

Formal Concept Analysis in Software Engineering

Paolo Tonella

ITC-irst

Centro per la Ricerca Scientifica e Tecnologica

38050 Povo (Trento), Italy

tonella@itc.it

Abstract

Given a binary relationship between objects and attributes, concept analysis is a powerful technique to organize pairs of related sets of objects and attributes into a concept lattice, where higher level concepts represent general features shared by many objects, while lower level concepts represent the object-specific features. Concept analysis was recently applied to several software engineering problems, such as: restructuring the code into more cohesive components, identifying class candidates, locating features in the code by means of dynamic analysis, reengineering class hierarchies. This tutorial provides the background knowledge required by such applications. Moreover, the methodological issues involved in the different applications of this technique are considered by giving a detailed presentation of three of them: module restructuring, design pattern inference and impact analysis based on decomposition slicing. The tutorial is concluded by an overview on other kinds of applications.

1. Introduction

Concept analysis, originally investigated by B. Birkhoff [1] in 1940 and more recently elaborated by Ganter and Wille [3], is a very general method to analyze a binary relationship between arbitrary objects and attributes. Its output is a lattice of so-called concepts, which offers non trivial insights into the structure underlying the original relationship. Each lattice node (concept) contains maximal sets of objects sharing common attributes. The hierarchy of concepts in the lattice can be interpreted as the possibility to generalize or specialize a concept. In fact, higher concepts constrain the objects to share less attributes, while lower concepts contain less objects with more numerous and more specific attributes.

In the analysis of software systems, several relationships among the composing entities emerge. For this reason, concept analysis found a very productive application area in software engineering. Static and dynamic relationships among software components can be subjected to concept analysis to obtain information useful during maintenance, for program comprehension, and in the execution of reengineering tasks.

2. Objective

The objective of this tutorial is to provide background and methodological knowledge on concept analysis and on its usage in software engineering. This is achieved by describing some recent, representative applications of concept analysis in detail: while it is not possible to concentrate the wide range of applications developed so far in a single tutorial, a thorough presentation of three examples offers the possibility to deeply investigate the involved methodological research issues as well as the practical implications in the usage of the considered technique. A basic background in concept analysis is also given for participants not familiar at all with this method. Finally, a quick survey of the most important types of applications not covered by the three selected ones is provided to complete the tutorial.

3. Structure

First of all, a common background on the mathematical framework behind concept analysis is established. Formal definitions of concepts and concept lattice are given, and their main properties are enumerated and commented. The algorithmic implications of the formal definitions are also considered. The material presented in this part of the tutorial is based on the book by Ganter and Wille [3].

Then, three examples of applications in the domain of software analysis are considered [6, 7, 8]. They concern respectively the reorganization of a legacy system into cohesive units [6], the inference of recurring architectural pat-

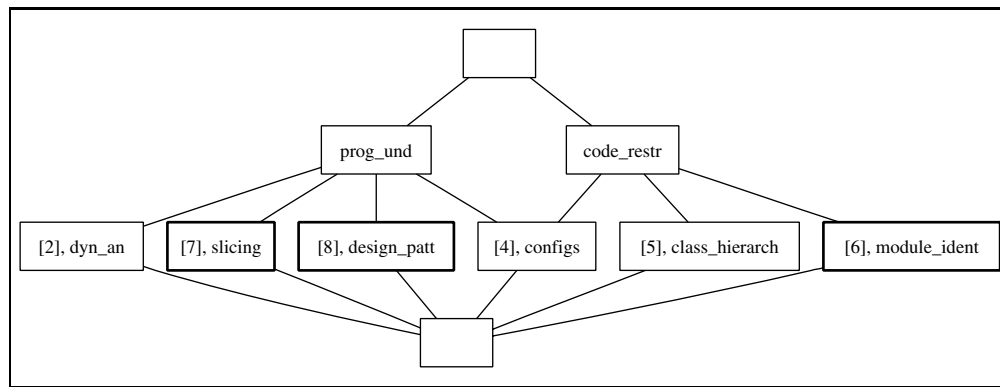


Figure 1. Concept lattice showing the relationship between applications and research areas.

terns without any usage of a priori information [8], and the decomposition of a software system into computational units (decomposition slices), that may be strongly dependent, weakly dependent or independent with each other [7].

There are other application areas, that are quite disjoint from the three selected for deeper presentation and in which important and interesting results have been produced. The final part of the tutorial is devoted to them. Namely, reengineering of class hierarchies [5], feature location by means of dynamic analysis [2], and derivation of the configuration structure of a software system [4], are briefly presented. For each of them, the summary description is focused on the problem to which concept analysis is applied, the way it is used, and the solution it produces.

4. Concept analysis of the tutorial

In total, six examples of applications of concept analysis are given [6, 8, 7, 5, 2, 4]. Three of them [6, 8, 7] are presented in detail, while the others [5, 2, 4] are just surveyed. All of them deal with problems encountered during software maintenance, such as code restructuring (*code_restr*) and program understanding (*prog_und*). However, they consider different aspects of such problems, ranging from module identification (*module_ident*), to the reengineering of class hierarchies (*class_hierarch*), to the dynamic analysis (*dyn_an*) and the alternative software configurations (*configs*). Correspondingly, the starting context to which concept analysis is applied differs remarkably from case to case, ranging from the relationship between functions and data structures, to that between variables and object members. The relationship between execution scenarios and computational units and that between macro directives and code fragments are also considered in some of these applications.

From the association between each selected application and the related research areas, the lattice in Figure 1 can be obtained by means of concept analysis.

The two concepts immediately below the top represent the two broad categories of *program understanding* and *code restructuring*. Five works fall mainly below just one of these two categories, while [4] belongs to both. With respect to the more specific categories, each paper belongs to a concept identified by a single research area. This indicates that the considered works are quite orthogonal with each other, thus giving a good coverage of the application types. More works are available in the literature within each specific category. Concepts with thicker borders refer to the works that are presented in more detail during the tutorial.

References

- [1] G. Birkhoff. *Lattice Theory*. American Mathematical Society Colloquium Publications 25, Providence, RI, USA, 1st ed. 1940.
- [2] T. Eisenbarth, R. Koschke, and D. Simon. Locating features in source code. *IEEE Transactions on Software Engineering*, 29(3):195–209, March 2003.
- [3] B. Ganter and R. Wille. *Formal Concept Analysis*. Springer-Verlag, Berlin, Heidelberg, New York, 1996.
- [4] M. Krone and G. Snelting. On the inference of configuration structures from source code. In *Proc. of the 16th International Conference on Software Engineering*, pages 49–57, Sorrento, Italy, May 1994.
- [5] G. Snelting and F. Tip. Reengineering class hierarchies using concept analysis. *ACM Transactions on Programming Languages and Systems*, 22(3):540–582, May 2000.
- [6] P. Tonella. Concept analysis for module restructuring. *IEEE Transactions on Software Engineering*, 27(4):351–363, April 2001.
- [7] P. Tonella. Using a concept lattice of decomposition slices for program understanding and impact analysis. *IEEE Transactions on Software Engineering*, 29(6):495–509, June 2003.
- [8] P. Tonella and G. Antoniol. Inference of object oriented design patterns. *Journal of Software Maintenance*, 13(5):309–330, 2001.