Graph-FCA in Practice

Sébastien Ferré, Peggy Cellier

Graph-FCA in Practice

Sébastien Ferré* and Peggy Cellier***

*IRISA/Université de Rennes 1
**IRISA/INSA Rennes
Campus de Beaulieu, 35042 Rennes cedex, France
ferre@irisa.fr, cellier@irisa.fr

Abstract. With the rise of the Semantic Web, more and more relational data are made available in the form of knowledge graphs (e.g., RDF, conceptual graphs). A challenge is to discover conceptual structures in those graphs, in the same way as Formal Concept Analysis (FCA) discovers conceptual structures in tables. Graph-FCA has been introduced in a previous work as an extension of FCA for such knowledge graphs. In this paper, algorithmic aspects and use cases are explored in order to study the feasibility and usefulness of G-FCA. We consider two use cases. The first one extracts linguistic structures from parse trees, comparing two graph models. The second one extracts workflow patterns from cooking recipes, highlighting the benefits of n-ary relationships and concepts.

Keywords: Formal Concept Analysis, Knowledge Graph, Semantic Web, Graph Pattern

1 Introduction

With the rise of the Semantic Web, more and more data are made available in the form of RDF graphs. More generally, *knowledge graphs* (e.g., RDF graphs, Conceptual Graphs) allow for the representation of complex information as a set of entities interlinked with binary, and possibly n-ary, relationships. A challenge is to discover conceptual structures in knowledge graphs, in the same way as Formal Concept Analysis (FCA) discovers conceptual structures in tables [6]. Relational Concept Analysis (RCA) [11], an extension of FCA, shows interesting results on relational data, in particular in software engineering [8].

Recently, Graph-FCA (G-FCA) [3] has been introduced as another extension of FCA for knowledge graphs. The specifity of G-FCA is to extract n-ary concepts from a knowledge graph using n-ary relationships. The intents of n-ary concepts are graph patterns with a focus on one or several nodes, i.e. *Projected Graph Patterns* (PGP). In practice, it allows to discover n-ary relational concepts. For instance, in a knowledge graph that represents family members with a "parent" relation, the "sibling" binary concept can be discovered, and described by a PGP as "a pair of persons having a common father and a common mother".

* This research is supported by ANR project IDFRAud (ANR-14-CE28-0012-02).

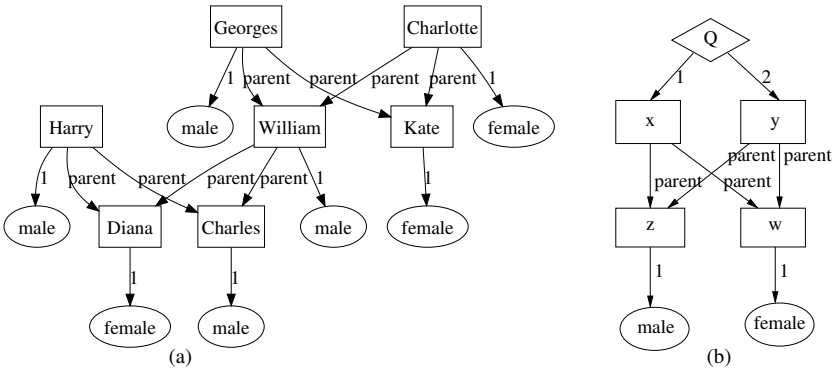

Fig. 1. (a) Graph context about the British royal family. Rectangles are objects, word-labelled links are binary edges, and ellipses are other edges. (b) PGP defining the "sibling" binary relation. Rectangles are variables, diamonds are projection tuples.

In this paper, algorithmic aspects of G-FCA and use cases are explored in order to study the feasibility and usefulness of G-FCA. We address the problem of an efficient generation strategy of concepts in order to avoid as much as possible duplication in computation and in the presentation of results. We have also conducted experiments on two use cases. The first one is the exploration of linguistic structures in parse trees, comparing two graph modellings. The second one is about the extraction of workflow patterns from cooking recipes

In the following, Section 2 recalls the main definitions of G-FCA. Section 3 discusses related work. Section 4 presents an algorithm to extract concepts from a graph context. Section 5 shows two use cases.

2 Graph-FCA

Graph Contexts. We here recall the main definitions and theoretical results of Graph-FCA (G-FCA) [3], and illustrate them with an example about the British royal family. Whereas FCA defines a formal context as an incidence relation between objects and attributes, G-FCA defines a *graph context* as an incidence relation between *tuples of* objects and attributes.

Definition 1 (graph context). *A* graph context *is 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 object tuples (*$\overline{o} \in O^*$*)*[1] *and attributes (*$a \in A$*).*

The graphical representation of a graph context uses objects as nodes, incidence elements as hyper-edges, and attributes as hyper-edge labels. Note that attributes can be interpreted as n-ary predicates, and graph contexts as First Order Logic (FOL) models (without functions and constants). Different kinds

[1] Empty tuples are covered for the sake of generality but are not used in this paper.

of knowledge graphs, such as conceptual graphs, RDF graphs, and RCA contexts, can all be mapped easily to a graph context. Figure 1.(a) shows the graphical representation of a small graph context about the British royal family. The objects are the royal family members (e.g., Harry, Georges). They are represented as rectangles. The attributes are a binary relation "parent" and two unary relations "male" and "female". The edges ((Harry,Charles),parent), ((Georges,Kate),parent), and ((Harry),male) belong to the incidence relation. More generally, a binary edge $((x, y), a)$ is represented by an edge from x to y labelled by a. Other edges $((x_1, \ldots, x_n), a)$ are represented by ellipses labelled by a, and having an edge labelled i to each node x_i.

Graph Patterns and Projections. *Graph patterns* generalize the incidence relation of a graph context by taking its nodes in an infinite set of variables $x, y, \ldots \in \mathcal{V}$. Therefore, a graph context can be seen as a graph pattern by abstracting its objects as variables. A *Projected* Graph Patterns (PGP) is a graph pattern with a tuple of distinguished variables.

Definition 2 (graph pattern and PGP). *A* graph pattern $P \subseteq \mathcal{V}^* \times A$ *is a set of directed hyper-edges with variables as nodes, and attributes as labels.*

A projected graph pattern *(PGP) is a couple* $Q = (\overline{x}, P)$ *where* P *is a graph pattern, and* $\overline{x} \in \mathcal{V}^*$ *is called the* projection tuple*. The* arity *of a PGP is the length of* $\overline{x}$*. We note* $\mathcal{Q}_k$ *the set of PGPs having arity* k*.*

A key aspect of G-FCA is that closure does not apply directly to graph patterns but to PGPs. PGPs are analogous to anonymous definitions of FOL predicates, and to SPARQL queries. They play the same role as sets of attributes in FCA, i.e. as concept intents. Figure 1.(b) shows a PGP defining the "sibling" binary relation as two persons sharing a male parent (father) and a female parent (mother). The projection tuple is (x, y). Its graphical representation is a diamond node pointing to each projected variable.

Set operations are extended from sets of attributes to PGPs. PGP inclusion $\subseteq_q$ is based on graph homomorphisms [7]. It is similar to the notion of *subsumption* on queries [2] or rules [10]. PGP intersection $\cap_q$ is defined as a form of graph alignment, where each pair of variables from the two patterns becomes a variable of the intersection pattern. It corresponds to the *categorical product* of graphs (see [7], p. 116).

Definition 3 (PGP inclusion). *A k-PGP* $Q_1 = (\overline{x_1}, P_1)$ *is included in a k-PGP* $Q_2 = (\overline{x_2}, P_2)$*, denoted by* $Q_1 \subseteq_q Q_2$*, iff there exists a mapping* ϕ *from* P_1*-nodes to* P_2*-nodes s.t.* $\overline{\phi(x_1)} = \overline{x_2}$*, and for every edge* $(\overline{y}, a) \in P_1$*,* $(\overline{\phi(y)}, a) \in P_2$*. Therefore,* ϕ *is an homomorphism from* P_1 *to* P_2 *that preserves the projection tuple. When* $Q_1 \subseteq_q Q_2$ *and* $Q_2 \subseteq_q Q_1$*, they are said* equivalent *(*$Q_1 \equiv_q Q_2$*).*

Definition 4 (PGP intersection). *Let* ψ *be an injective mapping from pairs of variables to variables. The intersection two k-PGPs* $Q_1 = (\overline{x_1}, P_1)$ *and* $Q_2 = (\overline{x_2}, P_2)$*, denoted by* $Q_1 \cap_q Q_2$*, is defined as* $Q = (\overline{x}, P)$*, where* $\overline{x} = \overline{\psi(x_1, x_2)}$*, and* $P = \{(\overline{\psi(y_1, y_2)}, a) \mid a \in A, (\overline{y_1}, a) \in P_1, (\overline{y_2}, a) \in P_2, |\overline{y_1}| = |\overline{y_2}|\}$*.*

Object Relations. FCA sets of objects are extended to *object relations*, i.e. sets of tuples of objects $R \subseteq O^k$, for some arity k. We note $\mathcal{R}_k$ the set of relations with arity k. For instance, $\{(\mathit{Charles}, \mathit{William}), (\mathit{Charles}, \mathit{Harry}), (\mathit{William}, \mathit{Georges})\}$ is an object relation with arity 2. Object relations are analogous to query answers in SPARQL. Object relations form a powerset lattice for each arity.

Galois Connection and Graph Concepts. Based on previous definitions, the following Galois connection can be defined and proved between PGPs and object relations (see [3] for the proof). The connection from PGP to object relation is analogous to query evaluation, and the connection from object relations to PGP to relational learning [10]. In the definitions of Q' and R' below, the PGP $(\overline{o}, I) \in \mathcal{Q}$ represents the description of an object tuple $\overline{o}$ by the relative position of objects in the whole incidence relation I.

Theorem 1 (Galois connection). *Let $K = (O, A, I)$ be a graph context. For every arity k, the following pair of mappings between PGPs $Q \in \mathcal{Q}_k$ and object relations $R \in \mathcal{R}_k$ forms a Galois connection.*

$$Q' := \{\overline{o} \in O^k \mid Q \subseteq_q (\overline{o}, I)\} \qquad\qquad R' := \cap_q \{(\overline{o}, I)\}_{\overline{o} \in R}$$

From there, concepts can be defined in the usual way, and proved to be organized into lattices. The only restriction compared to FCA is that the concept lattices may not be complete but this has no practical impact when data is finite.

Definition 5 (graph concept). *A* graph concept *with arity k is a pair (R, Q), made of an object relation $R \in \mathcal{R}_k$ (the* extent*) and a PGP $Q \in \mathcal{Q}_k$ (the* intent*), such that $R = Q'$ and $Q \equiv_q R'$.*

Figure 2 displays a compact representation of the graph concepts about the British royal family. Each node x identifies a unary concept (e.g., Q3e) along with its extent (here, $\{\mathit{Charlotte}, \mathit{Georges}, \mathit{Harry}, \mathit{William}\}$). The concept intent is the PGP $((x), P)$, where P is the subgraph containing node x and all white nodes (called the *pattern core*, i.e. the nodes that appear in all represented concepts). Concept Q3e is concept "child", which, in the graph context, always has a known father and mother, which always have a son. Note that the son maybe either the child's brother or the child himself because homomorphisms need not be injective. Concept Q1a is concept "female person". Concept Q4i uniquely characterizes Charlotte in the graph context as being female, having parents, paternal grand-parents, and a brother. Note that there is no concept that uniquely characterizes Harry because his description is a subset of William's description; hence concept Q4g gathering William and Harry. In total, there are 19 unary concepts (top and bottom concepts are not represented in Figure 2). Figure 3 shows the lattice structure of all unary concepts. Binary concept intents are obtained by picking two nodes to form a projection tuple, and by taking the union of the two patterns. For example, concept (Q3a,Q3b) is concept "couple sharing a son". Concept (Q3e,Q3d) defines the relationship "having a brother", if we exclude self-relationships. Ternary and other n-ary concepts are formed likewise.

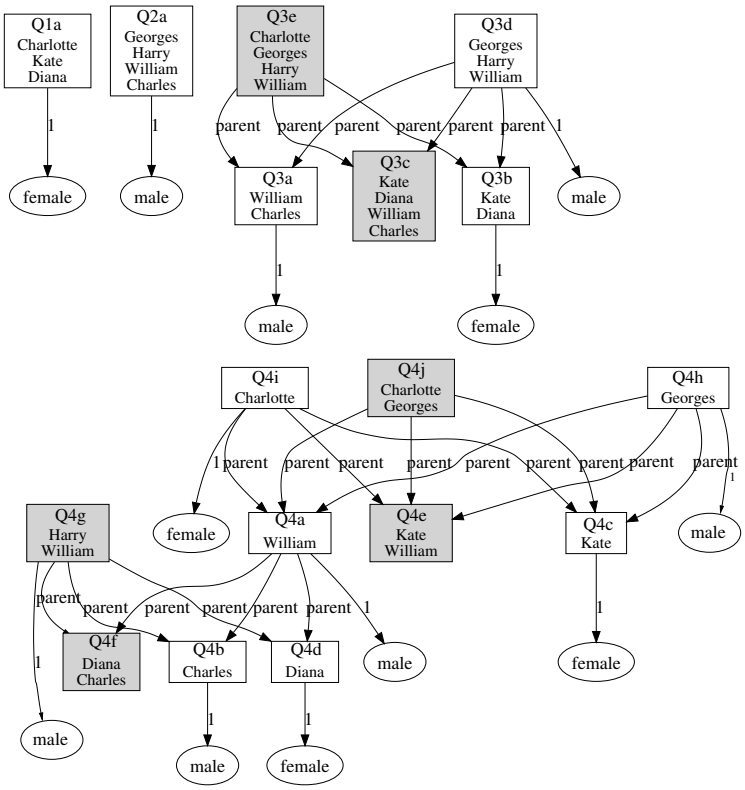

Fig. 2. Compact representation of graph concepts about the British royal family.

3 Related Work

G-FCA graph patterns bear much similarity with Conceptual Graphs (CG) [13,1], and we re-used their graphical notation. We adopted a slightly simpler formalization by not distinguishing between concept types, relation types, and individual markers, which are all modeled with attributes in G-FCA for uniformity. The semantics of knowledge graphs (e.g., CG type hierarchies, RDF Schema) is not natively handled but FCA techniques like scaling can easily be applied to G-FCA. Those differences are minor, and the novelty of G-FCA lies in projected graph patterns (PGPs), PGP intersection, and concept formation from a knowledge graph. For comparison, reasoning with CGs is mostly based on graph homomorphisms, typically between a source graph and a query graph.

G-FCA concept formation works differently from graph mining approaches [9,15,14] because they generally consist in finding frequent substructures in a collection of graphs (e.g., molecules), and they use subgraph isomorphism instead of homomorphisms.

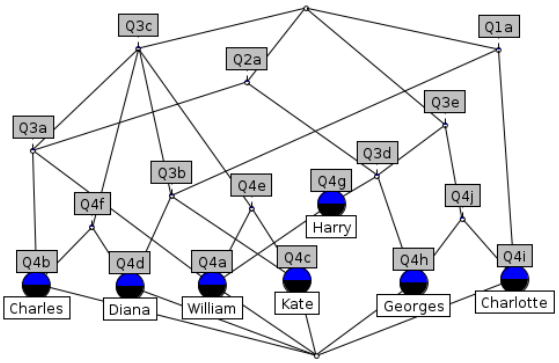

Fig. 3. Lattice of the unary concepts about the British royal family.

Previous FCA extensions, Logical Concept Analysis (LCA) [4] and Pattern Structures (PS) [5], have definitions for the Galois connection that look much like those of G-FCA (Theorem 1). However, in those extensions, descriptions only apply to single objects, and are independent one from the other. In G-FCA, the whole knowledge graph serves as a description, not only for single objects but also for tuples of objects. This allows for describing n-ary relationships between objects, as well as discovering new relationships (concepts) as complex combinations of primitive ones.

Another FCA extension, Relational Concept Analysis (RCA) [11], also discovers concepts in a graph context. RCA contexts are limited to unary and binary attributes, and RCA concepts are limited to unary concepts. However, the main difference lies in the nature of concept intents: (possibly infinite) rooted tree patterns instead of projected graph patterns. This implies that interesting cycles in data cannot be expressed in RCA concepts (e.g., concept Q3e in Figure 2). Other advantages of G-FCA are (1) a declarative, rather than iterative, characterization of concepts, and (2) a self-contained graph-based representation of concept intents instead of cascading references to concepts.

4 Computation and Presentation of Graph Concepts

The computation of graph concepts is challenging because of the complexity of computing with graphs. The fact that PGP inclusion is based on graph homomorphism rather than on subgraph isomorphism like in most other work on graph patterns [9,15] is both an advantage and a drawback. The advantage is that every intersection of two PGPs is a PGP, so that it is not necessary to reason on sets of PGPs for computing concept intents. The drawback is that an intersection $Q_1 \sqcap_q Q_2$ may be larger than both Q_1 and Q_2 when an object has several edges with the same attribute: e.g., a parent of several children. Another difficulty is that it is more difficult to get canonical representations of PGPs compared to FCA sets of attributes. Indeed, two PGPs may be equivalent al-

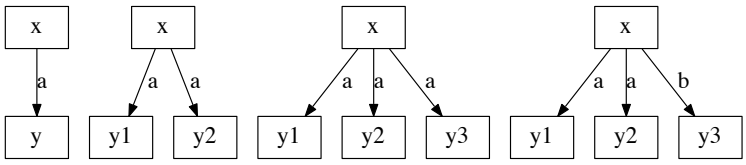

Fig. 4. Graphs **H** and **G1** are retracts of graph **G2** but not **G3**. Graph **H** is the core of graphs **G1** and **G2**. Hence, graphs **H**, **G1**, and **G2** are equivalent.

though their graph patterns are not isomorphic: e.g., graphs **H**, **G1**, and **G2** in Figure 4. In the following, we first describe the naive version of bottom-up generation of concepts up to some arity k (Section 4.1). We then sketch an algorithm to factorize computations, and to generate a *concept basis* as a subset of unary concepts from which other concepts can be derived by simple operations (Section 4.2).

4.1 Naive Generation of Concepts

The adopted strategy is to generate concept intents in a bottom-up fashion, based on the second half of the Galois connection: $R' := \cap_q\{(\overline{o}, I)\}_{\overline{o}\in R}$. The principle for each arity k is to start from the set of all descriptions of k-tuples of objects $\{(\overline{o}, I) \mid \overline{o} \in O^k\}$, and to close it by application of PGP intersection $\cap_q$. It is easy to see why this naive generation is far from optimal. For example, in the royal family context, the generation of the unary concepts from the 7 objects already generates $C_7^2 = 21$ PGP intersections, and hence 21 alignements of the incidence relation on itself. It gets exponentially worse with concept arity increasing.

In order to detect when an intent has already been generated, each PGP $Q = (\overline{x}, P)$ (including object tuple descriptions) must be given the canonical representation of its equivalence class. That canonical representation is computed in two steps. First, the minimal *retract* R of the graph pattern P that contains the projected nodes $\overline{x}$ must be found. Second, the nodes of R are numbered in a canonical way assuming a fixed ordering of attributes. Roughly, a *retract* of a graph G is a subgraph of G that conveys the same information (for details see [7], p. 112). In Figure 4, graphs **H** and **G1** are retracts of graph **G2** but not **G3**. Indeed, stating several times that x is in a a-relation to something adds nothing to stating it once because variables $y1$, $y2$, and $y3$ can map to the same object in the graph context. On the contrary, **G3** states that x is both in a a relation and a b-relation, and cannot retract to **H**: edge $((x, y3), b)$ cannot fold onto edge $((x, y2), a)$. A *core* is a minimal retract. In Figure 4, graph **H** is the core of graphs **G1** and **G2**. If there are several cores, any of them can be taken as they are isomorphic.

4.2 Efficient Generation of a Concept Basis

In this section, we sketch an algorithm for a more efficient generation of concepts (Algorithm 1). The objective is not quantitative performance (this is let to future work), but *qualitative* performance. By that we mean the orderly generation of concepts avoiding as much as possible duplication both in computation and in presentation of results.

Algorithm 1 Generation of concepts

Require: $K = (O, A, I)$ is a graph context
Ensure: $Concepts$ is the concept basis, a set of unary concepts $(R, Q)@P$ where each concept intent is presented as a subgraph of pattern P

1: $Concepts \leftarrow \emptyset$
2: $Patterns \leftarrow \{I\}$ // a queue of patterns to process
3: **for all** $P \in Patterns$ **do**
4: **for all** new $P_a \in ConnectedComponents(P \cap_q I)$ **do**
5: $P_a \leftarrow removeDuplicateNodes(P_a)$
6: **for all** new $P_b \in Retracts(P_a)$ **do**
7: $X \leftarrow P_b.nodes \setminus \bigcup_{R \in Retracts(P_a) | R \subsetneq P_b} R.nodes$ // nodes inducing P_b
8: **if** $X \neq \emptyset$ **then**
9: **for all** $x \in X$ **do**
10: $Q \leftarrow ((x), P_b); R \leftarrow Q'$ // intent and extent
11: $Concepts \leftarrow \{(R, Q)@P_a\} \cup Concepts$
12: **end for**
13: $Patterns \leftarrow \{P_b\} \cup Patterns$ // queuing P_b for intersection
14: **end if**
15: **end for**
16: **end for**
17: **end for**

Generation of Graph Patterns. Instead of generating PGPs directly by PGP intersection, we first generate alignments (categorical products) of graph patterns ($P_1 \times P_2$), ignoring at this stage projection tuples (Step 4). Thus, the incidence relation is aligned onto itself only once ($I \times I$). The graph product may have several connected components. However, if a concept has an intent whose pattern is made of several connected components, then each component is closed, hence forms a concept intent. Therefore, non-connected concepts could be composed from connected ones, and we choose to only generate connected PGPs, and hence only connected patterns. To summarize, whenever a product of two connected patterns is computed, each connected component P_a of the product becomes an input for the next stage (Step 4).

Generation of Concept Intents. For each connected graph pattern P_a, and for each projection tuple $\overline{x}$ taken from P_a-nodes, the PGP $(\overline{x}, P_a)$ is a concept

intent. The number of projection tuples can be reduced by abstracting over the tuple ordering, and by considering projection sets instead of projection tuples. Indeed, every permutation of the projection tuple of a concept intent obviously leads to another concept intent. For each projection set X, the PGP (X, P_a) can be minimally retracted to (X, P_b) by choosing the smallest retract P_b of P_a that contains nodes X. We say that X *induces* retract P_b. The generation of PGPs can be optimized for two reasons. First, several projection sets may induce the same retract; second, if a projection set X induces retract P_b, then every projection set Y of P_b-nodes that includes X induces the same retract P_b. A more efficient strategy is therefore to first generate all retracts of P_a (Step 6), and then for each retract P_b to find the minimal projection sets that induce P_b (Step 7, after second optimization below). A non-minimal projection set could simply be obtained by extending it with other nodes of P_b. For example, from unary concept Q3e in Figure 2, one could define the binary relationship "brother" by extending the projection set to Q3d. Note that all retracts P_b contains the core P_c of P_a as a subgraph. Generating all retracts amounts to computing all matches of P_a onto itself. Finding the minimal projection sets amounts to enumerating the minimal subsets of P_b-nodes that are not included in any retract that is smaller than P_b. In this way, all concept intents sharing a same graph pattern are found together (Steps 10-11). For an even more compact presentation, concepts can further be grouped by the core P_c of their pattern. Finally, each new retract P_b is fed as input to the previous stage (Step 13).

Optimization in case of symmetries. In case of 1-n or n-n relationships (e.g., persons having several children), the product graph patterns P_a often exhibit duplications in the sense that several non-adjacent nodes play exactly the same role in the pattern: e.g., nodes $y1$ and $y2$ in graph **G3** of Figure 4. Duplications can lead to a combinatorial explosion in the generation of all retracts of a pattern P_a. The optimization consists in first partitioning P_a-nodes by grouping those that play the same role, and then keeping only one node out of each group (Step 5). The only consequence is to miss the concept intents that have several projected nodes from the same group. However, those intents could easily be retrieved by duplicating projected nodes in generated intents. For example, the binary concept "parent couple" (Q3c,Q3c) is obtained by duplicating the unary concept "parent" Q3c (see Figure 2).

Optimization by reduction to unary concept intents. It can be proved that for a retracted pattern P_b induced by a projection set X, P_b is the union of the retracts induced by each projected node $x \in X$. Conversely, the union of the retracts induced by a row of projected nodes $x \in X$ is the retract induced by X. Therefore, n-ary intents could be derived from the set of unary intents extracted from a connected pattern P_a (Step 9). For that reason, it is important, in the representation of generated intents, to collectively show the patterns of unary intents as a subgraph of P_a (hence the notation $@P_a$ at Step 11), rather than individually.

4.3 Implementation

We have implemented the above algorithm as a prototype in about 1700 lines of OCaml[2]. It takes as input a graph context, i.e. a set of directed hyper-edges. It returns as output a compact representation of the concept basis, like in Figure 2. For each pattern, its core nodes are shown in white while other projection nodes are shown in grey. Each node represents a unary concept, and is identified by the pattern number and a letter (e.g., **Q1a**). The graph pattern of the concept intent is the subgraph induced by the set of nodes made of: the projected node x, the core nodes, and possibly other nodes between x and core nodes (indicated between brackets after x's label). N-ary concepts can be derived by picking several nodes, and merging their patterns. The prototype has options for specifying maximum intent pattern size (nb. of nodes), for computing and displaying unary concept extents, and for formatting results in graphical form (`.dot` file) or in Prolog-like textual form.

5 Use Cases

In this section, two use cases are presented in order to show the interest of G-FCA concepts and their PGPs. The first one extracts linguistic structures from parse trees, comparing two graph modellings. The second one extracts workflow patterns from cooking recipes, highlighting the benefits of n-ary relationships.

5.1 Extraction of Concepts in Parse Trees

We have conducted experiments on the poem of the french poet Charles Baudelaire named *"La beauté"*[3]. The goal of those experiments is the extraction of syntactic structures appearing in the text. We use the french chunker of the tool Treetagger [12][4] in order to automatically build the parse tree of the text.

From the computed parse tree, we propose two modellings to represent the poem: the first model represents the parse tree without taking into account the order between words, only the composition relation, whereas the second model explicitly encodes the order between words in addition to the composition relation. Both modellings are presented in the sequel and then the extracted patterns are discussed.

Composition Modelling In the first representation of the text, edges represent a "contain" relation between phrases and words. For instance, Figure 5 shows an excerpt of the graph of the sentence *"Je suis belle, ô mortels !"*. In this representation, the sentence ("s") is described as containing a verbal nucleus ("vn") which in turn contains: (1) a verb ("ver") which has a word ("suis"), a lemma ("suivre" or "être") and another part-of-speech (POS) information

[2] Source code, datasets, and concept sets at `https://bitbucket.org/sebferre/g-fca`
[3] In "Les Fleurs du mal". Charles Baudelaire. 1857
[4] `http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/`

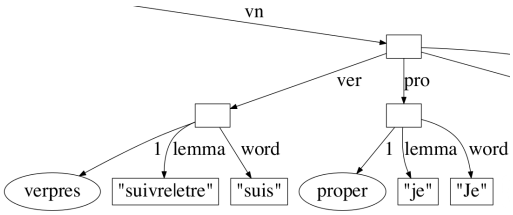

Fig. 5. Excerpt of the composition modelling for the phrase "Je suis".

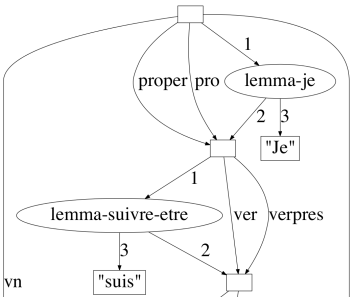

Fig. 6. Excerpt of the sequential modelling for the phrase "Je suis".

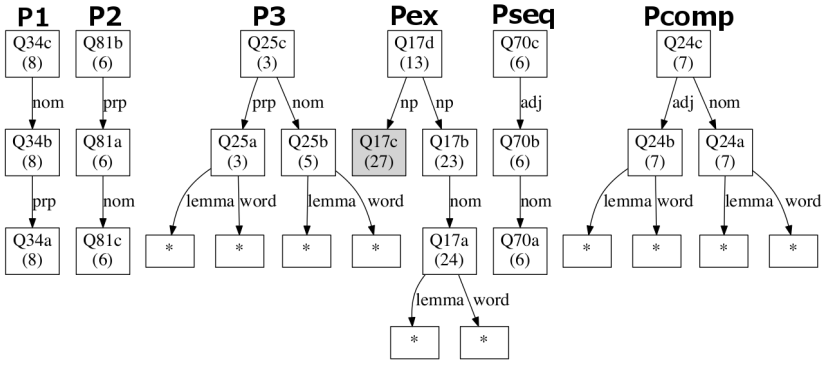

Fig. 7. Concepts extracted from the poem.

("verpres", i.e. verb in present) ; (2) a pronoun ("pro") which has a word ("je"), a lemma ("je") and another part-of-speech information ("proper", i.e. personal pronoun) ; and two other words (a punctuation and an adjective). Note that the part-of-speech tags ("ver", "vn", "proper", etc) are Treetagger tags.

Sequential Modelling In the second representation, nodes represent positions between words, and edges represent phrases and words between those positions. POS tags and lemmas are used as attributes. For instance, Figure 6 shows an excerpt of the graph of the sentence *"Je suis belle, ô mortels !"*. For instance, the word "Je" is represented by three edges from the top node: attributes "proper", "pro" (POS information), and "lemma-je" (lemma). The fact that edge "vn" overlaps edges "pro" and "ver" represents composition.

Discussion about extracted patterns When extracting concepts from both representations, we note that more patterns are extracted from the sequential model

than from the composition model. For instance, with parameter $maxsize = 10$ (maximum number of nodes per PGP), 284 patterns are extracted in the sequential model instead of 68 patterns for the composition model[5]. Indeed, in the sequential model, the graph structure is rigid, the order between words is really important. When concepts are extracted, the sequential model thus generates more distinct patterns. For instance, let us consider P1, P2 and P3 in Figure 7. P1 and P2 are extracted from the sequential model whereas P3 is extracted from the composition model. The three patterns represent the association between a preposition and a noun. However the composition modelling generates only one concept when the sequential modelling generates two concepts taking into account the ordering of the two words.

Some structural information about the text can be retrieved in the concepts. For instance, in Figure 7, P_{ex}, extracted from the composition model, exhibits an obvious pattern in the poem, i.e. a noun phrase (np) which contains a noun (*nom*). The size of the extent of unary concepts is given between brackets. Concept (Q17b) can be read as "a noun phrase that contains a noun and that belongs to something". Note that the size of the extension of (Q17b) is 23 objects. Concept (Q17a) can be read as "a noun that belongs to a noun phrase that belongs to something". Note that the size of the extension of (Q17a) is 24 objects. The size of (Q17a) is thus greater than the size of (Q17b), it means that an object in the extension of Q17b contains not only one but two nouns ("un rêve de pierre"). Concept (Q17c) can be read as "a noun phrase that belongs to something that contains a noun phrase that contains a noun". Note that the size of the extension of (Q17c) is 27 objects. This concept is interesting, indeed it exhibits the fact that a noun phrase does not necessarily contain a noun in this text, for instance it can also be a pronoun: *"où"* (where), *"chacun"* (everyone). It also shows that two noun phrases can be found in the same structure.

The information conveyed by the two modellings are not the same, however both representations are interesting. For instance, let us consider the two patterns P_{seq} and P_{comp} in Figure 7 that represent the association of a noun and an adjective. The size of the extension of concepts in P_{seq} is 6 and the size of the extension of concepts in P_{comp} is 7. Indeed, the six phrases that match P_{seq} also match P_{comp}. However the phrase *"toutes choses plus belles"*, which contains an adverb ("*plus*") between the noun and the adjective, only match Q_{comp}. The sequential modelling allows to take into account the order between words, it is more accurate whereas the composition modelling allows for more general patterns. The choice of the more appropriate modelling depends on the task.

5.2 Extraction of Concepts in Recipes

We have also conducted experiments on recipes. Four recipes are modelised: chocolate apple pie, strawberry-apple pie, mango-coconut pie and condoeuvre (Rhubarb pie or gooseberry pie). In this example, n-ary relations are used to

[5] Note that, the extraction of the 284 patterns in the sequential model takes about 4s and the extraction of the 68 patterns in the composition model takes about 20s.

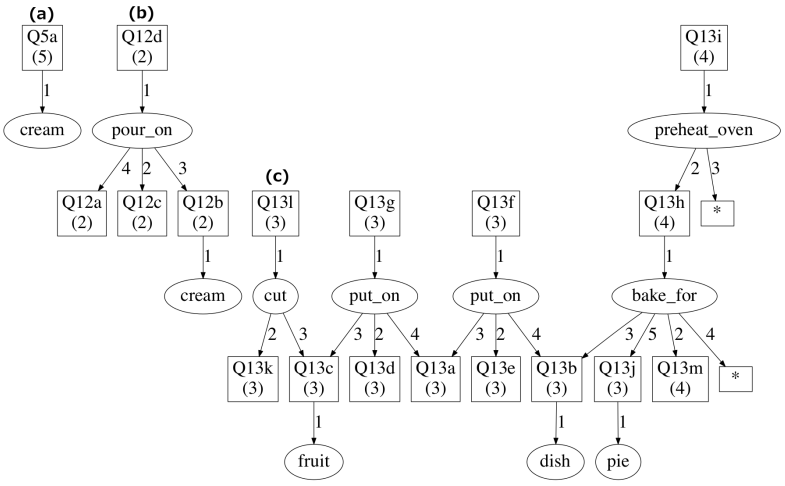

Fig. 8. Three concepts extracted from the recipes.

represent temporal constraints between actions, and entities manipulated by actions. For instance, "put_on" is a quaternary relationship relating (1) start, (2) end, (3) object (e.g., "fruit"), and (4) destination (e.g., "pastry"). All action attributes use a similar schema (e.g., "cut", "bake_for"); other attributes represent types of ingredients or ustensils (e.g., "cream", "dish"). From those four recipes, 43 patterns are extracted in less than 1s. An excerpt is given in Figure 8. Some patterns are very small and very frequent as (*a*) in Figure 8. They represent ingredients (e.g., sugar, cream) or atomic actions (e.g., "put something on something", "pour on"). Some patterns are larger but less frequent as (*b*) in Figure 8. They represent refinements of previous patterns or very specific actions (e.g., "pour cream on something"). Finally, some patterns are large and still frequent as (*c*) in Figure 8. They correspond to the abstraction of many recipes. In this example, the pattern represents an abstraction of a pie recipe. It means: "cut the fruit in order to put it on something (often a pastry), which is put on a dish, which is baked, after preheating the oven, in order to obtain a pie". As an example of n-ary concept, the ternary concept (Q13j,Q13a,Q13c) can be used to relate a pie (Q13j) to the kind of base (Q13a) and fruit (Q13c) it is made of, while abstracting over other details of the recipe.

6 Conclusion

In this paper, we have proposed an algorithm to compute graph concepts in knowledge graphs. In particular, we tackle the problem of the generation and representation of the PGPs representing concept intents in a compact way. We also describe two use cases. The first use case, on textual data, allows to discuss two kinds of modelling (with or without sequentiality). The other use case, on

cooking recipes, shows the interest of G-FCA for n-ary relations. With those two use cases we have seen that PGPs offer expressive patterns that can mix sequentiality, temporality, and composition thanks to n-ary relations. However, the set of extracted concepts can be large. Further work is to find a way to facilitate, for a user, navigation among them.

References

1. Chein, M., Mugnier, M.L.: Graph-based knowledge representation: computational foundations of conceptual graphs. Advanced Information and Knowledge Processing, Springer (2008)
2. Chekol, M.W., Euzenat, J., Geneves, P., Layaïda, N.: SPARQL query containment under RDFS entailment regime. In: Automated Reasoning (IJCAR), pp. 134–148. Springer (2012)
3. 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)
4. 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)
5. 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)
6. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer-Verlag New York. (1999)
7. Hahn, G., Tardif, C.: Graph homomorphisms: structure and symmetry. In: Graph symmetry, pp. 107–166. Springer (1997)
8. Huchard, M.: Analyzing inheritance hierarchies through formal concept analysis: A 22-years walk in a landscape of conceptual structures. In: MechAnisms on SPEcialization, Generalization and inHerItance (MASPEGHI). pp. 8–13. ACM (2007)
9. 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)
10. Muggleton, S., Raedt, L.D.: Inductive logic programming: Theory and methods. Journal of Logic Programming 19,20, 629–679 (1994)
11. 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)
12. Schmid, H.: Probabilistic part-of-speech tagging using decision trees. In: Int. Conf. New Methods in Language Processing (1994)
13. Sowa, J.: Conceptual structures. Information processing in man and machine. Addison-Wesley, Reading, US (1984)
14. Washio, T., Motoda, H.: State of the art of graph-based data mining. SIGKDD Explor. Newsl. 5(1), 59–68 (Jul 2003), `http://doi.acm.org/10.1145/959242.959249`
15. Yan, X., Han, J.: Closegraph: mining closed frequent graph patterns. In: ACM Int. Conf. Knowledge discovery and data mining (SIGKDD). pp. 286–295. ACM (2003)