Modeling Complex Structures in Graph-FCA: Illustration on Natural Language Syntax

Sébastien Ferré, Peggy Cellier

Modeling Complex Structures in Graph-FCA: Illustration on Natural Language Syntax

Sébastien Ferré and Peggy Cellier[⋆]

Univ Rennes, INSA, CNRS, IRISA
Campus de Beaulieu, 35042 Rennes, France
Email: ferre@irisa.fr, peggy.cellier@irisa.fr

Abstract. Graph-FCA is an extension of formal concept analysis for multi-relational data. In this paper, we discuss the freedom of representation offered by Graph-FCA, in particular by its support of n-ary relations, considering natural language syntax as a use case.

1 Introduction

Several extensions of formal concept analysis have been proposed to handle multi-relational data, such as Relational Concept Analysis (RCA) [7] and Graph-FCA [4]. Graph-FCA adopts a *graph* view of multi-relational data, seeing entities as *nodes* and relationships as *directed edges* between nodes. Graph-FCA concept intents are therefore characterized and visualized as graph patterns. Graph-FCA is the formal basis of *concepts of neighbors*, which have been applied to similarity search in knowledge graphs [3], knowledge graph completion [5], and relation extraction [1].

A distinctive feature of Graph-FCA is to uniformly represent entity attributes, relations and n-ary relations ($n > 2$) through the notion of *hyper-edge*, whereas RCA differentiates between attributes and binary relations. N-ary relations offer a lot of freedom in the representation of complex structures: e.g., a ternary relation can be represented directly by one ternary relation or decomposed into three binary relations, whereas RCA has only the latter option [6].

The objective of this paper is to demonstrate and discuss this freedom of representation in Graph-FCA. At the same time, the input and output format of the Graph-FCA tool is explained through concrete examples. As a use case, we have chosen to analyze syntactic descriptions of English sentences with the goal to discover common syntactic patterns. Such descriptions are complex because they combine: tokens, words, part-of-speech (POS) tags, parse trees, and the relative position of tokens and phrases. An important lesson learned is that a naive representation of structures may lead to uninteresting patterns, and that n-ary relations are useful to alleviate the problem.

2 Graph-FCA Theory

Graph-FCA defines a *graph context* as an incidence relation between *tuples of* objects and attributes. A *graph context* is a triple $K = (O, A, I)$, where O is a set

[⋆] This research is supported by ANR project SmartFCA (ANR-21-CE23-0023).

of *objects* (the nodes), A is a set of *attributes* (the hyper-edge labels), and $I \subseteq O^* \times A$ is an *incidence relation* (the hyper-edges). An hyper-edge $((o_1,...,o_k),a) \in I$ can be seen as the logical atom $a(o_1,...,o_k)$ where attribute a is used as a predicate with arity k (we use the atom notation for readability). Unary attributes ($k=1$) are used to label nodes while binary attributes ($k=2$) are used to label edges.

In a *graph concept*, the extent is a set of n-tuples of objects, and the intent expresses what those n-tuples of objects have in common, where n is the arity of the concept. Unary concepts ($n=1$) are about sets of objects, while other concepts ($n>1$) are about relationships between objects. A concept intent is a *graph pattern* with n distinguished nodes, called a *Projected Graph Pattern* (PGP), and it can be seen as conjunctive query. For example, the PGP $\big[(x,y) \leftarrow parent(x,z), parent(y,z)\big]$ has three nodes x,y,z, two edges $parent(x,z)$ and $parent(y,z)$, and two distinguished nodes x,y. It can be used as a definition of the "sibling" relationship, i.e. the fact that x and y have a common parent z. A *core concept* is a concept whose graph pattern cannot be "simplified", i.e. is not homomorphic to a smaller graph pattern. Unlike RCA, there is a concept lattice for each concept arity rather than for each object type. Classical FCA is a special case of Graph-FCA where $k=1$ (only unary attributes) and $n=1$ (only unary concepts).

3 Graph-FCA Implementation

A Graph-FCA tool, called `gfca`, is available as a web application, a web service and a source code (visit `https://bitbucket.org/sebferre/graph-fca/`). It is implemented in about 3000 lines of OCaml, and a general presentation is given in [4]. A reference manual details the command line options, the input format, and the different output formats. The input format is inspired by λProlog [2], an extension of Prolog to λ-terms that uses a currified notation for logical atoms. For example, the atom $capital(France, Paris)$ is noted `capital France Paris`. More in general, an hyper-edge $a(o_1,...,o_k)$ is noted $a\ o_1\ ...\ o_k$. The input format offers a system of macros (attribute definitions) and syntactic sugar to allow for more concise notations of graph contexts.

The tool can output graph concepts (and optionally the graph context) in a textual format close to the input format (option `-txt`), and/or in JSON (option `-json`), and/or a graphical format using DOT and SVG (option `-dot`). Examples are shown below. There are several options to control the display of concepts: e.g., `-only-cores` to only show core concepts; `-ext` to label each unary concept by its extent (as a list of objects); `-minsupp` to only show the concepts with *minsupp* objects in their extent.

4 Use Case: Modeling Complex Syntactic Structures

Syntactic structures offer an interesting use case for demonstrating and discussing the representation of complex structures in Graph-FCA. To make the following illustrations fit in the paper, we consider a small context made of only two short sentences: (1) *"a black cat sits on a mat"*, and (2) *"a dog barks at the red car"*. To obtain syntactic information about those sentences, we apply CoreNLP (`https://corenlp.run/`) to each of them. Figure 1 shows the two different parse

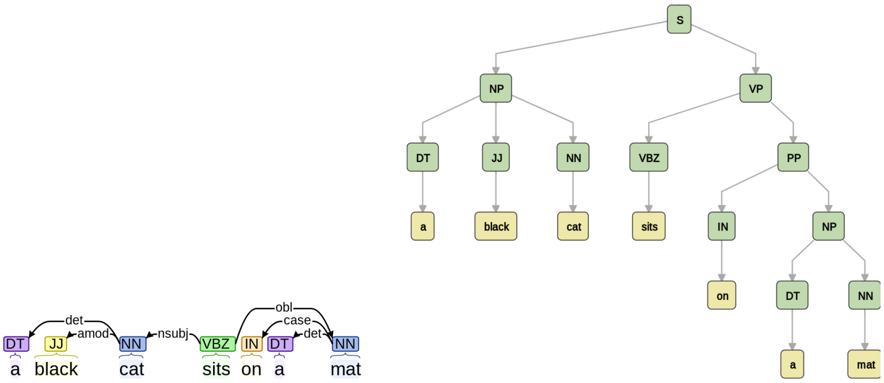
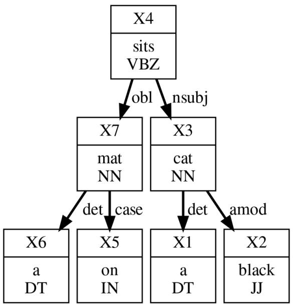


Fig. 1. Dependency/constituency tree of sentence "a black cat sits on a mat" (CoreNLP).



```
:-
X1 : [a, DT],
X2 : [black, JJ],
X3 : [cat, NN, det X1, amod X2],
X4 : [sits, VBZ, nsubj X3, obl X7],
X5 : [on, IN],
X6 : [a, DT],
X7 : [mat, NN, case X5, det X6].
```

Fig. 2. Representation of the dependency tree of sentence "a black cat sits on a mat", in graphical notation (left) and in textual notation (right).

trees returned by CoreNLP: the *dependency tree* that labels tokens with POS tags and links them with dependencies; and the *constituency tree* that aggregates tokens into phrases, and recursively phrases into larger phrases.

4.1 Labelled Directed Graphs

The dependency tree of a sentence is mathematically a labelled directed graph, with tokens as nodes and dependencies as edges. Each node is labelled by a word (e.g., "cat") and a POS tag (e.g., NN), and each edge is labelled by a dependency type (e.g., *nsubj*). A dependency tree therefore admits an immediate representation as a graph context, with tokens as objects, words and POS tags as unary attributes, and dependency types as binary attributes. Figure 2 (left) shows the graphical notation of the first sentence in the `gfca` tool. Each box represents an object, with its identifier at the top (e.g., `X4`, the 4th token), and its unary attributes at the bottom (e.g., `sits`, `VBZ`).

In `gfca`'s input format, the first sentence can be written as follows, as a set of edges: `sits X4, VBZ X4, nsubj X4 X3, cat X3, NN X3, det X3 X1`... To enable a more compact notation, the input format offers some syntactic sugar. Figure 2 (right)

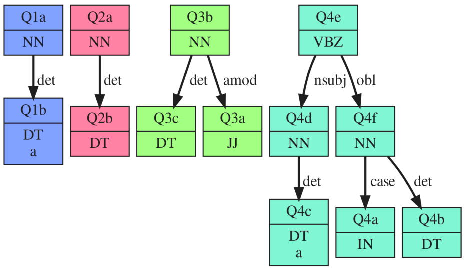


Fig. 3. Graph concepts of the dependency trees (minsupp=2).

shows an object-centric notation, as an alternative to the default edge-centric notation. Each line describes an object by listing all edges starting at it. For instance, the set of edges `cat X3, NN X3, det X3 X1, amod X3 X2` is compacted into `X3 : [cat, NN, det X1, amod X2]` by factorizing on the first argument `X3`.

Figure 3 shows the 4 distinct graph patterns of concept intents. Several concepts may share the same pattern, only differing by the distinguished nodes. Here, as in the rest of this paper, we applied the options `-only-cores -dot -minsupp 2` to get the graphical representation of core patterns that occur at least twice. The 4 graph patterns can be read as follows: (Q1) a noun (NN) with the specific determiner 'a'; (Q2) a noun with a determiner (det, DT); (Q3) a noun with a determiner and an adjective modifier (amod, JJ); (Q4) a verb (VBZ) with, as a subject (nsubj), a noun with determiner 'a', and as an oblique complement (obl), a noun with a preposition (case, IN) and a determiner. Q4 abstracts over the two sentences, showing what they have in common. Q2 abstracts over all noun phrases, showing that all nouns have a determiner. Q1 and Q3 are specializations of Q2, respectively with a specific determiner and an adjective.

4.2 Rooted Trees

The constituency tree is mathematically a rooted tree. It can be seen as a directed graph with edges from each phrase to its constituents. Compared to the dependency tree, the nodes represent not only tokens but also phrases, and there is only one type of edge that represents a part-of hierarchy over the nodes. The nodes are labelled with words and phrase types (e.g., S, NP). In our Graph-FCA representation, we choose to merge the atomic phrases and the tokens, and we introduce a binary attribute `part` as an edge label. Figure 4 (left) shows the graphical notation for the first sentence (`X`*i*–*j* denotes the phrase spanning from token `X`*i* to `X`*j*). Figure 5 shows the 7 found graph patterns, among which patterns Q3, Q4, Q6 and Q7 respectively correspond to the four patterns found on the dependency trees. The three other patterns have no other attribute than `part`. They only reflect the structural shape of trees, and do not bring information on English syntax.

In order to avoid uninteresting patterns, all attributes should be meaningful for the domain. It is therefore desirable to eliminate the `part` attribute, while retaining the tree structure. We observe that, whenever there is an edge (`part` *Parent Child*), e.g.,

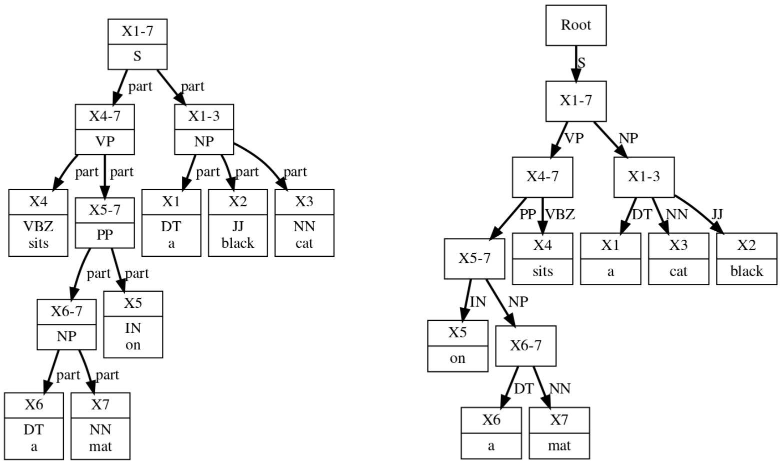


Fig. 4. Representation of the constituency tree of the sentence "a black cat sits on a mat", with the `part` attribute (left) and without (right).

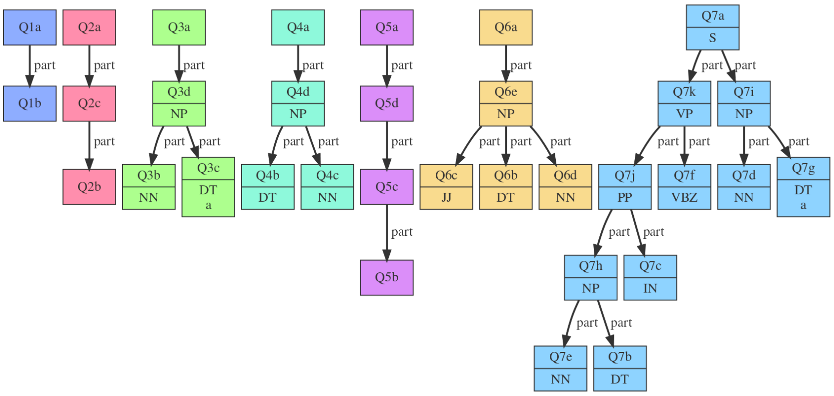


Fig. 5. Graph concepts of the constituency trees (with the `part` attribute).

`part X1-7 X4-7`, there is also a unique edge (σ *Child*) where σ is a phrase type, e.g., `VP X4-7`. A solution is therefore to merge the two edges into a domain-specific edge (σ *Parent Child*), e.g. `VP X1-7 X4-7`. Each unary attribute σ is therefore lifted into a binary attribute that encodes for both the tree structure and the phrase type. Figure 4 (right) shows the new representation for the first sentence. Note that phrase types now appear as edge labels because they are now binary attributes. It can be verified that only the four interesting graph patterns are now found (omitted for lack of space).

4.3 Constituency Trees: Handling Relative Positions

As one can see in the above figures, the previous representations adequately handle the part-of aspect of constituency trees but not the relative positions of phrases, which is a key aspect of syntax in general. For instance, in a noun phrase, the determiner comes first, not at an arbitrary position, and the adjective comes before the noun. An effective way to handle relative positions and ordering is to describe each phrase by its starting and ending position. In Graph-FCA terms, we introduce new objects for the offset positions (`I0`..`I7` for the first sentence), and two new binary attributes `start` and `end` for linking phrases to their start/end positions. Position `I`k is the position right after the kth token. For instance, the new edges for `X1-3` are `start X1-3 I0` and `end X1-3 I3`. However, like with `part`, the `start` and `end` attributes are purely structural, and lead to uninteresting patterns saying that phrases end where other phrases start and the like. We can eliminate them by observing that edges (`start` X I_s) and (`end` X I_e) come with one and only one edge (σ X' X). We can therefore merge them into a single quaternary edge (σ X' I_s X I_e) that expresses at the same time the phrase type, the tree structure, and the position in the sentence. For instance, the description `X1-7 : [NP I0 X1-3 I3, VP I3 X4-7 I7]` says that phrase X1-7 is composed by a NP from position 0 to 3, followed by a VP from position 3 to 7.

From there, we obtain the same four patterns, except that we now have additional information about the relative position of constituents. The patterns show that all noun phrases start with a determiner, and end with a noun. In some pattern, the determiner and the noun are adjacent while in another pattern, there is a gap between the two.

To conclude, we note that the elimination of structural attributes implies an increase of the arity of domain-specific attributes. In our example, the arity of phrase types goes from unary to binary when eliminating `part`, and then to quaternary when eliminating `start` and `end`. The ability of Graph-FCA to handle n-ary edges/attributes is here a key feature. Without them, one would be forced to keep the structural attributes, and this would result in possibly many uninteresting concepts and patterns.

References

1. 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)
2. Belleannée, C., Brisset, P., Ridoux, O.: A pragmatic reconstruction of λprolog. The Journal of Logic Programming **41**, 67–102 (1999)
3. 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)
4. Ferré, S., Cellier, P.: Graph-FCA: An extension of formal concept analysis to knowledge graphs. Discrete Applied Mathematics **273**, 81–102 (2019)
5. Ferré, S.: Application of concepts of neighbours to knowledge graph completion. Data Science: Methods, Infrastructure, and Applications **4**, 1–28 (2021)
6. Keip, P., Ferré, S., Gutierrez, A., Huchard, M., Silvie, P., Martin, P.: Practical comparison of FCA extensions to model indeterminate value of ternary data. In: Concept Lattices and their Applications (CLA). vol. CEUR 2668, pp. 197–208 (2020)
7. 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)