and perhaps meaningful), only person concepts will be created, i.e. concepts with extents in the set $\{\texttt{Tom}, \texttt{Frank}, \texttt{Mary}, \texttt{Rowan}, \texttt{Liam}, \texttt{Aideen}\}$.

As a first step, we determine the object intent for each person, i.e. its most complete description. For each person, its connected component in Fig. 2 doubles as a most complete description if we reinterpret the nodes and arcs as (distinct) variables, reinterpret the labels beside them as concept names (rather than resource names), and mark the variable corresponding to that person as distinguished. Let G_1 and G_2 denote, top to bottom, the components in Fig. 2 after reinterpretation, and let T, F, M, R, L and A denote the variables which replace `Tom`, `Frank`, `Mary`, `Rowan`, `Liam` and `Aideen`. Then the pairs (T, G_1), (F, G_1), (M, G_1), (R, G_2), (L, G_2) and (A, G_2) formalize the object intents. These pairs are considered patterns; each pattern consists of a distinguished variable and a triple graph (formalized in Sect. 2).

The person concepts are obtained from graph products of the six base patterns. Many concept intents have the same underlying triple graph (consider e.g. the base patterns), so there would be a lot of redundancy in the concept lattice if we were to draw intents next to each concept. A different kind of diagram avoids this redundancy while still showing all concepts and their intents : Fig. 6 shows all triple graphs which occur in the intents of person concepts, ordered by homomorphism. Choosing any of the nodes as a distinguished variable gives rise to a pattern intent, and this way concepts can be identified with triple graph nodes. The gray nodes in Fig. 6 are the person concepts (the white nodes are non-person concepts which occur in the descriptions of person concepts). The concept extents for the person concepts are drawn next to the gray nodes. The person concept lattice itself can be easily obtained from the diagram in Fig. 6 by connecting the gray nodes according to their extents.

Note however that some concepts have multiple representations in Fig. 6. These may occur in the same triple graph, which indicates symmetry (i.e., a triple graph automorphism). An example is the pair of lovers on the right side of Fig. 6. The other case is where the same concept occurs in different triple graphs. An example would be the Mary concept, which occurs in the lower left triple graph as well as in the triple graph above it on the left side of Fig. 6. In such cases, there is always a most specific triple graph containing that concept; it describes that concept's pattern intent. Other triple graphs containing the same concept can be folded onto the most specific triple graph (which is contained as a subgraph). But the fact that triple graphs may reappear as subgraphs of other triple graphs translates into another redundancy problem when drawing the hierarchy of triple graph product components. In Fig. 6, this kind of redundancy has been avoided by cutting those subgraphs out and indicate the glue nodes (for the gluing operation which inverts the cutting operation) with an asterisk. The glue nodes are only marked in the upper triple graph, not in the lower triple graph, because the lower triple graph can be considered self-contained (a

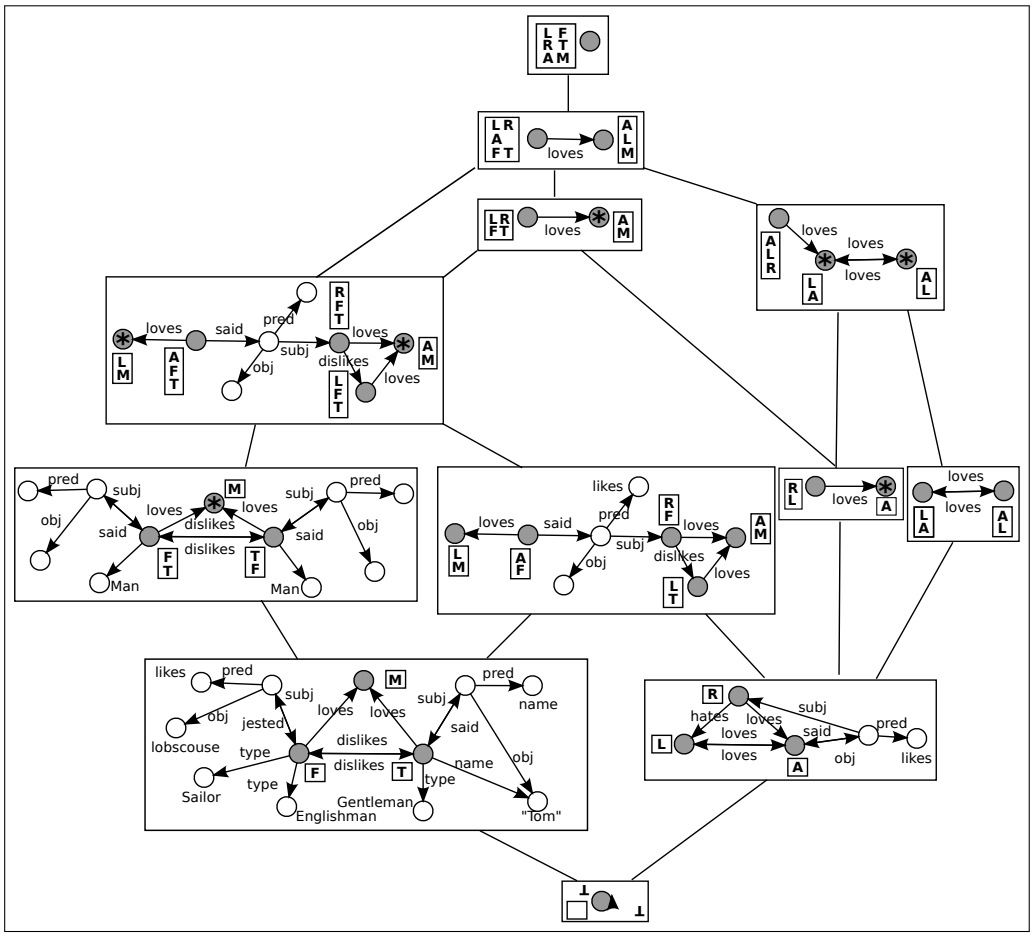


Fig. 6. Concepts of the RDF graph

glue node's counterpart in the lower graph can be identified by its extent). To put this differently: in every triple graph, the asterisk concepts are part of the description of the non-asterisk concepts, but not vice versa.

The extent of a concept reveals how its pattern intent can be obtained. Consider for example the concept with extension $\{R, F, T\}$, which is part of the top left triple graph. The extension tells us that the pattern intent is the (core graph of the) component of the pattern product $(R, G_2)\times(F, G_1)\times(T, G_1)$, which contains the vertex (R, F, T), and (R, F, T) becomes the designated variable. The diagram also shows that $\{R, T\}$ is a generating set for this extent, because it is not contained in a concept extent of the triple graph's lower neighbors, and so $(R, G_2) \times (T, G_1)$ already produces the same pattern. As was described in [8], Ganter's NextConcept algorithm can be used to systematically generate all extents (and thus all concepts), computing pattern products in the process. It is advisable to use graph folding to obtain minimal equivalent patterns, for output as well as inbetween computational steps.

Product patterns could also be computed directly from the triples in Turtle notation (Fig. 1), by combining triples with each other. Informally, we could write triple graphs as sets of triples $(x : \kappa(x), p : \kappa(p), y : \kappa(y))$ (borrowing from RDF and CG notation), and compute the product triples by taking suprema componentwise, like so:

$$\begin{array}{rl} & (F : \top, p : \text{type}, y_1 : \text{Englishman}) \\ \vee & (T : \top, p : \text{type}, y_2 : \text{Gentleman}) \\ = & (\left[\begin{smallmatrix} F \\ T \end{smallmatrix}\right] : \top, \left[\begin{smallmatrix} p \\ p \end{smallmatrix}\right] : \text{type}, \left[\begin{smallmatrix} y_1 \\ y_2 \end{smallmatrix}\right] : \text{Man}). \end{array}$$

However, one would still have to identify and minimize connected components.

Note that the property and class hierarchies in our example are trees. So the attribute concepts of $\mathbb{K}$ are closed under suprema, and each concept label can be expressed by a single attribute. The suprema of properties may be of type "Resource". Such arcs are removed from the patterns; it can be shown (using the product property) that the resulting patterns can still be considered MSQs.

4 Connectedness

We have identified two components of the RDF graph in Fig. 1, and this corresponds to our understanding that objects are "connected" if they contribute to the same instance of a situation. The notion of connectedness deserves closer examination, however. Let us first observe that the notion of strong connectivity is not the right one, because the second to top concepts in Fig. 6 (we might name them "Lover" and "Beloved") would not have been created under this notion. But also, we can in general not rely on weak connectivity alone. If it had been stated explicitly that all persons are of type "Person", all persons would be connected through that class node.

Clearly, objects do not contribute to the same situation just on the basis of belonging to the same class, or having equal property values, like being of the

same age. So connectedness of patterns should not rely on properties, classes or values. In this paper, no further requirements have been made, but if Liam liked lobscouse (a traditional sailors' dish), this would have not been sufficient because all objects would be connected through the lobscouse node. From a machine perspective, there is no indication in the RDF data (or the schema) that "ex:lobscouse" is of a different quality than "ex:Tom" or "ex:Mary". A system that generates patterns from automatically acquired data without proper preprocessing would possibly generate a large number of meaningless (and unnecessarily large) concepts because of such insubstantial connections, unless it can be guaranteed that some standard for making such distinctions is applied on the level of the ontology language.

Further questions may include if and how real-world objects should be connected through objects of spoken language; on what basis we could allow a pattern like the one describing the set of all wifes older than their husband, and at the same time keep meaningless comparison of values from having an impact on concept formation; or if pattern connection should be described in terms of edges rather than nodes, and if we can classify or characterize the kind of relations which contribute to pattern formation. It seems that, when dealing with pattern concepts, questions of philosophical ontology may have (at least subjectively) measurable impact through the notion of connectedness.

5 Related Work

In Sect. 2, the category theoretical framework for lattices of pattern concepts has been applied to triple graphs, and we will now show how it is applied to *simple concept graphs*. In [15], a simple concept graph over a power context family $\vec{\mathbb{K}} = (\mathbb{K}_0, \mathbb{K}_1, \mathbb{K}_2, \dots)$ is defined as a 5-tuple $(V, E, \nu, \kappa, \rho)$. We may interpret the 4-tuple (V, E, ν, κ) as a query, $\vec{\mathbb{K}}$ as a data source and the map ρ as a set of solutions. The map κ assigns concepts of the contexts $\mathbb{K}_i$ to the vertices and edges in V and E, respectively. The definition of queries should be independent from any particular data source, so it makes sense to interpret $\vec{\mathbb{K}}$ as a power context family describing schema information (background knowledge) instead, the contexts could describe attribute implications, like the one in Fig. 4. The map ρ will however not be part of the query. For the rest of the exposition, we shall call the 4-tuple (V, E, ν, κ) an *abstract concept graph* over $\vec{\mathbb{K}}$, after a term used in the first paper on concept graphs [14, p.300].

In [15], the triple (V, E, ν) is called a *relational graph*. The morphisms in the category of abstract concept graphs over $\vec{\mathbb{K}}$ are relational graph morphisms (which replaces condition (1) for triple graphs) satisfying (2). The product is defined in analogy to (3) (cartesian products of edges have to be taken individually for each arity k).

A datasource for the schema $\vec{\mathbb{K}}$ is a power context family $\vec{\mathbb{D}} = (\mathbb{D}_0, \mathbb{D}_1, \mathbb{D}_2, \dots)$ for which the attribute implications described by $\mathbb{K}_i =: (H_i, N_i, J_i)$ are satisfied in $\mathbb{D}_i =: (G_i, M_i, I_i)$, $i = 0, 1, 2, \dots$, and $N_i \subseteq M_i$ holds. We can represent $\vec{\mathbb{D}}$ by an abstract concept graph $\mathfrak{D} = (G_0, \bigcup_{i \geq 1} G_i, \mathrm{id}, \gamma_{\mathfrak{D}})$ with $\gamma_{\mathfrak{D}}(u) := ((u^{I_k} \cap$

$N_k)^{J_k}, u^{I_k} \cap N_k) \in \underline{\mathcal{B}}(\mathbb{K}_k)$ for $u \in G_k$. Let us furthermore define $\mathrm{Ext}_{\mathfrak{D}}(\kappa(u)) := \mathrm{Ext}(\kappa(u))^{J_k I_k}$ for $u \in V \cup E$. If we define a realization of $\mathfrak{G} =: (V, E, \kappa, \nu)$ over $\mathfrak{D}$ as a map with $\rho(u) \in \mathrm{Ext}_{\Delta}(\kappa(u))$ for all $u \in V \cup E$ and $\rho(e) = (\rho(v_1), \ldots, \rho(v_k))$, we obtain that the realizations ρ of $\mathfrak{G}$ over $\mathfrak{D}$ are precisely the morphisms $\rho : \mathfrak{G} \to \mathfrak{D}$. This means that product patterns for abstract concept graphs can be interpreted as MSQs, as was mentioned at the end of Sect. 1.

Triple graphs are now obtained as a special case of concept graphs: We define a power context family $\vec{\mathbb{D}}$ where $\mathbb{D}_0$ is a supercontext of $\mathbb{K}_0$ which in addition contains all objects (having only the "Resource" attribute), and where $\mathbb{D}_3 = (T, \emptyset, \emptyset)$ represents the set of triples.

In Relational Concept Analysis, concept lattices for different kinds of interrelated objects are generated by an iterative algorithm and, as in Fig. 6, illustrations displaying graphs of concepts have been given [4, 10]. Computational aspects of generating pattern structures are covered e.g. in [13, 12]. In [3], concept lattices are generated from Conceptual Graphs, but the approach seems to be tailored towards a more specific case where edges (and thus paths) express dependencies. A comparison with Relational Semantic Systems [16] still has to be made.

6 Conclusion

In [11], a lattice of closures of database queries, which links products of query patterns to the join operation on result tables, has been introduced. The queries and database were formalized by relational structures. The fact that the method could be applied again, first to RDF graphs and then to abstract concept graphs, suggests that the underlying category theoretical notions provide a recipe that could be applied to still different formalizations of queries, allowing in every case the mathematical description of lattices of most specific queries. Combinatorial explosion is a computational problem that quickly turns up when computing the concept lattices. From a practical perspective, the lattices are useless if complexity problems can not be solved. However, in order to support data exploration, patterns must be understandable, thus also limited in size, and a solution to this problem may entail a solution to the problem of computational complexity.

In the section on connectedness, we have seen that ontological considerations stand in a direct relation to the quality of computed concepts. As a first consequence, although the idea of treating all kinds of resources in a homogeneous way seemed appealing, triple graphs must be replaced by some other formalization which reflects the importance of the distinction between instances and classes/properties. While such questions naturally arise in the setting of pattern concepts, they seem to have no obvious analogy for standard FCA, where intents are sets of attributes. Pattern concepts have thus the potential to further and substantiate a perspective on FCA as a branch of mathematical modeling, where the entities to be modeled are not "real world" systems but rather systems of concepts. Future work may be concerned with extensions of RDF/RDFS which support this perspective.

References

1. SPARQL 1.1 Query Language. Tech. rep., W3C (2013), http://www.w3.org/TR/sparql11-query
2. Adámek, J., Herrlich, H., Strecker, G.E.: Abstract and concrete categories : the joy of cats. Pure and applied mathematics, Wiley, New York (1990)
3. Andrews, S., Polovina, S.: A mapping from conceptual graphs to formal concept analysis. In: Andrews, S., Polovina, S., Hill, R., Akhgar, B. (eds.) Proceedings of ICCS 2011. LNCS, vol. 6828, pp. 63 – 76. Springer (2011)
4. Azmeh, Z., Huchard, M., Napoli, A., Hacene, M.R., Valtchev, P.: Querying relational concept lattices. In: Proc. of the 8th Intl. Conf. on Concept Lattices and their Applications (CLA'11). pp. 377–392 (2011)
5. Beckett, D., Berners-Lee, T., Prud'hommeaux, E., Carothers, G.: RDF 1.1 Turtle. Tech. rep., W3C (2014), http://www.w3.org/TR/turtle
6. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational databases. In: Proceedings of the ninth annual ACM symposium on Theory of computing. pp. 77–90. STOC '77, ACM, New York, NY, USA (1977)
7. Ferré, S.: Conceptual navigation in RDF graphs with SPARQL-like queries. In: Kwuida, L., Sertkaya, B. (eds.) Proceedings of ICFCA 2010. LNCS, vol. 5986, pp. 193–208. Springer, Heidelberg (2010)
8. Ganter, B., Kuznetsov, S.O.: Pattern structures and their projections. In: Delugach, H.S., Stumme, G. (eds.) Proceedings of ICCS 2001. LNCS, vol. 2120, pp. 129–142. Springer (2001)
9. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. Chapman & Hall/CRC (2009)
10. Huchard, M., Hacene, M.R., Roume, C., Valtchev, P.: Relational concept discovery in structured datasets. Annals of Mathematics and Artificial Intelligence 49(1-4), 39–76 (2007)
11. Kötters, J.: Concept lattices of a relational structure. In: Pfeiffer, H.D., Ignatov, D.I., Poelmans, J., Gadiraju, N. (eds.) Proceedings of ICCS 2013. LNCS, vol. 7735, pp. 301–310. Springer (2013)
12. Kuznetsov, S.O.: Computing graph-based lattices from smallest projections. In: Wolff, K.E., Palchunov, D.E., Zagoruiko, N.G., Andelfinger, U. (eds.) KONT/KPP. Lecture Notes in Computer Science, vol. 6581, pp. 35–47. Springer (2007)
13. Kuznetsov, S.O.: Fitting pattern structures to knowledge discovery in big data. In: Formal Concept Analysis, 11th International Conference, ICFCA 2013, Dresden, Germany, May 21-24, 2013. Proceedings. pp. 254–266 (2013), http://dx.doi.org/10.1007/978-3-642-38317-5_17
14. Wille, R.: Conceptual graphs and formal concept analysis. In: Lukose, D., Delugach, H.S., Keeler, M., Searle, L., Sowa, J.F. (eds.) Proceedings of ICCS 1997, 5th International Conference on Conceptual Structures. LNCS, vol. 1257, pp. 290–303. Springer, Heidelberg (1997)
15. Wille, R.: Formal concept analysis and contextual logic. In: Hitzler, P., Schärfe, H. (eds.) Conceptual Structures in Practice, pp. 137–173. Chapman & Hall/CRC (2009)
16. Wolff, K.E.: Relational scaling in relational semantic systems. In: Rudolph, S., Dau, F., Kuznetsov, S.O. (eds.) Proceedings of ICCS 2009. LNCS, vol. 5662, pp. 307–320. Springer (2009)