Graph-FCA Meets Pattern Structures

Sébastien Ferré

Graph-FCA Meets Pattern Structures

Sébastien Ferré*

Univ Rennes, CNRS, Inria, IRISA
F-35000 Rennes, France
ferre@irisa.fr

Abstract. A number of extensions have been proposed for Formal Concept Analysis (FCA). Among them, Pattern Structures (PS) bring complex descriptions on objects, as an extension to sets of binary attributes; while Graph-FCA brings n-ary relationships between objects, as well as n-ary concepts. We here introduce a novel extension named Graph-PS that combines the benefits of PS and Graph-FCA. In conceptual terms, Graph-PS can be seen as the *meet* of PS and Graph-FCA, seen as sub-concepts of FCA. We demonstrate how it can be applied to RDFS graphs, handling hierarchies of classes and properties, and patterns on literals such as numbers and dates.

1 Introduction

Formal Concept Analysis (FCA) [17] has been applied to many different tasks – such as information retrieval, recommendation, ontology engineering, or knowledge discovery – and in many application domains, e.g., social sciences, software engineering, bioinformatics or chemoinformatics, natural language processing [15]. The variety of those tasks and application domains early called for FCA extensions in order to handle complex data. Complex data includes non-binary attributes, concrete domains, heterogeneous data, uncertain data, and structured data.

The earlier extensions enable to use complex descriptions of objects in place of sets of binary attributes. Three similar extensions have been introduced at almost the same time: Generalized Formal Concept Analysis [6], Logical Concept Analysis [12], and Pattern Structures [16]. They enable to describe objects with valued attibutes, intervals over numbers and dates, convex polygons, partitions, sequence patterns, tree patterns, or labelled graph patterns [15]. Other extensions address the uncertainty of object descriptions [22], notably Fuzzy FCA [5] where the incidence between an object and an attribute is a truth degree in $[0, 1]$ instead of a crisp Boolean value. Triadic Concept Analysis [21] adds conditions to the incidence between an object and an attribute, making the formal context a ternary relation instead of a binary relation. Polyadic Concept Analysis [24] generalizes this idea by allowing any number of dimensions for the context. Finally, a number of more recent extensions add relationships between objects, so

* This research is supported by ANR project SmartFCA (ANR-21-CE23-0023).

that the concepts do not depend only on the individual descriptions of objects but also on relationship patterns over interconnected objects. Relational Concept Analysis [23] combines several classical FCA contexts and several binary relations to form concepts whose intents are similar to description logic class expressions [3], combining object attributes, binary relations, and quantifier operators. Relational structures [19] and Graph-FCA [8] add n-ary relationships between objects ($n \geq 1$), and form n-ary concepts, i.e. concepts whose intents are equivalent to conjunctive queries, and whose extents are equivalent to the results of such queries, i.e. sets of n-tuples of objects.

In this paper, we propose to merge two FCA extensions that are representative of the first and last categories above: Pattern Structures (PS) and Graph-FCA. The aim is to combine the benefits of the two categories of extensions, in short complex descriptions and relationships between objects. Logical Concept Analysis (LCA) could have been used in place of PS but we have chosen PS as it has been more widely adopted, and because it is better suited to the effective computation of concepts. In this paper we choose to merge PS with Graph-FCA but it would be perfectly relevant to do so with Relational Concept Analysis (RCA). We hope this work will encourage and facilitate the merge with RCA in a future work. The merge results in a new FCA extension called Graph-PS. It is an elegant extension in the sense that PS is a special case of Graph-PS, obtained by not using inter-object relationships; and Graph-FCA is a special case of Graph-PS, obtained by using sets of binary attributes as descriptions of individual objects and inter-object relationships. As a consequence, classical FCA is also a special case of Graph-PS. It therefore acts as an unifying FCA theory encompassing classical FCA and two mainstream FCA extensions.

The paper is structured as follows. Section 2 recalls the main definitions and results of Pattern Structures and Graph-FCA, as preliminaries. Section 3 defines Graph-PS as the extension of Graph-FCA with PS-like descriptions, and illustrates the different notions with a running example combining binary relationships, valued attributes and intervals. Section 4 describes its application to RDFS graphs by defining a custom set of descriptions and similarity operator. Section 5 concludes the paper, and draws some perspectives.

2 Preliminaries

In this section, we recall the main definitions of two extensions of Formal Concept Analyis (FCA): Pattern Structures (PS) [16] and Graph-FCA [11]. The former extends FCA attributes with complex descriptions and patterns. The latter extends FCA with n-ary relations between objects, and n-ary concepts.

2.1 Pattern Structures (PS)

A *pattern structure* is a triple $K = (O, (D, \sqcap), \delta)$ where O is a set of objects, $(D, \sqcap)$ is a meet-semi-lattice of *patterns*, and $\delta \in O \to D$ is a mapping taking

each object to its *description*. The meet operator $d_1 \sqcap d_2$ represents the *similarity* between two patterns d_1 and d_2. It entails a partial ordering $\sqsubseteq$, called *subsumption*, defined for all patterns $d_1, d_2 \in D$ as $d_1 \sqsubseteq d_2 \iff d_1 \sqcap d_2 = d_1$. Conversely, the pattern $d_1 \sqcap d_2$ is the most specific pattern, according to $\sqsubseteq$, that subsumes patterns d_1 and d_2. The *extension* of a pattern d is defined as the set of objects whose description contains the pattern[1].

$$ext(d) := \{o \in O \mid d \sqsubseteq \delta(o)\}, \quad \text{for every } d \in D$$

The *intension* of a set of objects X is defined as the most specific pattern that subsumes the description of all objects in X, i.e. the similarity between all those descriptions.

$$int(X) := \prod_{o \in X} \delta(o), \quad \text{for every } X \subseteq O$$

The two derivation operators (ext, int) form a Galois connection between the two posets $(2^O, \subseteq)$ and $(D, \sqsubseteq)$. A *pattern concept* of a pattern structure $K = (O, (D, \sqcap), \delta)$ is a pair (X, d) with $X \subseteq O$ and $d \in D$ such that $X = ext(d)$ and $d = int(X)$. The component X of a pattern concept (X, d) is called the *extent*, and the component d is called the *intent*. Pattern concepts are partially ordered by $(X_1, d_1) \leq (X_2, d_2) \Leftrightarrow X_1 \subseteq X_2 \Leftrightarrow d_2 \sqsubseteq d_1$. This partial ordering forms a complete lattice called the *pattern concept lattice*.

Pattern structures were first applied to labeled graphs [16,20], e.g. to discover frequent patterns in molecular structures. They were then applied to various types of descriptions: e.g., numbers and intervals [18], partitions for characterizing functional dependencies [4], RDF graphs [1].

Classical FCA is the special case of PS when $D := 2^A$ for a given set of attributes A, and the similarity between two sets of attributes is their intersection: $(\sqcap) := (\cap)$.

2.2 Graph-FCA

We first need to introduce notations for tuples. Given a set X of elements, a *k-tuple* of elements is an ordered collection of k elements that is written $(x_1, \ldots, x_k)$. For the sake of concision, a tuple is often written as an overlined letter $\overline{x}$, whose element at position i can be written $\overline{x}[i]$, or simply x_i if there is no ambiguity. The set of all k-tuples over X is written X^k. The set of all tuples of any arity is written $X^* = \bigcup_{k \geq 0} X^k$. The latter includes the empty tuple $()$ for arity 0.

Graph-FCA extends a formal context into a *graph context*, defined as a triple $K = (O, A, I)$ where O is a set of objects, A is a set of attributes, and $I \subseteq O^* \times A$ is an incidence relation between *tuples of objects* $\overline{o} \in O^*$ and attributes $a \in A$. Objects are graph nodes, attributes are graph labels, and an

[1] The original notation for the two PS derivation operators is $(.)^{\square}$. We use the notations $ext(.)$ and $int(.)$ because they are more explicit and also consistent with notations in other FCA extensions.

incidence $((o_1, \ldots, o_k), a) \in I$ – also written $a(o_1, \ldots, o_k)$ – is an ordered hyper-edge between nodes $o_1, \ldots, o_k$, labeled with a.

A *projected graph pattern of arity* k (k-PGP) is a pair $Q = (\overline{x}, P)$ where $\overline{x} = (x_1, \ldots, x_k) \in \mathcal{V}^k$ is a tuple of k *projected variables*, and $P \subseteq \mathcal{V}^* \times A$ is a *graph pattern*. Variables are here graph nodes, and pattern elements $((y_1, \ldots, y_k), a)$ – also written $a(y_1, \ldots, y_k)$ – are hyper-edges. The set of k-PGPs $\mathcal{Q}_k$ is equipped, for each arity k, with a subsumption operator $\subseteq_q$ and a similarity operator $\cap_q$. The *description* of a tuple of objects $\overline{o}$ is the PGP $Q(\overline{o}) := (\overline{o}, I)$, which uses objects as variables.

The *extension* of a k-PGP Q is defined as the set of k-tuples of objects whose description contains the PGP.

$$ext(Q) := \{\overline{o} \in O^k \mid Q \subseteq_q Q(\overline{o})\}, \quad \text{for every } Q \in \mathcal{Q}_k$$

The *intension* of a set of k-tuples of objects R is defined as the most specific PGP that subsumes the description of all tuples of objects in R, i.e. the PGP-intersection of all those descriptions.

$$int(R) := \bigcap_{\overline{o} \in R} Q(\overline{o}), \quad \text{for every } R \subseteq O^k$$

Those two derivation operators (ext, int) form a Galois connection between the two posets $(2^{O^k}, \subseteq)$ and $(\mathcal{Q}_k, \subseteq_q)$. A *graph concept* of a graph context $K = (O, A, I)$ is a pair (R, Q) with $R \subseteq O^k$ and $Q \in \mathcal{Q}_k$, for some arity k, such that $R = ext(Q)$ and $Q =_q int(R)$. The component R of a graph concept (R, Q) is called the *extent*, and the component Q is called the *intent*. Graph concepts are partially ordered by $(R_1, Q_1) \leq (R_2, Q_2) \Leftrightarrow R_1 \subseteq R_2 \Leftrightarrow Q_2 \subseteq_q Q_1$. This partial ordering forms a complete lattice called the *graph concept lattice*. There is a distinct lattice for each arity.

Graph-FCA has been applied to syntactic representations of texts, either for mining linguistic patterns [10] or for information extraction [2]. It has also been applied to knowledge graphs through the notion of *concepts of neighbors* for approximate query answering [9] and for link prediction [14].

Classical FCA is a special case of Graph-FCA when only 1-tuples (i.e., singletons) are used, for the incidence relation, the projected variables of PGPs, and hence for concepts. This implies that PGPs and concept intents have the shape $((x), (\{a_1(x), \ldots, a_p(x)\}))$, they use a single variable x, and hence they are equivalent to sets of attributes $\{a_1, \ldots, a_p\}$. Concept extents are then sets of singleton objects, which are equivalent to sets of objects.

3 Graph-PS: Extending Graph-FCA with Pattern Structures

For concision sake, we reuse the terms of Graph-FCA in Graph-PS as the graph structure remains, and only the description of nodes and edges are affected by the extension. In the following, we first define graph contexts as a common

generalization of Graph-FCA contexts and pattern structures. Then, we define Projected Graph Patterns (PGP) and operations on them as they play the role of concept intents. Finally, we define graph concepts, their formation through a Galois connection, and their organization into a lattice.

3.1 Graph Context

Definition 1 (graph context). *A* graph context *is a triple* $K = (O, (D, \sqcap), \delta)$*, where* O *is a set of* objects*,* D *is a meet-semi-lattice of* descriptions*, and* $\delta \in O^* \to D$ *is a mapping taking each* tuple of objects *to its description. The meet operator* $\sqcap$ *on descriptions entails a smallest description* $\bot$ *(called the* empty description*), and a partial ordering* $\sqsubseteq$ *(called* subsumption*), defined for all descriptions* $c, d \in D$ *as* $c \sqsubseteq d \iff c \sqcap d = c$*.*

Compared to Graph-FCA, each hyperedge is mapped to one description instead of to zero, one or several attributes. Graph-FCA is therefore equivalent to the special case of Graph-PS where descriptions are sets of attributes. An hyperedge that is mapped to the empty description is considered as a non-existent relationship: $\delta((o_1, o_2)) = \bot$ means that there is no relation from o_1 to o_2; in other words, the pair (o_1, o_2) is nothing more than a pair of objects. This can be paralleled with the blank cells ($(o, a) \notin I$) in a classical formal context.

Compared to PS, descriptions can not only be attached to objects but also to tuples of objects, which enables to express relationships between objects. Those relationships are taken into account, in addition to PS-like descriptions, when forming concepts. Concepts are sets of objects that have similar descriptions, and that also have similar relationships to objects that have similar descriptions, and so on.

Example 1. As an example of graph context K_{ex}, we extend an example from Graph-FCA about the British royal family, introducing taxonomic relationships between attributes, and numeric invervals. The objects are people belonging to three generations.

$$O_{ex} := \{Charles, Diana, William, Harry, Kate, George, Charlotte\}$$

They are respectively abbreviated as C, D, W, H, K, G, A. Objects (people) are described by a pair made of their gender and their birth year:

$$\begin{aligned} &\delta(C) = man : 1948, \quad && \delta(D) = woman : 1961, && \\ &\delta(W) = man : 1982, && \delta(K) = woman : 1982, && \delta(H) = man : 1984, \\ &\delta(G) = man : 2013, && \delta(A) = woman : 2015. && \end{aligned}$$

Pairs of objects are described whether the second is a parent of the first, and with the rank among siblings.

$$\begin{aligned} &\delta(W, C) = parent : 1 \quad && \delta(H, C) = parent : 2 \quad && \text{(same for D in place of C)} \\ &\delta(G, W) = parent : 1 && \delta(A, W) = parent : 2 && \text{(same for K in place of W)} \end{aligned}$$

For instance, $\delta(W, C) = parent : 1$ tells that William is the first child of Charles. Any other tuple of objects has the empty description: $\delta(\overline{o}) = \bot$.

We want to take into account similarities between attributes and values. First, we define the similarity between genders, $man \sqcap woman = person$, saying that men and women have in common to be persons. Second, like in [18], we define the similarity between two numeric values as the smallest interval that contains the two values: e.g., $1 \sqcap 3 = [1, 3]$. This extends to intervals by using the convex hull of two intervals, considering a value v as equivalent to the interval $[v, v]$.

$$[u_1, v_1] \sqcap [u_2, v_2] = [min(u_1, u_2), max(v_1, v_2)]$$

However, to avoid intervals that are too large and hence meaningless, intervals $[u, v]$ s.t. $v - u > \epsilon$ are generalized into the symbol $*$ that represents the range of all possible values. The threshold ϵ depends on the type of values: $\epsilon = 20$ for birth years so that similarity means "in the same generation", and $\epsilon = 1$ for birth ranks. To summarize, the set of descriptions in our example is defined as:

$$D_{ex} := \{a : [u, v], a : * \mid a \in \{person, man, woman, parent\}, u \leq v \in \mathbb{Z}\} \cup \{\bot\},$$

with $a : v$ as a shorthand for $a : [v, v]$, and $\bot$ the empty description. The similarity operator $d = d_1 \sqcap d_2$ is defined as follows. If $d_1 = \bot$ or $d_2 = \bot$ then $d = \bot$, otherwise, $d_1 = a_1 : V_1$ and $d_2 = a_2 : V_2$. Then, if the attribute similarity $a_1 \sqcap a_2$ is defined as attribute a, then $d = a : V$ where $V = V_1 \sqcap V_2$ as defined above with the ϵ threshold depending on the attribute, else $d = \bot$. □

3.2 Projected Graph Patterns (PGP)

A *graph pattern* over a graph context shares the same structure as a graph context, with nodes and hyperedges labelled by descriptions, except that nodes are variables that range over the objects of the context.

Definition 2 (graph pattern). *Let $K = (O, (D, \sqcap), \delta)$ be a graph context. A* graph pattern *over K is a pair $P = (V, \delta_P)$, where $V \subseteq \mathcal{V}$ is a finite set of variables* (nodes)*, and $\delta_P \in V^* \to D$ is a mapping taking each hyperedge to its description* (hyperedge label)*.*

Compared to Graph-FCA, hyperedges are labelled by custom descriptions rather than by sets of attributes. An *embedding* of a graph pattern in a graph context is a mapping $\phi \in V \to O$ from pattern variables to context objects such that for each hyperedge $\overline{x}$, the pattern description of the edge subsumes the context description of the corresponding edge, i.e. $\delta_P(\overline{x}) \sqsubseteq \delta(\phi(\overline{x}))$.

Example 2. Given the graph context in Example 1, we introduce the example graph pattern $P_{ex} = (\{x, y\}, \delta_P)$, where the description δ_P is defined as follows:

$$\delta_P(x) = man : [1980, 1989], \quad \delta_P(y) = person : *, \quad \delta_P(x, y) = parent : [1, 2].$$

This description can be more concisely written as follows:

$$\delta_P = \{x \mapsto man : [1980, 1989], y \mapsto person : *, (x, y) \mapsto parent : [1, 2]\}.$$

This pattern represents the situation where a man born in the eighties (x) is the first or second child of some person with unconstrainted birthdate (y). The pattern has four embeddings in the context, e.g. $\{x \mapsto Harry, y \mapsto Diana\}$ because Harry is a man born in 1984, and is the second child of Diana, who is a woman and hence a person. □

A *Projected Graph Pattern (PGP)* is a graph pattern with a tuple of distinguished variables, called *projected variables*.

Definition 3 (PGP). *A* projected graph pattern *(PGP) is a couple $Q = (\overline{x}, P)$ where $P = (V, \delta)$ is a graph pattern, and $\overline{x} \in V^*$, called* projection tuple*, is a tuple of variables from the pattern. $|Q| = |\overline{x}|$ denotes the* arity *of the PGP. We note $\mathcal{Q}$ the set of PGPs, and $\mathcal{Q}_k$ the subset of k-PGPs, i.e. PGPs having arity k.*

A PGP can be seen as a SPARQL query `SELECT` $\overline{x}$ `FROM { P }`, whose answers are the embeddings of the pattern restricted to the projected variables.

Example 3. The PGP $Q_{ex} = ((x), P_{ex})$ based on the graph pattern in Example 2 selects all men born in the eighties as the first or second child of somebody. The answers over the example context are therefore Harry and William. The PGP $((x, y), P_{ex})$ would select pairs *(child, parent)*, such as (Harry, Diana). □

In Graph-FCA and Graph-PS, PGPs play the role of descriptions in PS. We therefore have to define two key operations on them: inclusion $\subseteq_q$ (aka. subsumption) and intersection $\cap_q$ (aka. similarity). In Graph-PS, their definitions depend on the corresponding operations on PS-like descriptions, $\sqsubseteq$ and $\sqcap$.

Definition 4 (PGP inclusion). *Let $K = (O, (D, \sqcap), \delta)$ be a graph context. Let $Q_1 = (\overline{x}_1, (V_1, \delta_1))$, $Q_2 = (\overline{x}_2, (V_2, \delta_2))$ be two k-PGPs for some arity k. Q_1 is* included in Q_2*, or equivalently Q_2* contains Q_1*, which is written $Q_1 \subseteq_q Q_2$ iff*

$$\exists \phi \in V_1 \to V_2 : \phi(\overline{x}_1) = \overline{x}_2 \wedge \forall \overline{y} \in V_1^* : \delta_1(\overline{y}) \sqsubseteq \delta_2(\phi(\overline{y}))$$

According to this definition, the inclusion of Q_1 into Q_2 is analogous to the embedding of a pattern into a context, with the difference that variables are mapped to the variables of another pattern instead of the objects of the context. There is also the additional constraint that the projected variables match.

Example 4. For example, the PGP $Q' = ((z), (\{z\}, \{z \mapsto man : *\}))$, which selects the set of men, is included in the above PGP Q_{ex}, through the embedding $\phi = \{z \mapsto x\}$. □

Definition 5 (PGP intersection). *Let ψ be an injective mapping from pairs of variables to fresh variables. The intersection of two k-PGPs $Q_1 = (\overline{x}_1, (V_1, \delta_1))$ and $Q_2 = (\overline{x}_2, (V_2, \delta_2))$, written $Q_1 \cap_q Q_2$, is defined as $Q = (\overline{x}, (V, \delta))$, where*

$$\begin{aligned}
&\overline{x} = \psi(\overline{x}_1, \overline{x}_2),\\
&V = \{\psi(v_1, v_2) \mid v_1 \in V_1, v_2 \in V_2\},\\
&\delta(\overline{y}) = \delta_1(\overline{y}_1) \sqcap \delta_2(\overline{y}_2), \text{ for } \overline{y} = \psi(\overline{y}_1, \overline{y}_2) \in V^*
\end{aligned}$$

PGP intersection works as a product of two PGPs where each pair of edges $(\overline{y}_1, \overline{y}_2)$ makes an edge whose description is the similarity $\delta_1(\overline{y}_1) \sqcap \delta_2(\overline{y}_2)$ between the descriptions of the two edges.

Example 5. The intersection of Q_{ex} and Q' results in the PGP

$$Q'' = ((xz), (\{xz, yz\}, \{xz \mapsto man : *, yz \mapsto person : *\})).$$

Variables xz and yz result from the pairing of variables from each PGP (function ψ). The tuples of variables that are not shown in the δ_P part have the empty description. For instance, $\delta_P((xz, yz)) = \delta_{ex}((x, y)) \sqcap \delta'((z, z)) = parent : [1, 2] \sqcap \bot = \bot$.

In Q'' the description of yz is disconnected from the projected variable xz, and is therefore useless to the semantics of Q''. Q'' can therefore be simplified to $((xz), (\{xz\}, \{xz \mapsto man : *\}))$, which is equal to Q' up to renaming variable xz as x. More information about such simplifications are available in [11]. □

The above example suggests as expected that $Q_1 \subseteq_q Q_2$ implies $Q_1 \cap_q Q_2 = Q_1$. The following lemma proves that this is indeed the case, like with PS descriptions.

Lemma 1. *Let Q_1, Q_2 be two PGPs. Their PGP intersection $Q_1 \cap_q Q_2$ is their* infimum *relative to query inclusion $\subseteq_q$.*

Proof. To prove that $Q = Q_1 \cap_q Q_2$ is a lower bound, it suffices to prove that Q is included in both Q_1 and Q_2. To prove $Q \subseteq_q Q_1$, it suffices to choose the mapping $\phi_1(x) = (\psi^{-1}(x))[1]$ (recall that ψ is an injective mapping from 2-tuples of variables to variables), and to prove that $\phi_1(\overline{x}) = \overline{x}_1$ and $\delta(\overline{y}) \sqsubseteq \delta_1(\phi_1(\overline{y}))$ for all $\overline{y} \in V^*$. This is easily obtained from the definition of Q. The proof of $Q \subseteq_q Q_2$ is identical with $\phi_2(x) = (\psi^{-1}(x))[2]$.

To prove that $Q_1 \cap_q Q_2$ is the *greatest* lower bound (the infimum), we have to prove that every PGP Q' that is included in both Q_1 (via ϕ_1) and Q_2 (via ϕ_2) is also included in Q. To that purpose, it suffices to choose $\phi(x') = \psi(\phi_1(x'), \phi_2(x'))$, and to prove that $\phi(\overline{x}') = \overline{x}$ and $\delta'(\overline{y}') \sqsubseteq \delta(\phi(\overline{y}'))$ for all $\overline{y}'$. This can be obtained from the definition of Q, and from the hypotheses. □

3.3 Graph Concepts

As usual in FCA, concepts are composed of an extent and an intent. In Graph-PS like in Graph-FCA, k-PGPs in $\mathcal{Q}_k$ play the role of intents. For the extents we use the answers of PGPs seen as queries, i.e. sets of tuples of objects. The latter are mathematically k-ary *relations* over objects: $R \subseteq O^k$, for some arity $k \geq 0$. We note $\mathcal{R}_k = 2^{O^k}$ the set of k-relations over the objects of some graph context K.

Example 6. The set of father-mother-child triples can be represented as the following 3-relation (with abbreviated people names).

$$R := \{(C, D, W), (C, D, H), (W, K, G), (W, K, A)\}$$

Note that the order of objects in tuples matters while the order of tuples in the relation does not. A k-relation can be seen as a table with k unlabeled columns.

Charles	Diana	William
Charles	Diana	Harry
William	Kate	George
William	Kate	Charlotte

□

Before defining the Galois connection between PGPs and relations, we introduce the notion of *graph description* $\gamma(\overline{o})$ of an object or a tuple of objects. It incorporates everything that is known about an object or tuple of objects, in terms of relationships in the graph context around those objects, and in terms of D-description of those relationships. It therefore integrates the description δ of individual hyperedges.

Definition 6 (graph description). *Given a graph context* $K = (O, (D, \sqcap), \delta)$*, the* graph description *of any object* $o \in O$ *is defined as the PGP* $\gamma(o) := ((o), P_K)$ *where* $P_K = (O, \delta)$*. By extension, the description of any tuple of objects* $\overline{o} \in O^*$ *is defined as* $\gamma(\overline{o}) := (\overline{o}, P_K)$*.*

This definition says that the graph description of an object is the whole graph context, seen as a graph pattern (objects as variables), and projected on the object. In practice, only the part of the graph context that is connected to the object is relevant. The generalization to tuples of objects enables to have a description for pairs of objects, triples of objects, and so on.

From there, we can define two derivation operators between PGPs and relations, and prove that they form a Galois connection.

Definition 7 (extension). *Let* $K = (O, (D, \sqcap), \delta)$ *be a graph context. The* extension *of a* k*-PGP* $Q \in \mathcal{Q}_k$ *is the* k*-relation defined by*

$$ext(Q) := \{\overline{o} \in O^k \mid Q \subseteq_q \gamma(\overline{o})\}$$

The extension of a k-PGP is the set of k-tuples of objects whose graph description contains the PGP. It can be understood as the set of answers of the PGP seen as a query.

Example 7. In the example context, the extension of the 2-PGP

$$Q = ((x, y), (\{x, y\}, \{x \mapsto person : *, y \mapsto woman : *, (x, y) \mapsto parent : 1\}))$$

is the 2-relation

$$R = ext(Q) = \{(William, Diana), (George, Kate)\},$$

i.e. the set of pairs *(first child, mother).* □

Definition 8 (intension). *Let* $K = (O, A, I)$ *be a graph context. The* intension *of a* k*-relation* $R \in \mathcal{R}_k$ *is the* k*-PGP defined by*

$$int(R) := \bigsqcap_{\overline{o} \in R} \gamma(\overline{o})$$

The intension of a k-relation is the PGP intersection of the graph descriptions of all tuples of objects in the relation, hence the most specific projected graph pattern shared by them.

Example 8. In the example context, the intension of the 2-relation from Example 7

$$R = \{(William, Diana), (George, Kate)\}$$

is the 2-PGP

$$Q = int(R) = ((x, y), (\{x, y, z, w\}, \delta_P))$$

where $\delta_p = \{x \mapsto man : *,\ y \mapsto woman : *,\ z \mapsto man : *,\ w \mapsto person : *,$ $(x, y) \mapsto parent : 1,\ (x, z) \mapsto parent : 1,\ (w, y) \mapsto parent : 2, (w, z) \mapsto parent : 2\}$. Note that this intension expands the PGP in Example 7 with the following elements: x is a man, x is the first child of some man z, his father, and there is a second child w of parents y and z. The extension of this expanded PGP remains the relation R, which suggests that $int \circ ext$ is a closure operator. □

We can actually prove that *ext* and *int* form a Galois connection. This implies that $int \circ ext$ and $ext \circ int$ are closure operators, respectively on PGPs and relations.

Theorem 1 (Galois connection). *Let $K = (O, (D, \sqcap), \delta)$ be a graph context. For every arity k, the pair of mappings (ext, int) forms a Galois connection between $(\mathcal{R}_k, \subseteq)$ and $(\mathcal{Q}_k, \subseteq_q)$, i.e. for every object relation $R \in \mathcal{R}_k$ and PGP $Q \in \mathcal{Q}_k$,*

$$R \subseteq ext(Q) \iff Q \subseteq_q int(R)$$

Proof. $R \subseteq ext(Q) \iff \forall \overline{o} \in R : \overline{o} \in ext(Q)$
$\iff \forall \overline{o} \in R : Q \subseteq_q \gamma(\overline{o})$ (Definition 7)
$\iff Q \subseteq_q \bigcap_{\overline{o} \in R} \gamma(\overline{o})$ (Lemma 1)
$\iff Q \subseteq_q int(R)$ (Definition 8) □

From the Galois connection, *graph concepts* can be defined and organized into concept lattices, like in classical FCA, with one concept lattice for each arity k.

Definition 9 (graph concept). *Let $K = (O, (D, \sqcap), \delta)$ be a graph context. A* k-graph concept *of K is a pair (R, Q), made of a k-relation (the* extent*) and a k-PGP (the* intent*), such that $R = ext(Q)$ and $Q =_q int(R)$.*

Example 9. The 2-relation and 2-PGP in Example 8 form a 2-graph concept. It can be understood as the *(first child, mother)* binary relationship. Its intent tells us that in the example context, every first child whose mother is known also has a known father, and a sibling (man or woman) that was born after him. □

Theorem 2 (graph concept lattices). *The set of graph k-concepts $\mathcal{C}_k$, partially ordered by $\leq$, which is defined by*

$$(R_1, Q_1) \leq (R_2, Q_2) : \iff R_1 \subseteq R_2 \iff Q_2 \subseteq_q Q_1,$$

forms a bounded lattice $(\mathcal{C}_k, \leq, \wedge, \vee, \top, \bot)$, the k-graph concept lattice.

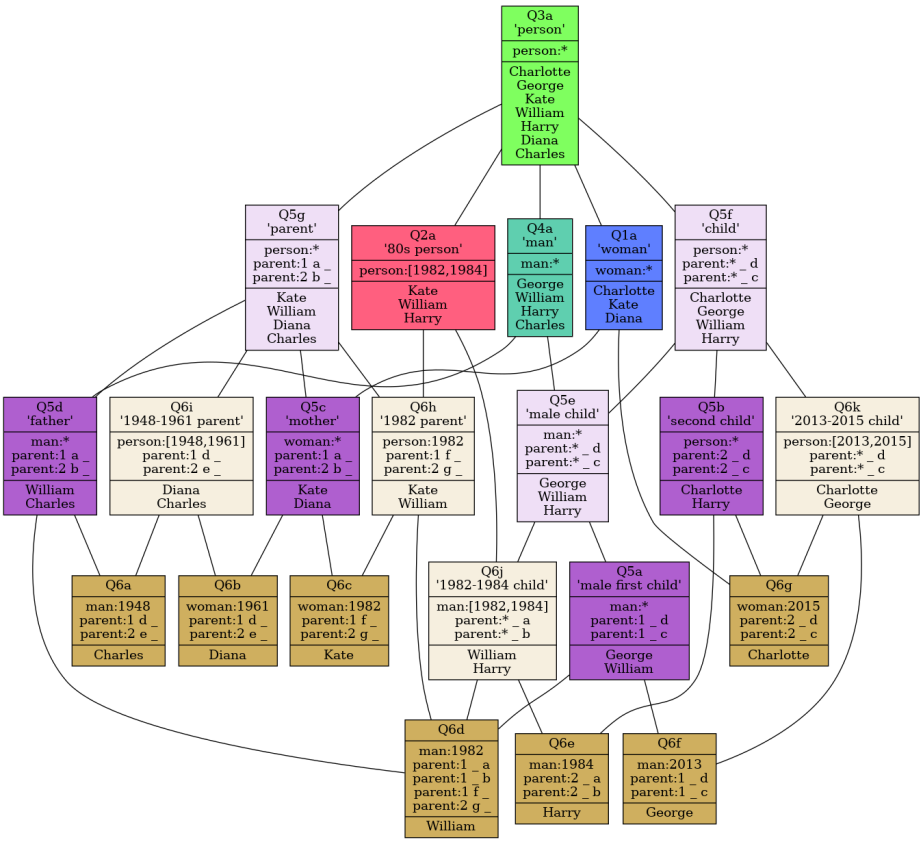

Fig. 1. The 1-graph concept lattice of the royal family context (less the bottom concept). The descriptor `parent:1 a _` in box Q5g reads $\delta_{P5}(a, g) = parent:1$, as part of the graph pattern $P5$ of the intent of concept Q5g. The nodes of pattern $P5$ are the boxes Q5a-g.

Figure 1 is a representation of the 1-graph concept lattice – less the bottom concept – of the royal family graph context from Example 1. This is a compact representation because each box represents at the same time a 1-concept and a node of the graph pattern of a concept intent. Each box is made of three parts:

1. A concept/node identifier Qnx made of a number n and a letter x. The number identifies a pattern P_n, and the letter identifies a node of this pattern. Together they form the concept intent $Q_n = (x, P_n)$. The set of nodes V_n of pattern $P_n = (V_n, \delta_{P_n})$ is therefore made of all boxes whose identifier has Qn as a prefix. (Some concepts also have an informal description under the identifier, manually added to help the reading of the lattice.)
2. A list of pattern hyperedges in the form $d\ y_1\ \ldots\ y_n$, to be read as $\delta_{P_n}(y_1, \ldots, y_n) = d$, where an underscore `_` refers to the current node x.

A 1-edge d _ is abbreviated as d as it corresponds to an attribute in classical FCA.
3. A list of objects composing the concept extent.

A k-concept with $k > 1$ can be read by selecting k different boxes that belong to the same pattern. The graph pattern is the same, whatever the nodes chosen as projected variables, so the concept intent can be read like for 1-concepts. However, the concept extent cannot be read exactly from this representation, it is a subset of the Cartesian product of the extents of the selected boxes.

Example 10. Given the way different concepts share the same graph pattern in their intent, it makes sense to interpret the concept lattice in Figure 1 pattern by pattern.

- Pattern P_3 is a about a person born at any time. It has a single node a, hence there is a single concept Q3a, the concept of all persons in the context. It is the top concept.
- Pattern P_1 is the refinement of P_3 on women.
- Pattern P_4 is the refinement of P_3 on men.
- Pattern P_2 is the refinement of P_3 on people born in the eighties.
- Pattern P_5 has 7 nodes (a-g). It is about a man (a) that is the first child of two parents, a man (d) and a woman (c), and about another person (b) that is the second child of the same parents. Those nodes respectively correspond to the concepts of "male first child" (Q5a), "father" (Q5d), "mother" (Q5c), and "second child" (Q5b). The other nodes in the pattern actually define generalizations of those concepts for which the a-b-c-d pattern is always present in the context: "parent" (Q5g), "male child" (Q5e), and "child" (Q5f). It can be observed that the latter concepts (e-g) are in a lighter color and the former concepts (a-d) are in a more vivid color (called *core concepts/nodes* [11]). When reading a concept intent, the boxes in a lighter color can be ignored when they are not selected because the information they provide is redundant with the core nodes.
- The core nodes of pattern P_6 reproduce the graph context, each node corresponding to a specific object. The non-core concepts provide generalizations over single objects: "grand-parents born between 1948 and 1961" (Q6i), "parents born in 1982" (Q6h), "children born between 1982 and 1984" (Q6j), and "grand-children born between 2013 and 2015" (Q6k).

From the concept lattice structure, it is possible to find what different people have in common. For instance, from $Q6d \vee Q6g = Q5f$, we learn that what William (Q6d) and Charlotte (Q6g) have in common is that they are persons with a father (Q5d, Charles or William) and a mother (Q5c, Diana or Kate) who have together a male first child (Q5a, William or George), and a second child (Q5b, Harry or Charlotte). We also learn that they share that with Harry and George, the other instances of concept Q5f. □

4 Application to RDFS Graphs

As an application case of Graph-PS we consider RDFS graphs. An RDFS graph is a structure $\langle R, \mathcal{L}, (\mathcal{C}, \leq), (\mathcal{P}, \leq), T \rangle$, where R is a collection of resources (IRIs and blank nodes), $\mathcal{L}$ is a set of literal values of various datatypes (e.g., strings, numbers, dates), $\mathcal{C}$ is a hierarchy of classes, $\mathcal{P}$ is a hierarchy of properties, and T is a set of triples expressing the factual knowledge.

In order to apply Graph-PS to RDFS graphs, we need to identify what are the objects, the descriptions, and the similarity between descriptions. From the usual RDFS graph representation that uses resources, literals, and classes as nodes, it is tempting to use them as objects. However, it is desirable to define similarity over literals and classes. The similarity between two integer literals could be an interval like in the example of the previous section. The similarity between two classes should be the most specific common ancestor class. Moreover, we think that literals and classes are more appropriate as descriptors of objects than as objects to be described. We therefore define the set of object as $O = R$.

We now look at the description of (tuples of) objects. The description information lies in the triples. We define below their conversion into elementary descriptions, according to the three kinds of triples.

- $(r, \texttt{rdf:type}, c) \rightsquigarrow \delta(r) = \{c, \ldots\}$
  The triple states that resource r is an instance of class c. The class is used as a descriptor of the resource, `rdf:type` can be ignored because it is always used with a class. We use an open set containing c because a resource can be declared an instance of several classes.
- $(r, p, r') \rightsquigarrow \delta(r, r') = \{p, \ldots\}$
  The triple states that resource r is related to resource r' with property p. The property is used as a descriptor of the pair of resources, hence representing a binary edge. We again use an open set because RDFS graphs are multigraphs, i.e. several properties can relate the same resources (e.g., a person who is both the director and an actor of some film).
- $(r, p, l) \rightsquigarrow \delta(r) = \{p : l, \ldots\}$
  The triple states that resource r is related to literal l with property p. Both the property and literal are descriptors, so they must be combined into a composite descriptor, similarly to the example of previous section. We again use an open set because RDFS properties can be multi-valued, and also because resources also have classes as descriptors.

To summarize, the set of descriptions can be defined as follows.

$$D = D_1 \cup D_2 \quad \text{where} \quad D_1 = 2^{\mathcal{C}} \times 2^{\mathcal{P} \times \mathcal{L}} \quad \text{and} \quad D_2 = 2^{\mathcal{P}}$$

D_1 is the set of descriptions of individuals resources, where a description is a pair made of a set of classes, and a set of valued properties. D_2 is the set of descriptions of edges between resources, where a description is a set of properties. The empty description is therefore simply the empty set: $\bot = \emptyset$. From there, we

can formally define the description of every resources and pairs of resources.

$$\delta(r) = (\{c \mid (r, \texttt{rdf:type}, c) \in T\}, \{p : l \mid (r, p, l) \in T,\ l \in \mathcal{L}\})$$
$$\delta(r, r') = \{p \mid (r, p, r') \in T,\ r' \in R\}$$

It remains to define the similarity operator $\sqcap$ over descriptions. As our descriptions of RDFS resources are based on sets of elementary descriptors, we derive similarity on sets from similarity on elements. We can allow the similarity between two elements to be a set of elements, the *least general generalizations (lgg)*. This set-based approach has already been used in Pattern Structures, e.g. for graphs and subgraphs [16]. On classes and properties, we have a partial ordering $\leq$ from which the lgg operator can defined as follows:

$$lgg(x, y) := Min_{\leq}\{z \in X \mid x \leq z, y \leq z\}$$

On literals, given that $\mathcal{L}$ is in general infinite, it is more convenient to assume the lgg operation to be defined, and to derive the partial ordering from it: $x \leq y \iff lgg(x, y) = \{y\}$. Here are a few examples on how lgg could be defined on literals:

- $lgg(10, 20) = \{[10, 20]\}$,
- $lgg([10, 30], [25, 50]) = \{[10, 50]\}$,
- $lgg(\text{"Formal Concept Analysis"}, \text{"Relational Concept Analysis"}) = \{\text{"Concept"}, \text{"Analysis"}\}$.

Of course, this assumes to extend the set of literals $\mathcal{L}$ with all patterns that may be generated by the lgg operator, e.g. intervals. On valued properties the lgg operator can be obtained by combining the lgg operators on properties and literals.

$$lgg(p' : l', p'' : l'') = \{p : l \mid p' \in lgg(p', p''), l \in lgg(l', l'')\}$$

Each lgg operator can be lifted to the PS similarity operator on sets of elements by collecting all least general generalizations of elements pairwise, and then filtering them to keep only the most specific ones.

$$d' \sqcap d'' = Min_{\leq}\{x \in lgg(x', x'') \mid x' \in d', y' \in d''\}$$

This is enough to define similarity on D_2-descriptions, which are sets of properties. On D_1-descriptions, similarity can be defined element-wise as they are pairs (C, PL) of sets of elementary descriptors: C is a set of classes, and PL is a set of valued properties.

$$(C', PL') \sqcap (C'', PL'') = (C' \sqcap C'', PL' \sqcap PL'')$$

Finally, the similarity between a D_1-description and a D_2-description is simply the empty description $\bot$, although Graph-PS only applies similarity to the descriptions of tuples of objects with the same arity.

5 Conclusion and Perspectives

We have introduced a new extension of Formal Concept Analysis that merges two existing FCA extensions, Pattern Structures (PS) and Graph-FCA. In short, PS-like descriptions are used to describe the nodes and hyperedges of graphs, in place of sets of attributes. The new extension therefore combines the benefits of the two existing extensions: complex descriptions and relationships between objects. A strength of Graph-PS is that it is a proper generalization of PS and Graph-FCA, in the sense that PS and Graph-FCA – as well as FCA – are special cases of Graph-PS. Hence, all previous work about defining custom pattern structures can be reused in Graph-PS, and the compact graphical representations of concept lattices in Graph-FCA can be reused in Graph-PS. We have also shown that Graph-PS can accurately represent existing graph-based models like RDFS graphs.

This paper focuses on the theoretical aspects of Graph-PS, and the most immediate perspectives concern its implementation and its applications. The implementation could be adapted from the existing implementation of Graph-FCA [7], by taking into account the similarity operator $\sqcap$ in the PGP operations $\subseteq_q$ and $\cap_q$. The additional cost of using Graph-PS in PS and Graph-FCA settings should be evaluated. A toolbox of components should be built in order to facilitate the design of new sets of descriptions, by capitalizing on previous applications of pattern structures, and by adopting the methodology of logic functors [13]. In the end, we plan to experiment Graph-PS in diverse knowledge graphs and other complex structures like sequences and trees.

References

1. Alam, M., Buzmakov, A., Napoli, A.: Exploratory knowledge discovery over web of data. Discrete Applied Mathematics **249**, 2–17 (2018)
2. Ayats, H., Cellier, P., Ferré, S.: Extracting relations in texts with concepts of neighbours. In: Formal Concept Analysis. pp. 155–171. LNCS 12733, Springer (2021)
3. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
4. Baixeries, J., Kaytoue, M., Napoli, A.: Characterizing functional dependencies in formal concept analysis with pattern structures. Annals of Mathematics and Artificial Intelligence **72**, 129–149 (2014)
5. Belohlavek, R.: Fuzzy closure operators. Journal of Mathematical Analysis and Appl. **262**, 473–489 (2001)
6. Chaudron, L., Maille, N.: Generalized formal concept analysis. In: Mineau, G., Ganter, B. (eds.) Int. Conf. Conceptual Structures. LNCS 1867, Springer (2000)
7. Ferré, S., Cellier, P.: Modeling complex structures in Graph-FCA: Illustration on natural language syntax. In: Existing Tools and Applications for Formal Concept Analysis (ETAFCA). pp. 1–6 (2022)
8. Ferré, S.: A proposal for extending formal concept analysis to knowledge graphs. In: Baixeries, J., Sacarea, C., Ojeda-Aciego, M. (eds.) Int. Conf. Formal Concept Analysis (ICFCA). pp. 271–286. LNCS 9113, Springer (2015)

9. Ferré, S.: Answers partitioning and lazy joins for efficient query relaxation and application to similarity search. In: Gangemi, A., et al. (eds.) The Semantic Web (ESWC). pp. 209–224. LNCS 10843, Springer (2018)
10. Ferré, S., Cellier, P.: Graph-FCA in practice. In: Haemmerlé, O., et al. (eds.) Int. Conf. Conceptual Structures (ICCS). pp. 107–121. LNCS 9717, Springer (2016)
11. Ferré, S., Cellier, P.: Graph-FCA: An extension of formal concept analysis to knowledge graphs. Discrete Applied Mathematics **273**, 81–102 (2019)
12. Ferré, S., Ridoux, O.: A logical generalization of formal concept analysis. In: Mineau, G., Ganter, B. (eds.) Int. Conf. Conceptual Structures. pp. 371–384. LNCS 1867, Springer (2000)
13. Ferré, S., Ridoux, O.: A framework for developing embeddable customized logics. In: Pettorossi, A. (ed.) Int. Work. Logic-based Program Synthesis and Transformation. pp. 191–215. LNCS 2372, Springer (2002)
14. Ferré, S.: Application of concepts of neighbours to knowledge graph completion. Data Science: Methods, Infrastructure, and Applications **4**, 1–28 (2021)
15. Ferré, S., Kaytoue, M., Huchard, M., Kuznetsov, S.O., Napoli, A.: A guided tour of artificial intelligence research, vol. II, chap. Formal Concept Analysis: from knowledge discovery to knowledge processing (Chapter 13), pp. 411–445. Springer (2020)
16. Ganter, B., Kuznetsov, S.: Pattern structures and their projections. In: Delugach, H.S., Stumme, G. (eds.) Int. Conf. Conceptual Structures. pp. 129–142. LNCS 2120, Springer (2001)
17. Ganter, B., Wille, R.: Formal Concept Analysis — Mathematical Foundations. Springer (1999)
18. Kaytoue, M., Kuznetsov, S.O., Napoli, A.: Revisiting numerical pattern mining with formal concept analysis. In: Int. Joint Conf. Artificial Intelligence (IJCAI) (2011)
19. Kötters, J.: Concept lattices of a relational structure. In: Pfeiffer, H., and others (eds.) Int. Conf. Conceptual Structures for STEM Research and Education, pp. 301–310. LNAI 7735, Springer (2013)
20. Kuznetsov, S.O., Samokhin, M.V.: Learning closed sets of labeled graphs for chemical applications. In: Kramer, S., Pfahringer, B. (eds.) Int. Conf. Inductive Logic Programming. pp. 190–208. LNCS 3625, Springer (2005)
21. Lehmann, F., Wille, R.: A triadic approach to formal concept analysis. In: Int. Conf. Conceptual Structures (ICCS). pp. 32–43. Springer (1995)
22. Poelmans, J., Ignatov, D.I., Kuznetsov, S.O., Dedene, G.: Fuzzy and rough formal concept analysis: a survey. Int. J. General Systems **43**(2), 105–134 (2014)
23. Rouane-Hacene, M., Huchard, M., Napoli, A., Valtchev, P.: Relational concept analysis: mining concept lattices from multi-relational data. Annals of Mathematics and Artificial Intelligence **67**(1), 81–108 (2013)
24. Voutsadakis, G.: Polyadic concept analysis. Order **19**, 295–304 (2002)