A logic-based approach to compute a direct basis from implications

P. Cordero, M. Enciso, A. Mora, M. Ojeda-Aciego, E. Rodríguez-Lorenzo,
amora@ctima.uma.es
Universidad de Málaga

September 19, 2014

Abstract

Formal Concept Analysis is an emergent area in the topic of data analysis based on lattice theory. In this framework, a context is defined as the relation between a set of objects and a set of attributes and from here it is possible to extract relevant knowledge. One of the important topics is to study the implications between the attributes considered. In a context, some equivalent sets of implications can be compute using different techniques. We are studying the direct optimal basis, which enables us to compute the closure of a set of attributes in just one iteration. A Prolog method has been implemented that computes a direct basis from a set of implications.

1 Preliminaries

Formal Concept Analysis (FCA) considers a formal context as the relationship between a set of objects and a set of attributes. Formally, this relation can be defined as follows:

Definition 1.1 *Let G be a set of objects, M a set of attributes, and $I \subseteq G \times M$ a binary relation between G and M, then the triple $\mathbf{K} = (G, M, I)$ is called a formal context.*

In this triple, I is a binary relation between G and M such that, for $o \in G$ and $a \in M$, $o\ I\ a$ means that the object o has the attribute a. Two mappings are defined:

- $(\ )' \colon 2^G \to 2^M$ is defined for all $A \subseteq G$ as $A' = \{m \in M \mid g\ I\ m \text{ for all } g \in A\}$.
- $(\ )' \colon 2^M \to 2^G$ is defined for all $B \subseteq M$ as $B' = \{g \in G \mid g\ I\ m \text{ for all } m \in B\}$.

These two mappings are closure operators and their fixpoints are the so-called *formal concepts*.

Definition 1.2 *A **formal concept** is a pair (A, B) such that $A \subseteq G$, $B \subseteq M$, $A' = B$ and $B' = A$. Consequently, A and B are closed sets of objects and attributes, respectively called extent and intent.*

If the following partial ordering

$$(A_1, B_1) \leq (A_2, B_2) \text{ if and only if } A_1 \subseteq A_2 \text{ (or equivalently } B_1 \supseteq B_2)$$

is considered in the set of all the formal concepts, then this set has the structure of a complete lattice, the **concept lattice associated to the context**.

In FCA, the study of attribute implications is an important topic that allows us to capture all the information which can be deduced from a context. In fact, these implications summarize the semantics of the context. They are expressions of the form $A \to B$, where A and B are sets of attributes. A context *satisfies* the implication $A \to B$ if every object that has all the attributes from A also has all the attributes from B.

Definition 1.3 *An (attribute) implication of a formal context $\mathbf{K} = (G, M, I)$ is defined as a pair (A, B), written $A \to B$, where $A, B \subseteq M$. Implication $A \to B$ holds (is valid) in $\mathbf{K}$ if $A' \subseteq B'$.*

One can find in the literature a lot of works focused on how to compute the attribute implications, and specially on how to improve the implications obtained [9]. A *basis* of a set of implications is a basis of least cardinality. Some approaches compute the Duquenne-Guigues basis [7], focusing on the minimality of the cardinality of the set of implications. In [5], we have developed a method to remove redundant attributes in this kind of minimal basis. In [6] a new logic-based method to obtain a basis with minimal size in the left-hand side of the implications was proposed.

We emphasize the survey proposed by Bertet and Monjardet in [3]. In this work, the authors studied five unit implicational systems obtained by different authors in different fields and show that these formalisms are, in fact, identical. Between these definitions we outline the following:

Definition 1.4 *A set of implications, Σ, is said to be minimal or non-redundant if, for all $X \to Y \in \Sigma$, the set $\Sigma \setminus \{X \to Y\}$ is not equivalent to Σ.*

Definition 1.5 *A set of implications, Σ, is called* minimum set *if $|\Sigma| \leq |\Sigma'|$ for all set of implications Σ' equivalent to Σ.*

Definition 1.6 *A set of implications, Σ, is said to be* optimal *if $||\Sigma|| \leq ||\Sigma'||$ for all set of implications Σ' equivalent to Σ, where the size of Σ is defined as*

$$||\Sigma|| = \sum_{\{X \to Y \in \Sigma\}} (|X| + |Y|)$$

$\Sigma = \{X_0 \rightarrow Y_0, \ldots X_n \rightarrow Y_n\}$ be a set of implications, it is said a left-minimal basis if there does not exist a $X_i \rightarrow Y_i$ and a subset $X_i' \subsetneq X_i$ such that $\Sigma \setminus \{X_i \rightarrow Y_i\} \cup \{X_i' \rightarrow Y_i\}$ is equivalent to Σ.

One of the main problems in FCA is the computation of the closure of a set of attributes. Interesting approaches to this problem can be seen in [2, 3, 8].

Definition 1.7 *A set of implications, Σ, is said to be* direct *if the computation of the closure with respect to Σ of any set X of attributes requires just one iteration, that is, a unique traversal of the set of implications.*

Obviously, the direct-optimal property is the combination of the directness and optimality properties and a method to obtain a direct-optimal basis is proposed in [2].

In this paper, we introduce a Prolog implementation based on the Simplification Logic proposed in [4] to compute a direct optimal basis from a set of implications. The use in FCA of the logic programming paradigm, via the Prolog language, is explored and proposed as a framework to develop fast prototypes in which the methods based in logic are implemented in a direct way.

2 Simplification logic and closures

In [4], the Simplification Logic for Functional Dependencies (i.e. implications) called $\mathbf{SL}_{FD}$, was proposed. This logic is equivalent to well-known Armstrong's Axioms [1]. The main difference is that $\mathbf{SL}_{FD}$ avoids the use of transitivity, and is guided by the idea of simplifying the set of functional dependencies by efficiently removing redundant attributes.

$\mathbf{SL}_{FD}$ considers reflexivity as an axiom scheme, together with the following inferences rules:

$$\texttt{[Ref]}\ \frac{A \supseteq B}{A \rightarrow B}$$

$$\texttt{[Frag]}\ \frac{A \rightarrow B \cup C}{A \rightarrow B} \qquad \texttt{[Comp]}\ \frac{A \rightarrow B,\ C \rightarrow D}{A \cup C \rightarrow B \cup D} \qquad \texttt{[Simp]}\ \frac{A \rightarrow B,\ C \rightarrow D}{A \cup (C \smallsetminus B) \rightarrow D}$$

If we have a set of implications Σ and a set of attributes A, the closure of A in $\mathbf{SL}_{FD}$ is defined as the maximum set of attributes A^+ such that $\Sigma \vdash A \rightarrow A^+$.

Theorem 2.1 *Let $\mathbf{K} = (G, M, I)$ be a formal context and Σ a basis for $\mathbf{K}$. For all $A \subseteq M$, the equality $A^+ = A''$ holds.*

We introduced an efficient algorithm to compute the closure of a set of attributes which improves the classical closure algorithms in [8], where the interested reader can find all the details of the algorithm, which was implemented in Prolog as the predicate `closure`. We show an example about the execution in Prolog with this predicate.

Example 2.1 *Let $\Omega = a, b, c, d, e, f$ a set of attributes. We would like to compute the closure of $A = \{a\}$.$\Sigma = \{ab \rightarrow c,\ bd \rightarrow d, de \rightarrow f, ce \rightarrow f\}$.*

```
? closure([a],[ implication([a], [b,c]),  implication([a,b], [e]),
                implication([c], [f]),  implication([c,e], [g])).

% the result is
?  ([a,b,c,e,g]).
   True.
```

The set of attributes $\{a, b, c, e, g\}$ *is the closure of* $\{a\}$.

3 Computing a direct optimal basis

In this section, we show how the $\mathbf{SL}_{FD}$ is the main foundation of the new method to compute the direct optimal basis from a set of implications. As we stated before, the best choice to make the implementation is Prolog, since the method is logic-based.

The input of the Prolog program is a set of attributes M and a set of implications Σ over the attributes in M. The output is the direct optimal basis equivalent to this set of implications.

The main predicate of the method developed is

$$directoptimalSL(ImplicationsInput, DOBasisOutput)$$

which receives the set of implications from the input file `ImplicationsInput`, and renders the direct optimal basis, which equivalent to the input, in the output file `DOBasisOutput`, .

There are two main operations in the method:

- `applyCompositionSimplification` is a predicate which applies exhaustively the rule composition-simplification to any pair of implications to obtain a direct basis.
- `applySimplification` is a predicate that applies exhaustively the rules of $\mathbf{SL}_{FD}$ to remove redundancy in the direct basis obtained in the previous step in order to get a direct optimal basis.

Below, we outline the description of the Prolog method `applySimplificationLogic` which calls these two main predicates.

```
applySimplificationLogic:-
    fixPoint_Non,
    applyCompositionSimplification,
    removeRedundancy,!.
```

The predicate `applyCompositionSimplification` is applied exhaustively until the fixpoint is reached. To begin with, the method collects all the implications in a List, called `ListImplications`, and invokes the predicate `compositionsimplificationrule` that applies the rule to any pair of implications.

```
applyCompositionSimplification:-
    fixpoint(no),
    fixPoint_Yes,
    findall([implication(X,Y)],implication(X,Y),ListImplications),
    compositionsimplificationrule(ListImplications),
    applyComposition,
    applyCompositionSimplification.

applyCompositionSimplification.


compositionsimplificationrule([]):-!.

compositionsimplificationrule([implication(X,Y)|Rest]):-
    compositionsimplification([X,Y],Rest),
    compositionsimplificationrule(Rest),!.

compositionsimplification([_,_],[]):-!.

compositionsimplification(implication(A,B),[implication(C,D)|Rest]):-
     cs(implication(A,B),implication(C,D)),
     cs(implication(C,D),implication(A,B)),
     compositionsimplification(implication(A,B),Rest).
```

where the predicate *cs* applies the following rule derived from the $\mathbf{SL}_{FD}$:

$$\texttt{cs}(implication(A,B), implication(C,D)) = implication(A \cup C \setminus B, D \setminus (A \cup B)\}$$

When an implication is added in certain step of the previous algorithm, the flag `fixpoint` takes the value `false` in order to repeat the method again.

The second step is the exhaustive application of the rules of $\mathbf{SL}_{FD}$ to remove redundancy [4]. It has been implemented with the predicate `applySimplification`.

```
applySimplification:-
    lrSimplificationRule,!,
    applySimplification.
applySimplification.

% lrSimplification Rule:   X → Y, Z → U to X → Y, Z-Y → U-Y
%    Z-Y → U-Y  if X included in Z
%    Z   → U    another case
```

Example 3.1 *In this example, we will compute the direct basis of the following set of implications stored in the file* `ganter.txt`*:*

```
implication([a],[b]).
implication([a],[c]).
implication([d],[b]).
```

```
implication([c],[b]).
implication([a,b,c,d],[e]).
implication([a,b,c,d],[g]).
implication([a,b,c,e],[d]).
implication([a,b,c,e],[g]).
```

We call the Prolog predicate:

```
directoptimalSL('ganter.txt','Output_ganter.txt').
```

and it renders the direct optimal basis as follows:

```
-- INPUT --
:- dynamic implication2/2.
→ Preparing the input for Simplification:
→ Reduction:
→ Composition:
 implication([a,b,c,e],[g]) FD removed
 implication([a,b,c,e],[d]) FD removed
 implication([a,b,c,e],[d,g]) FD added
→ Composition:
 implication([a,b,c,d],[g]) FD removed
 implication([a,b,c,d],[e]) FD removed
 implication([a,b,c,d],[e,g]) FD added
→ Composition:
 implication([a],[c]) FD removed
 implication([a],[b]) FD removed
 implication([a],[b,c]) FD added

-- Implications composed and reduced:
:- dynamic implication/2.
implication([c], [b]).
implication([d], [b]).
implication([a, b, c, e], [d, g]).
implication([a, b, c, d], [e, g]).
implication([a], [b, c]).

**    First Step   **
→ Equivalence  -   Composition + Simplification:
        implication([c],[b]) + implication([a,b,c,e],[d,g]) |---
        implication([a,c,e],[d,g]) FD added

→ Equivalence  -   Composition + Simplification:
        implication([c],[b]) + implication([a,b,c,d],[e,g]) |---
        implication([a,c,d],[e,g]) FD added
→ Equivalence  -   Composition + Simplification:
        implication([d],[b]) + implication([a,b,c,e],[d,g]) =
        implication([a,c,d,e],[g]) not added  implication([a,c,e],[d,g])
→ Equivalence  -   Composition + Simplification:
       implication([a],[b,c]) + implication([a,b,c,e],[d,g]) |---
       implication([a,e],[d,g]) FD added
→ Equivalence  -   Composition + Simplification:
```

```
      implication([a],[b,c]) + implication([a,b,c,d],[e,g]) |---
      implication([a,d],[e,g]) FD added

**    Second Step **
.......
.......
***  END Composition + Simplification ( A direct basis) **

implication([c], [b]).
implication([d], [b]).
implication([a, b, c, e], [d, g]).
implication([a, b, c, d], [e, g]).
implication([a], [b, c]).
implication([a, c, e], [d, g]).
implication([a, c, d], [e, g]).
implication([a, e], [d, g]).
implication([a, d], [e, g]).

*** BEGIN   Simplification: Removing redundancy **
→Equivalence  -   Simplification and Simplification + Axiom:
   implication([c],[b]) + implication([a,b,c,e],[d,g]) |---
   implication([a,b,c,e],[d,g]) Implication removed
   implication([a,c,e],[d,g]) yet exist

→ Equivalence  -   Simplification and Simplification + Axiom:
  implication([c],[b]) + implication([a,b,c,d],[e,g]) |---
  implication([a,b,c,d],[e,g]) Implication removed
  implication([a,c,d],[e,g]) yet exist

→ Equivalence  -   Simplification and Simplification + Axiom:
  implication([a],[b,c]) + implication([a,c,e],[d,g]) |---
  implication([a,c,e],[d,g]) Implication removed
  implication([a,e],[d,g]) yet exist

→ Equivalence  -   Simplification and Simplification + Axiom:
  implication([a],[b,c]) + implication([a,c,d],[e,g]) |---
  implication([a,c,d],[e,g]) Implication removed
  implication([a,d],[e,g]) yet exist

** OUTPUT: DIRECT OPTIMAL BASIS

implication([c], [b]).
implication([d], [b]).
implication([a], [b, c]).
implication([a, e], [d, g]).
implication([a, d], [e, g]).
```

4 Conclusions

We have presented a Prolog implementation of a novel method to compute the direct optimal basis from a set of implications. The soundness and correctness of the method will be included in a extended version of this paper. As future work, we are planning a comparison with other methods in the literature.

Acknowledgements

Supported by grants TIN12-39353-C04-01 and TIN2011-28084 of the Science and Innovation Ministry of Spain, co-funded by the European Regional Development Fund (ERDF).

References

[1] W.W. Armstrong, Dependency structures of data base relationships, Proc. IFIP Congress, pp. 580–583, 1974.

[2] K. Bertet, M. Nebut, *Efficient algorithms on the Moore family associated to an implicational system*, DMTCS, 6(2): 315–338, 2004.

[3] K. Bertet, B. Monjardet, *The multiple facets of the canonical direct unit implicational basis*, Theor. Comput. Sci., 411(22-24): 2155–2166, 2010.

[4] P. Cordero, M. Enciso, A. Mora, I.P, de Guzmán: SLFD logic: Elimination of data redundancy in knowledge representation, LNCS 2527: 141–150, 2002.

[5] P. Cordero, M. Enciso, A. Mora, M. Ojeda-Aciego, *Computing Minimal Generators from Implications: a Logic-guided Approach*, CLA 2012: 187–198, 2012.

[6] P. Cordero, M. Enciso, A. Mora, M. Ojeda-Aciego, *Computing Left-Minimal Direct Basis of implications*, CLA 2013: 293–298, 2013.

[7] J.L. Guigues and V. Duquenne, Familles minimales d'implications informatives résultant d'un tableau de données binaires. Math. Sci. Humaines, 95, 5–18, 1986.

[8] A. Mora, P. Cordero, M. Enciso, I.Fortes, Closure via functional dependence simplification, International Journal of Computer Mathematics, 89(4): 510–526, 2012.

[9] K. Nehmé, P. Valtchev, M. H. Rouane, R. Godin, On Computing the Minimal Generator Family for Concept Lattices and Icebergs, LNCS 3403: 192–207, 2005.