Biseau: An Answer Set Programming Environment for High-Level Specification and Graph Visualization applied to FCA

Lucas Bourneuf

Univ Rennes 1
lucas.bourneuf@inria.fr

Abstract. When studying mathematical relations, soon arise the need to visualize the resulting structures. This often leads to tedious development or time consuming copy-pasting of visualization routines. This paper presents Biseau, a programming environment aiming at simplifying the visualization task. Biseau takes advantage of Answer Set Programming, enabling user to write, maintain, debug and reuse close to specification encodings, and eventually obtain a graphical display of mathematical relations. This paper shows as a use-case how Formal Concept Analysis can be efficiently described at the level of its properties, without needing a costly development process. Biseau uses the graphviz software to render in real-time the calculated graphs to user, thanks to an Answer Set Programming to (graphviz) dot compiler. More generally, it is shown that Biseau leads on to a general method to prototypes new software features, which could unify the development process before the efficiency race. As such, it reproduces the core results of existing tools like LatViz or In-Close. We hope that it will help to speed up the prototyping process and simplify the exchange and reuse of proof of concept implementations.

1 Introduction

Software environments oriented towards data mining use efficient implementations of data structures and their visualizations. For instance, in Formal Concept Analysis, LatViz is a lattice visualization software, allowing end-user to explore the lattice structure efficiently [1]. In-Close implements efficient concept mining algorithms and provides a concept tree visualization of contexts [2]. All these softwares work with a formal model that provides an abstract view and a fixed search space on the data. Users cannot work on the model itself, they are expected to use the implemented methods, not to design new ones. In contrast, Biseau is a software focused on designing and exploring elements of the data

structure, rather than the data itself [3]. It is a general purpose model builder that relies on graphs and logic languages.

Graphs are rendered in multiple ways. Biseau uses dot, a generic graph description language specified by the graphviz software, which provides a gallery of visualization engines [4]. Together with a graph data structure, Biseau offers a logical view of the associated exploration methods. A pure declarative language is used for this purpose, Answer Set Programming (ASP). It allows users to transcript the formal properties they are looking for in a straightforward way. ASP has already been applied to FCA to accomplish expressive query languages for formal contexts [5, 6].

Biseau is supplied with a graphical user interface and a command line interface to write an ASP encoding, from which the dot files and the resulting visualizations are generated. The main interest of Biseau is therefore to build graph visualizations directly from formal relations. As such, it is not dedicated to lattices only, and their (efficient or scalable) exploration, as performed by specialized software such as LatViz or In-Close. It provides instead a general purpose programming environment that is able to visualize graph structures. Biseau is therefore suited for rapid design and easy testing of extensions in the framework of FCA. It is freely available under the GNU/GPL license[1].

2 From ASP to Dot With Biseau

Biseau implements an ASP to dot compiler. The graph is therefore defined in intension: instead of describing manually all objects and properties, the user specify their definitions, and let the ASP solver infers all necessary relations.

A given ASP encoding yields so-called "stable models" consisting of true facts, which can be represented by atoms like `link(woman,human)`. For each stable models found from the ASP user encoding, Biseau will convert atoms into dot lines. For instance, the ASP atom `link(woman,human)` will translate to `woman -> human` in the dot output. This controlled vocabulary will be only partially explored in Section 3, but note that it maps the full dot language, including colors, shapes, and general graph options. A complete documentation is available online[2].

Biseau internal process can be seen as a compilation from ASP models to dot, then from dot to image (the last one being delegated to graphviz software).

3 Build and Visualize Galois Lattices With Biseau

This section shows how to build FCA basic mathematical relations in order to get a visualization of the Galois lattice in Biseau. The context in Table 1 will be used as case study, encoded in ASP using `rel/2` atoms.

[1] `https://gitlab.inria.fr/lbourneu/biseau`
[2] `https://gitlab.inria.fr/lbourneu/biseau/blob/master/doc/user-doc.mkd`

	male	female	adult	child	human
man	×		×		×
woman		×	×		×
boy	×			×	×
girl		×		×	×

Table 1: Formal context of human relations. It can be encoded in ASP in atoms such as `rel(man,male)` and `rel(woman,adult)`.

3.1 Mining the Formal Concepts

In a formal context defined by objects O, attributes A, and the binary relation $R \subseteq O \times A$, a formal concept is a pair (X, Y), such as:

$$X = \{x \in O \mid (x, y) \in R \ \ \forall y \in Y\} \tag{1}$$

$$Y = \{y \in A \mid (x, y) \in R \ \ \forall x \in X\} \tag{2}$$

Where $X \subseteq O$ and $Y \subseteq A$. The search for formal concepts in ASP can be expressed like in the above definition:

```
1 ext(X):- rel(X,_) ; rel(X,Y): int(Y).
2 int(Y):- rel(_,Y) ; rel(X,Y): ext(X).
```

`rel(X,_)` fixes variable `X` as the first term of a relation, i.e. an object. Notation `rel(X,Y): int(Y)` ensure that there is a relation between `X` and all attributes Y of the intent. As a consequence, the extent `ext(X)` holds for all objects X in relation with all attributes of the intent. The second rule is a symmetric definition for the concept's intent. ASP search comes with the guarantee that all minimal fixed points will be enumerated. Therefore, each answer set is a different concept, or the supremum or infimum (where extent or intent are empty sets). These models/concepts are aggregated in order to produce an encoding containing `ext/2` (and `int/2`) atoms, where `ext(N,A)` (`int(N,A)`) gives an element of N-th concept's extent (intent).

3.2 Galois lattice

A Galois lattice is defined by the partial order on the concepts, i.e. a graph with concepts as nodes, and an edge between a concept and its successors in the ordering:

```
1 % Shortcut to infimum, supremum and concepts identifiers.
2 c(N):- ext(N,_). c(N):- int(N,_).
3 % Ordering of two concepts: the first has all objects of the second.
4 contains(C1,C2):- c(C1) ; c(C2) ; C1!=C2 ; ext(C1,X): ext(C2,X).
5 % Concepts linked to another in the Galois Lattice.
6 link(C1,C3):- contains(C1,C3) ; not link(C1,C2): contains(C2,C3).
7 % Annotate nodes with extent and intent.
8 annot(upper,X,A):- ext(X,A). annot(lower,X,B):- int(X,B).
```

These lines yield the visualization shown in Figure 1. Line 2 enables the access to the infinum, supremum and concepts with one atom. Line 4 yields pairs of concepts that are included, based on their extent. Line 6 ensure that a link exists in the lattice between a concept $C1$ containing another concept $C3$ if there no link between $C1$ and a concept $C2$ smaller than $C3$. The `annot/3` atoms are a Biseau convention (as `link/2`), allowing us to print the extent and intent of each concept, respectively above and below the node.

3.3 Reduced Labelling

The reduced labelling of a lattice is computed as the set of specific objects and attributes for each concept. This can be defined as `specext/1` and `specint/1` atoms in ASP, using the following lines along the search for formal concepts in section 3.1:

```
1 % An outsider is any object or attribute linked to
2 % an attribute or object not in the concept.
3 outsider(X):- ext(X) ; rel(X,Z) ; not int(Z).
4 outsider(Y):- int(Y) ; rel(Z,Y) ; not ext(Z).
5 % The specific part of each concept contains no outsider.
6 specext(X):- ext(X) ; not outsider(X).
7 specint(Y):- int(Y) ; not outsider(Y).
```

With these lines and the collapsing into one model described in section 3.1, we obtain `specext/2` and `specint/2` atoms, describing the AOC poset elements, attached to each concept. We can then replace the previously defined annot/3 definitions in section 3.2 with the reduced labelling:

```
1 annot(upper,X,A):- specext(X,A). annot(lower,X,B):- specint(X,B).
```

Using these definitions, Biseau produces the Figure 2.

4 Discussion & Conclusion

The main contribution of Biseau lies into the straightforward use of the structure specifications to produce a simple code and a proper visualization. To achieve that feat, Biseau is compiling a controlled subset of ASP atoms to dot lines, effectively building a dot formatted file that is compiled to an image by graphviz software. This enables definition of graphs in intension and gives an abstract access to dot expressions, letting the user focus on the fast prototyping of data exploration and the elaboration of mathematical properties.

In other words, Biseau allows user to work on the model in which data are processed, instead of providing an implementation of a single model to be used on particular data, as usually performed in field-specialized softwares.

In this paper, FCA was implemented with Biseau with a focus on the ASP implementation, although more generally it can be extended with *scripts*, units of ASP or Python to add to (or run on) the model encoding. Scripts may expose some options to tune their behavior, and integrate other programs implementing

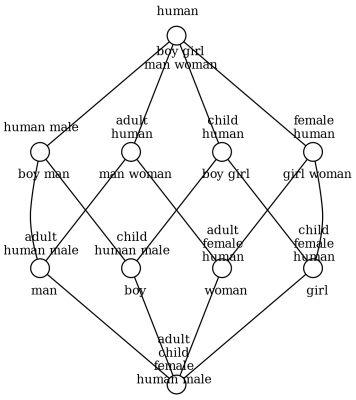

Fig. 1: Visualization of the Galois Lattice of context in Table 1 using Biseau, with extent and intent shown for each node/concept.

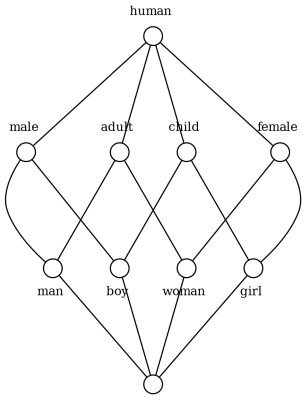

Fig. 2: Visualization of the Galois Lattice of context in Table 1 using Biseau, with reduced labelling.

standard operations efficiently, such as In-Close for concept mining. When a Biseau encoding is correctly specified, it is basically a working proof of concept, which can be distributed as a standalone script, or adapted into a stand-alone program, iteratively replacing ASP parts by other programs.

Biseau current state is a fully working proof of concept, although it misses a lot of features typically found in text editors and IDEs, that could help user to write and maintain the ASP code. It also runs into ASP typical limits, such as the absence of float numbers handling and scaling problem.

Future work will focus on Biseau as a proof of concept generator, through the automatic generation of standalone programs, and built-in embedding of other paradigms such as linear programming.

References

1. M. Alam, T. N. N. Le, and A. Napoli. Steps towards interactive formal concept analysis with latviz. In *FCA4AI@ECAI*, 2016.
2. S. Andrews. In-close, a fast algorithm for computing formal concepts. In *International Conference on Conceptual Structures*, 2009.
3. L. Bourneuf. An Answer Set Programming Environment for High-Level Specification and Visualization of FCA. In *FCA4AI'18 - 6th Workshop "What can FCA do for Artificial Intelligence?"*, pages 1–12, Stockholm, Sweden, July 2018.
4. E. R. Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *Softw. Pract. Exper.*, 30(11):1203–1233, Sept. 2000.
5. P. Hitzler and M. Krötzsch. Querying formal contexts with answer set programs. In *Proc. of the 14th Int. Conf. on Conceptual Structures: Inspiration and Application*, ICCS'06, pages 260–273, Berlin, Heidelberg, 2006. Springer-Verlag.

6. S. Rudolph, C. Săcărea, and D. Troancă. Membership constraints in formal concept analysis. In *Proc.of the 24th Int. Conf. on Artificial Intelligence*, IJCAI'15, pages 3186–3192. AAAI Press, 2015.