An Answer Set Programming environment for high-level specification and visualization of FCA

Lucas Bourneuf ✉

Université de Rennes 1
Campus de Beaulieu, 35042 Rennes cedex, France.
lucas.bourneuf@inria.fr

Abstract. This paper introduces Biseau, a programming environment dedicated to the exploration of relations through a graphical display. The use of Answer Set Programming enables the production of small code modules which are easy to maintain and debug since they are very close to the specifications. This paper shows how a mathematical framework such as Formal Concept Analysis can be efficiently described at the level of its properties, without needing a costly development process. We hope that it will help to quickly adapt a given code to the peculiarities of a data set, thereby speeding up the development of prototypes. Besides, it will also help the integration of the ideas of the FCA community in a readable and shareable format. From a practical point of view, Biseau provides an Answer Set Programming to (graphviz) dot compiler and uses the graphviz software to render in real-time the calculated graphs to user, for instance to produce concept lattices or aoc posets visualizations. Its relation with existing tools like LatViz and FCAbundles is also discussed.

1 Introduction

Large scale data production requires availability of high-level visualizations for their exploration. This is usually performed by building generic visualization models, that users may later use to explore their data. Thus, 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]. Lattice Miner builds and visualize Galois lattices and provides data mining tools to explore data [15]. FCA Tools Bundle consists in a web interface exposing multiple FCA-related tools for contexts and (ternary) concept lattices exploration [14]. In-Close algorithm reference implementation provides a concept trees visualization of contexts encoded in standard formats [3]. All these tools 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, this paper introduces Biseau, a software focused on designing and exploring elements of the data structure, rather than the data itself. In this

approach, data are only a support to the model validity, and the user's aim is the proper design of a general model. Biseau is a general purpose model builder that relies on graphs and logic languages.

Graphs are rendered in multiple ways, using field-specialized softwares like Cytoscape [21] in biology, graph-specific softwares (like LatViz for lattices), or more generalist like Tulip [4]. Another generic approach is dot, a graph description language specified by the graphviz software, which provides a gallery of visualization engines [6]. Dot is the internal graphical language used by Biseau (see Section 3).

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 (see Section 2). ASP has already been applied to FCA to accomplish expressive query languages for formal contexts [12], later extended to n-adic FCA and improved with additional membership constraints, in order to handle large context exploration [20]. In our approach, ASP is also used for visualization.

Biseau is supplied with a graphical user interface and a command line interface to write an ASP encoding. Biseau uses this encoding as a script to generate the dot files and the resulting visualizations. The main interest of Biseau is therefore to build graph visualizations directly from formal relations. Biseau is not only dedicated to lattices, and their (efficient or scalable) exploration. It provides instead a general purpose programming environment that is able to visualize any ordered structure. Biseau is therefore suited for rapid design and easy testing of works or extensions in the framework of FCA. It is freely available under the GNU/GPL license[1].

The structure of this paper is as follows. Sections 2 and 3 quickly present the ASP and dot languages used by Biseau. Section 4 explains how Biseau takes advantage of these languages to allow the user to build models. Section 5 proposes as a case study the reconstruction of the Galois lattice. Section 6 shows how Biseau can easily handle some of the FCA extensions typically used in FCA applications as knowledge processing [19]. Finally the paper concludes by some insights about Biseau interest when used in FCA and in artificial intelligence.

2 Answer Set Programming

The following presentation of ASP is taken from [5]. For an in-depth dive into the language, the reader is redirected to [8].

ASP is a form of purely declarative programming oriented towards the resolution of combinatorial problems [17]. It has been successfully used for knowledge representation, problem solving, automated reasoning, and search and optimization. Unlike Prolog, ASP handles cross-references of rules, enabling the writing of code much closer to the specification. In the sequel, we rely on the input language of the ASP system Potassco (*Potsdam Answer Set Solving Collection* [8])

[1] `https://gitlab.inria.fr/lbourneu/biseau`

developed in Potsdam University. An ASP program consists of Prolog-like rules h :- b_1, ..., b_m, *not* b_{m+1}, ..., *not* b_n, where each b_i and h are literals and *not* stands for *default negation*. Mainly, each literal is a predicate whose arguments can be constant atoms or variables over a finite domain. Constants start with a lowercase letter, variables start with an uppercase letter or an underscore (don't care variables). The rule states that the head h is proved to be true (h is in an answer set) if the body of the rule is satisfied, i.e. $b_1, \ldots, b_m$ are true and one can not prove that $b_{m+1}, \ldots, b_n$ are true. Note that the result is independent on the ordering of rules or of the ordering of literals in their body, as it is the case in Prolog. An ASP solver can compute one, several, or all the answer sets (stable models) that are solutions of the encoded problem. If the body is empty, h is a fact while an empty head specifies an integrity constraint. Together with model minimality, interpreting the program rules this way provides the stable model semantics (see [11] for details). In the head part, A *choice rule* of the form $\{p(X) : q(X)\}$ will generate $p(X)$ as the powerset of $q(X)$ for all values of X. In the body part, $\{p(\_)\}$ will count the number of atom p with one parameter, and $N = \{h\}$ evaluates N to the cardinal of the set of h. In the body part, $p(X) : q(X)$ holds if for all X, if $q(X)$ holds, then $p(X)$ holds. Finally, lines starting by `%` are comments. In practice, several syntactical extensions to the language that are not interesting for this paper are available. An example of ASP encoding is presented in Figure 1, using atoms to reproduce the context in Table 1 and a rule to build a bipartite graph linking objects and attributes.

```
1  % Facts.
2  age(john,7). age(eve,71). age(alice,15).
3  male(john). male(bob). female(alice).
4  mother(eve,bob).
5  % Rules.
6  rel(H,child):- age(H,A) ; A<12.
7  rel(H,adult):- age(H,A) ; A>=18.
8  rel(H,male):- male(H).
9  rel(H,female):- female(H).
10 rel(H,man) :- rel(H,male) ; rel(H,adult).
11 rel(H,boy) :- rel(H,male) ; rel(H,child).
12 rel(H,woman):- rel(H,female) ; rel(H,adult).
13 rel(H,girl) :- rel(H,female) ; rel(H,child).
14 rel(H,adult):- rel(H,male) ; not rel(H,boy).
15 rel(H,female):- mother(H,_).
16 % Build the visualization in Figures 2 and 3.
17 link(O,A):- rel(O,A).
```

Fig. 1: ASP program encoding the context in Table 1, in the form of `rel/2` relations between objects and attributes. The last line yield links/2 atoms that are compiled by Biseau as edges in the output dot file.

We used the Potassco system [9] that proposes an efficient implementation of ASP. ASP processing implies two steps, grounding and solving. The grounder generates a propositional program replacing variables by their possible values. The solver is in charge of producing the stable models (answer sets) of the propositional program. Of course, a dedicated algorithm for a specific problem will be generally more efficient than its equivalent compact ASP encoding. However, ASP systems are useful for the design of prototypes. It is an attractive alternative to standard imperative languages that enable fast developments.

	adult	child	female	male	boy	woman	man
alice			×				
bob	×			×			×
eve	×		×			×	
john		×		×	×		

Table 1: Formal context of human relations.

3 Graph Drawing With Dot

Dot is a graph description language, allowing one to generate a graph visualization from the definition of its content [6]. Dot enables the control of precise visual properties, such as node and edge labelling, position, shape, or color. For instance, the dot line `woman [color="blue"]` will color in blue the node labelled *woman*. The full language is defined by the graphviz graph visualization software, which provides multiple engines to interpret and compile dot encoded files to other formats, including images. Figure 2 shows an example of a working dot description, which given to a graphviz engine yields the visualization in Figure 3.

```
1 Digraph biseau_graph {
2   node [penwidth="0.4" width="0.1"];
3   edge [penwidth="0.4" arrowhead="none"];
4   john->boy; john->male; john->child;
5   eve->female; eve->woman; eve->adult;
6   bob->man;  bob->adult; bob->male;
7   alice->female;
8 }
```

Fig. 2: Dot encoding of the graph in Figure 3.

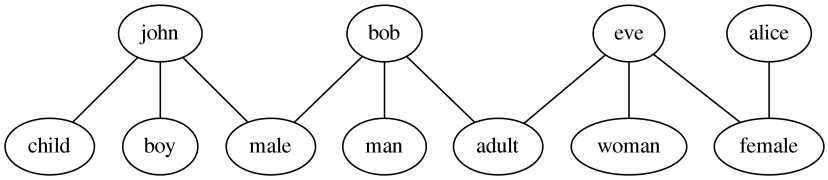

Fig. 3: Visualization of the relations described by context in Table 1.

4 From ASP to Dot With Biseau

Biseau allows the user to write some ASP encoding and retrieve in real-time the corresponding graph visualization. To achieve this, it implements an ASP to dot compiler and a Graphical User Interface that helps writing the ASP encoding and that performs automatically all necessary compilations.

As explained in Section 2, a given ASP encoding yields 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 5, but note that it maps the full dot language, including colors, shapes, and general graph options. A complete documentation is available online[2].

Because of the use of ASP to yield the dot description, 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. More generally, 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, as seen in Section 3).

As a matter of example, the ASP encoding in Figure 1 will be compiled to the dot description in Figure 2, itself compiled to the image in Figure 3. If the ASP expression `color(A,blue):- rel(_,A).` was added to the ASP encoding in Figure 1, the final figure would show in blue all attributes nodes. The reader familiar with software engineering may recognize the use of ASP as a metamodel, and dot as the model.

Biseau can be extented with *scripts*, units of ASP (or Python) code to add to (or run on) the user encoding. They may expose some options to tune their behavior. Moreover, user can implement and add its own scripts to Biseau, allowing him (and others he shares with) to encapsulate ASP or Python programs that behave accordingly to their preferences. Biseau is shipped with scripts related to FCA, for data extraction from standard format like SLF or CXT, concept mining or lattice visualization (as shown in Section 5).

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

5 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 as shown in the first five lines of Figure 1.

5.1 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:

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

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

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 of the intent. As a consequence, `ext(X)`, the extent, holds for all objects in relation with all attributes of the intent. The second rule is a symmetric definition for the concept's intent. For those familiar to Prolog, note that such a program would lead to an infinite loop. The treatment of loops is a nice feature of ASP that gives access to a fixed-point semantics. 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). To avoid the yield of supremum (infimum), one may include a constraint specifying that extent (intent) must include at least one element.

These models/concepts can be 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). This numbering is arbitrary and serves no other purpose than identifying the different concepts.

5.2 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,_).
3 c(N):- int(N,_).
4 % Ordering of two concepts: the first has all objects of the second.
```

```
5 contains(C1,C2):- c(C1) ; c(C2) ; C1!=C2 ; ext(C1,X): ext(C2,X).
6 % Concepts linked to another in the Galois Lattice.
7 link(C1,C3):- contains(C1,C3) ; not link(C1,C2): contains(C2,C3).
8 % Annotate nodes with extent and intent.
9 annot(upper,X,A):- ext(X,A).
10 annot(lower,X,B):- int(X,B).
```

These lines yield the visualization shown in Figure 4. Line 2 and 3 are here to enable the access to the infinum, supremum and concepts with one atom. Line 5 yields pairs of concepts that are included, based on their extent. Line 7 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 $C3$. Finally, the `annot/3` atoms are a Biseau convention (just as `link/2` that define an edge in the dot output), allowing us to print the extent and intent of each concept, respectively above and below the node.

5.3 Reduced Labelling

The reduced labelling of a lattice is computed as the set of specific objects and attributes for each concept. This is easily defined as `specext/1` and `specint/1` atoms in ASP, using the following lines along the search for formal concepts in section 5.1:

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

With these lines and the collapsing into one model described in section 5.1, we obtain `specext/2` and `specint/2` atoms, describing the AOC poset elements, attached to each concept. We can then compute the reduced labelling of the lattice with the following lines, replacing the previously defined annot/3 definitions in section 5.2:

```
1 % Minimalist annotation of nodes with their extent/intent:
2 annot(upper,X,A):- specext(X,A).
3 annot(lower,X,B):- specint(X,B).
```

Using these definitions, Biseau produces the visualization shown in Figure 5.

6 Pulling Constraints On The Model

This section exposes the implementation in ASP and Biseau of some FCA variants and extensions often used in knowledge processing [19].

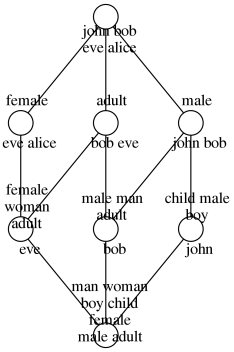

Fig. 4: Visualization of the Galois Lattice of context in Table 1 using Biseau, with extent and intent shown for each node/concept.

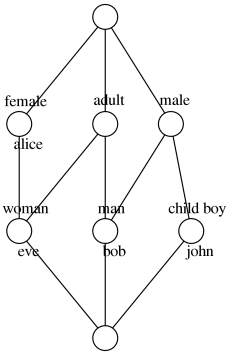

Fig. 5: Visualization of the Galois Lattice of context in Table 1 using Biseau, with reduced labelling.

6.1 Object and Property Oriented Concept Lattices

Following definitions from [23], it is also possible to encode the mining of object oriented concepts (X, Y) defined by $X = Y^{\Diamond}$ and $Y = X^{\Box}$, such as:

$$Y^{\Diamond} = \bigcup_{y \in Y} Ry \qquad X^{\Box} = \{y \in A | Ry \subseteq X\}$$

With $Ry = \{x \in O | (x, y) \in R\}$.

```
% Any object linked to an attribute in the intent is in the extent.
ext(X):- rel(X,Y) ; int(Y).
% Objects in the complementary set of the extent.
not_ext(Nx):- rel(Nx,_) ; not ext(Nx).
% The intent is made of attributes exclusively linked to objects of the extent.
int(Y):- rel(_,Y) ; not rel(Nx,Y): not_ext(Nx).
```

The code for property-oriented concepts is similar, and both replace the encoding in section 5.1.

6.2 Iceberg Lattices

The iceberg lattice, loosely defined as the Galois lattice stripped of all concepts with a too small support (i.e. number of objects in their extents) [22], can be built by discarding any model containing too few objects in his extent. For instance, the Figure 3 of [22], reproduced in this paper in Figure 6, shows the iceberg lattice of the running example MUSHROOMS database of `nbobj` objects with a minimal support of `minsupp`%. It can be reproduced by discarding models using a constraint:

```
% The number of ext/1 atoms must not fall behind the minimal.
:- {ext(_)} < nbobj*minsupp/100.
```

This constraint can be generated by Biseau knowing the number of objects and the minimal support.

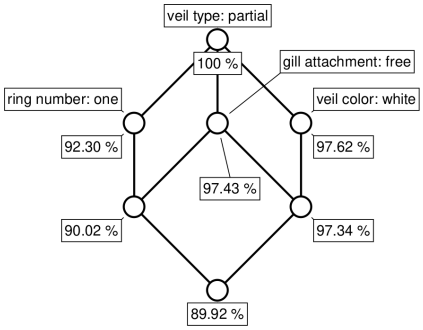

Fig. 6: Iceberg lattice with a minimal support of 85% of the MUSHROOMS database. Figure extracted from [22].

6.3 n-adic FCA

n-adic FCA [16] can be encoded the same way as regular FCA, by extending the number of parameters for `rel` atoms. For instance, in triadic FCA, conditions are given as the third argument of `rel/3` atoms, such as `rel(O,A,C)` is true when the relation between object O, attribute A and condition C holds. Triadic concepts can thus be generated using the following encoding:

```
1 ext(X):- rel(X,_,_) ; rel(X,A,C): int(A), cnd(C).
2 int(X):- rel(_,X,_) ; rel(O,X,C): ext(O), cnd(C).
3 cnd(X):- rel(_,_,X) ; rel(O,A,X): ext(O), int(A).
```

6.4 Pattern Structures

As introduced in [7], a pattern structure is a generalization of FCA applied on attributes structured in semi-lattices. Pattern concepts are pairs of objects and lattices, producing the expected pattern lattice. This technics have been applied to gene expression data [13]. Here, we reproduce the pattern lattice construction for an example of non-binary data from the same publication:

```
1 rel(1,1,5) .  rel(1,2,7) .  rel(1,3,6) .  % 5 objects
2 rel(2,1,6) .  rel(2,2,8) .  rel(2,3,4) .  % 3 situations
3 rel(3,1,4) .  rel(3,2,8) .  rel(3,3,5) .  % one value from 4 to 9
4 rel(4,1,4) .  rel(4,2,9) .  rel(4,3,8) .
5 rel(5,1,5) .  rel(5,2,8) .  rel(5,3,5) .
```

Note that data are encoded in `rel/3` atoms over 5 objects, 3 conditions, and expression values in the interval $[4; 9]$ associated with a given gene and condition,

such as `rel(O,S,V)` holds when object O in situation S has an expression value of V. Similarly to section 5.1, we can enumerate the pattern concepts:

```
1 % Choose a subset of objects as the extent.
2 { ext(O): rel(O,_,_) }.
3 % The intervals of extent.
4 interval(C,Min,Max):- rel(_,C,_) ; Min=#min{V,O: rel(O,C,V), ext(O)} ;
5                       Max=#max{V,O: rel(O,C,V), ext(O)}.
6 % Object is valid on Condition.
7 valid_on(O,C):- rel(O,C,V) ; interval(C,Min,Max) ; Min<=V ; V<=Max.
8 % Object is valid for all Conditions.
9 valid(O):- rel(O,_,_) ; valid_on(O,C): rel(_,C,_).
10 % Avoid any model that do not include maximal number of objects.
11 :- not ext(O) ; valid(O).
```

The use of the meta-programming directives `#min` and `#max` allows us to retrieve the minimal and maximal value associated to the extent. Therefore, `interval(C,Min,Max)` stands for the minimal and maximal values on condition C, e.g. 5 and 6 for condition 1 when extent is $\{1, 2, 5\}$. Unlike the concept model seen in Section 5.1, this model relies on an explicit choice rule for the extent with subsequent constraints to ensure its maximality. Line 2 generates an answer set for each element of the power set of the object set. Following lines will discard answer sets that are not infinum, supremum or concept. Line 4 associate for each condition the minimal and maximal values over the extent. Line 7 selects an object and a condition such as they are associated to a value in the interval. Line 9 selects all objects that are valid for all conditions, and line 11 ensure that they belong to the extent.

The code in section 5.2 can be reused without modifications to produce and show the resulting pattern lattice.

7 Discussion & Conclusion

Using the ASP language in the Biseau environment, some well-known FCA structures (Galois, object-oriented, iceberg, integer pattern lattices) have been reconstructed. 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. By letting the user manipulate the visualization with the full power of ASP, Biseau enables definition of graphs in intension. This gives an abstract access to dot expressions and lets 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.

ASP limits lies into the absence of float numbers handling, and scaling problems inherited from the total grounding of data before solving. However, Potassco

system users may benefit from several extensions of the language like linear programming [18] or propagators [10], allowing one to take advantage of other programming paradigms, or improving performances by an iterative replacement of bottlenecks by dedicated algorithms. For instance, the standard concept mining can be replaced by an implementation of the in-close algorithm [2].

Biseau current state is a very simple proof of concept, and therefore miss a lot of features typically found in Integrated Development Environments, that could help user to write, understand and debug produced ASP code.

Future work will focus on the Biseau generalization : other languages like GML allow to describe graphs, some are field-specific, and some enable outsourcing of the visualization to other (field-)specialized softwares. Future development of Biseau could provide support for ASP advanced features, and embedding of more scripts for FCA and its extensions.

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. 2009.
3. S. Andrews and L. Hirsch. A tool for creating and visualising formal concept trees. In *CEUR Workshop Proceedings*, volume 1637, pages 1–9. Tilburg University, 2016.
4. D. Auber, D. Archambault, R. Bourqui, M. Delest, J. Dubois, A. Lambert, P. Mary, M. Mathiaut, G. Mélançon, B. Pinaud, B. Renoust, and J. Vallet. TULIP 5. pages 1–28, Aug. 2017.
5. L. Bourneuf and J. Nicolas. *FCA in a Logical Programming Setting for Visualization-Oriented Graph Compression*, pages 89–105. Springer International Publishing, Cham, 2017.
6. 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.
7. B. Ganter and S. O. Kuznetsov. Pattern structures and their projections. In *International Conference on Conceptual Structures*, pages 129–142. Springer, 2001.
8. M. Gebser, R. Kaminski, B. Kaufmann, , M. Lindauer, M. Ostrowski, J. Romero, T. Schaub, and S. Thiele. *Potassco User Guide*, 2015.
9. M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and M. Schneider. Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):107–124, 2011.
10. M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and P. Wanko. Theory Solving Made Easy with Clingo 5. In M. Carro, A. King, N. Saeedloei, and M. D. Vos, editors, *Technical Communications of the 32nd International Conference on Logic Programming (ICLP 2016)*, volume 52 of *OpenAccess Series in Informatics (OASIcs)*, pages 2:1–2:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
11. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In *Proc. of the 7th International Conf. on Logic Programming (ICLP)*, pages 579–97, 1990.
12. 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. Springer-Verlag, 2006.

13. M. Kaytoue, S. O. Kuznetsov, A. Napoli, and S. Duplessis. Mining gene expression data with pattern structures in formal concept analysis. *Information Sciences*, 181(10):1989 – 2001, 2011. Special Issue on Information Engineering Applications Based on Lattices.
14. L. L. Kis, C. Sacarea, and D. Troanca. Fca tools bundle-a tool that enables dyadic and triadic conceptual navigation. *Proc. of FCA4AI*, 2015.
15. B. Lahcen and L. Kwuida. Lattice miner: a tool for concept lattice construction and exploration. In *Suplementary Proceeding of International Conference on Formal concept analysis (ICFCA'10)*, 2010.
16. F. Lehmann and R. Wille. A triadic approach to formal concept analysis. In G. Ellis, R. Levinson, W. Rich, and J. F. Sowa, editors, *Conceptual structures: applications, implementation and theory*, number 954 in Lecture Notes in Artificial Intelligence, pages 32–43, Berlin–Heidelberg–New York, 1995. Springer–Verlag.
17. V. Lifschitz. What is answer set programming? In *Proc. of the 23rd National Conf. on Artificial Intelligence - Vol. 3*, AAAI'08, pages 1594–97. AAAI Press, 2008.
18. G. Liu, T. Janhunen, and I. Niemelä. Answer set programming via mixed integer programming. In *KR*, pages 32–42, 2012.
19. J. Poelmans, D. I. Ignatov, S. O. Kuznetsov, and G. Dedene. Formal concept analysis in knowledge processing: A survey on applications. *Expert Systems with Applications*, 40(16):6538 – 6560, 2013.
20. 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.
21. P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, and D. e. a. Ramage. Cytoscape: A software environment for integrated models of biomolecular interaction networks. *Genome Research*, 13(11):2498–2504, 2003.
22. G. Stumme, R. Taouil, Y. Bastide, and L. Lakhal. Conceptual clustering with iceberg concept lattices. In *In: Proc. of GI-Fachgruppentreffen Maschinelles Lernen'01, Universität Dortmund*, 2001.
23. Y. Yao. Concept lattices in rough set theory. In *Fuzzy Information, 2004. Processing NAFIPS'04. IEEE Annual Meeting of the*, volume 2, pages 796–801. IEEE, 2004.