

A Formal Method for the Analysis of Product Maps

Thomas Eisenbarth Rainer Koschke Daniel Simon
Universität Stuttgart
Breitwiesenstraße 20–22
70565 Stuttgart, Germany
`{eisenbarth,koschke,simon}@informatik.uni-stuttgart.de`

Abstract

During the initiation and evolution of a software product line, developers make use of *product maps* [3] for scoping and requirements engineering. We present how to utilize a formal mathematical method for analyzing the product maps for an anticipated software product line. The result of the method reveals variabilities and commonalities of products, as well as dependencies between products.

1 Introduction

During the initiation phase of a software product line, developers build a product map [3], i.e., a table that contains the features of the products of the product line. The manual analysis of a large product map is difficult. In this paper, we describe a mathematically founded method that helps in understanding the provided facts and the relations between individual products and features of the software product line.

2 Approach: Formal Concept Analysis

This section describes how formal concept analysis can be used to automatically identify commonalities and variabilities in product maps.

Formal concept analysis is a mathematical technique for analyzing binary relations (in our case, this is the product map). The mathematical foundation of concept analysis was laid by Birkhoff [2] in 1940. For a full description of the mathematical background of formal concept analysis, we refer to [6].

The product map lists the set of features for each product. It is basically a binary relation $\mathcal{M} \subseteq \mathcal{P} \times \mathcal{F}$ between a set of products \mathcal{P} and a set of features \mathcal{F} . Figure 1 shows a product map.

Based on the product map, the *common features* of a set of products $P \subseteq \mathcal{P}$ can be identified as $\sigma(P)$:

$$\sigma(P) = \{f \in \mathcal{F} \mid (p, f) \in \mathcal{M} \text{ for all } p \in P\} \quad (1)$$

For instance, the common features of *Siemens* and *Casio* in Fig. 1 are *mail*, *sms*, and *infra*.

Products	Features						
	mail	wap	sms	light	blue	infra	serial
Siemens	×	×	×			×	
Casio	×		×	×		×	
Handspring	×			×		×	×

Figure 1: A small product map.

Analogously, the set of *common products* $\tau(F)$ that possess a given set of features $F \subseteq \mathcal{F}$ can be described as:

$$\tau(F) = \{p \in P \mid (p, f) \in \mathcal{M} \text{ for all } f \in F\} \quad (2)$$

E.g., the products that share features *sms* and *light* in Fig. 1 are *Casio* and *Handspring*.

In product maps, we are interested—amongst other things—in sets of products that have all features in common. This leads us to the notion of *concept* in formal concept analysis. A pair $c = (P, F)$ is called *concept* iff $F = \sigma(P)$ and $P = \tau(F)$, i.e., every product has all features and every feature is offered by every product in this concept. Intuitively, this definition gives us a maximally large rectangle of filled product map cells, where row and column permutations are allowed. For example, $C1 = (\{Handspring, Casio\}, \{mail, light, infra\})$ and $C2 = (\{Handspring\}, \{mail, sms, light, infra, serial\})$ form concepts in Fig. 1.

Comparing the two concepts C1 and C2, we notice that the products in C2 are a subset of the products in C1, and the features in C1 are a subset of the features in C2. We state that concept C1 is more general than concept C2 because it has fewer features. Because fewer features are required, more products offer these features. Formally, we can define this relationship between two concepts \leq as follows:

$$(P_1, F_1) \leq (P_2, F_2) \Leftrightarrow P_1 \subseteq P_2 \quad (3)$$

or, dually, by

$$(P_1, F_1) \leq (P_2, F_2) \Leftrightarrow F_1 \supseteq F_2 \quad (4)$$

If we have $c_1 \leq c_2$, then c_1 is called a *subconcept* of c_2 and c_2 is called *superconcept* of c_1 . The relationship \leq is only a partial order because some concepts are incomparable. For example, C2 and C5 = $(\{Siemens\}, \{mail, wap, sms, infra\})$ are incomparable.

The set \mathcal{L} of all concepts and the partial order \leq form a complete lattice

$$\mathcal{L}(C) = \{(P, F) \in 2^{\mathcal{P}} \times 2^{\mathcal{F}} \mid F = \sigma(P) \text{ and } P = \tau(F)\} \quad (5)$$

which is called *concept lattice*. It can be depicted as a directed acyclic graph whose nodes represent the concepts and whose edges denote the relation \leq . For instance, Fig. 2(a) shows the lattice derived from the product map in Fig. 1.

To improve the legibility, we use the *sparse representation* of the lattice. Each product is attached to the unique concept c determined by $c = s(p)$, where $s(p)$ computes the

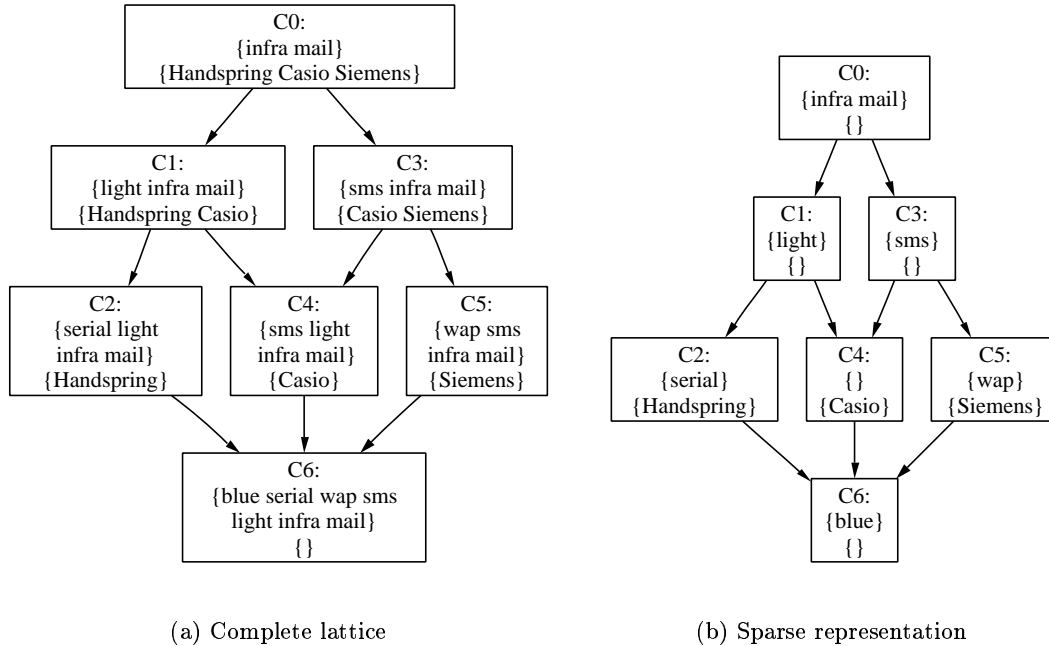


Figure 2: Concept lattices for the product map in Fig. 1.

largest (with respect to \leq) concept containing p . Likewise, each feature f is attached to the unique concept determined by $c = s(f)$, where $s(f)$ computes the smallest (with respect to \leq) concept containing f . Figure 2(b) shows the sparse representation of the lattice in Fig. 2(a). Note that, if required, the complete representation can be retrieved from the sparse representation: all features of a product p lie on the paths from $s(p)$ to the top concept. Further, all products offering a feature f can be found on the paths from $s(f)$ to the bottom concept.

Benefits from concept analysis

The concept lattice is a precise and semantically equivalent representation of the original product map, i.e., there is exactly one lattice that corresponds to the product map and vice versa—modulo row and column permutations. In other words, the product map and the lattice are two different views on the same relation. However, relationships between products and features are more obvious in the lattice than in the product map. This can be illustrated by comparing the product map in Fig. 3 to the corresponding lattice in Fig. 4. The product map in Fig. 3 lists the features of a number of PDAs¹.

We can derive the following facts more easily (i.e., automatically) from the lattice than from the table representation:

¹PDA stands for Personal Digital Assistant. The product map is offered by <http://www.tariftip.de/default.asp?HB=1&BID=6&GID=82>. The web site is intended as a product comparison. We preselected "price less than 500€" and "infrared interface".

Product	mail	wap	sms	light	blue	infra	serial	usb	gsm	speaker	micro	line	module
Siemens	×	×	×			×							×
Oregon				×		×	×						
Palm m105	×	×	×	×		×	×						
Casio	×		×	×		×							
IBM c3	×			×		×	×						
Agenda	×			×		×	×			×	×		×
Palm m500	×		×	×		×		×					×
Handspring	×			×		×	×	×		×	×		×
IBM c500	×			×		×		×					
Ericsson	×	×	×	×		×	×	×	×				

Figure 3: A product map for a selected number of PDAs

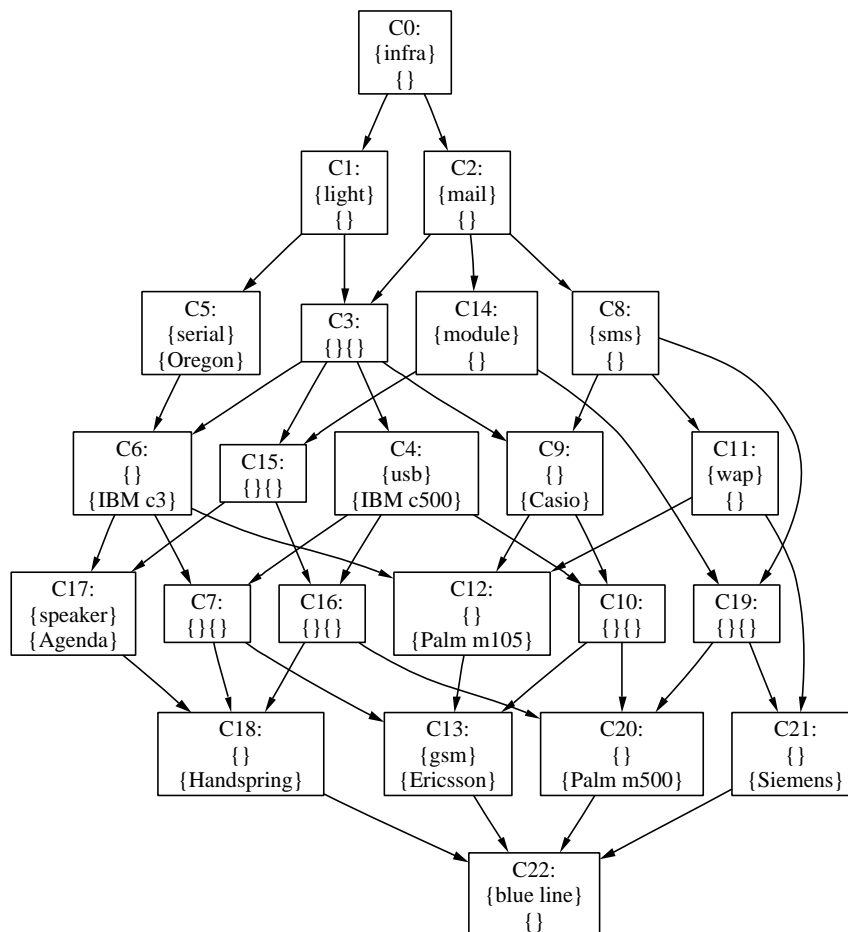


Figure 4: Full concept lattice for the product map in Fig. 3.

Shared features. Commonalities among a set of products $\{p_1, p_2, \dots, p_n\}$ can be identified as the features of the concept that is the nearest common superconcept of the concepts $s(p_i)$. E.g., The nearest common superconcept of *IBM c3*, *IBM c500*, and *Casio* is *C3* in Fig. 4 and, hence, the shared features are *light*, *mail*, and *infrared*.

Distinct features. Differences between a set of products $\{p_1, p_2, \dots, p_n\}$ are features of the superconcepts of the concepts $s(p_i)$ not identified as shared features (as described above). E.g., the distinct feature of *IBM c500* w.r.t. *IBM c3* and *Casio* is its USB port.

Feature-offering products. For a given set of features $\{f_1, f_2, \dots, f_n\}$ the products that offer these features can be identified as the products of the concept that is the nearest common subconcept of the nodes $s(f_i)$. E.g., the product that offers a *speaker* (*C17*) and a *USB port* (*C4*) is *Handspring* attached to *C18*.

Potential feature implications. Feature f_1 potentially implies feature f_2 if every product offering feature f_2 also offers f_1 : $f_1 \rightarrow f_2$. This is the case if $s(f_2) \leq s(f_1)$. E.g., a *speaker* potentially implies a *serial port* according to Fig. 4. The example shows that the implication may be spurious and needs to be checked by an expert. Inversely, one can, however, automatically check an expert's assertion about a feature implication based on the lattice.

Feature categorization. The lattice reflects the specificity of features: The lower a feature in the lattice, the more special it is because fewer products offer it. Features that are common to all products will appear at the top concept. E.g., all products offer an infrared interface (this was the original selection criterion), but only *Ericsson* offers *GSM*. Note that features that are not offered by any product are at the bottom element of the concept lattice and no product will be attached to the bottom element. E.g., *bluetooth* and *line* are features offered by no product in the given price range.

Product categorization. Analogously to features, the lattice reflects the capability degree of products: The lower a product in the lattice, the more capable it is because it offers more features. E.g., *Ericsson*, *Handspring*, *Palm m500*, and *Siemens* are the most capable PDAs.

3 Related Work

Snelting [9] introduced formal concept analysis to software engineering. Since then, there is a growing number of applications of formal concept analysis. We used it to analyze features in existing code [4]. Based on that, we proposed a guided asset mining process [5] and a process for incrementally shifting from monolithic software to software product lines [8].

The product maps stem from PuLSETM-Eco [3], where they are used for scoping. Our method helps in the analysis of these maps.

4 Conclusions and Future Work

We presented a mathematical method to analyze product maps. The method is intuitive and easy to apply. The mathematical details can be hidden from the user of the method. It is also easy to implement our approach. Our prototypical implementation combines existing tools. The concept analysis is performed by Lindig's tool [7] and the lattices are visualized by AT&T's GraphViz [1]. To connect these tools, some simple Perl glue code was sufficient.

Future research will address transferring the already elaborated parts of the theory dealing with multi-value features (rather than "feature present or not"). Further, to provide support for evolution in product lines, we are working on an incremental application of our method.

References

- [1] AT&T Labs-Research. GraphViz. Available at <http://www.research.att.com/sw/tools/graphviz/>, 2002.
- [2] Garret Birkhoff. *Lattice Theory*. American Mathematical Society Colloquium Publications 25, Providence, RI, USA, first edition, 1940.
- [3] Jean-Marc DeBaud and Klaus Schmid. A Systematic Approach to Derive the Scope of Software Product Lines. In *Proceedings of the 21st International Conference on Software Engineering*, pages 34–43, Los Angeles, CA, USA, May 1999. IEEE Computer Society Press.
- [4] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Aiding Program Comprehension by Static and Dynamic Feature Analysis. In *Proceedings of the International Conference on Software Maintenance*, pages 602–611, Florence, Italy, November 2001. IEEE Computer Society Press.
- [5] Thomas Eisenbarth and Daniel Simon. Guiding Feature Asset Mining for Software Product Line Development. In *Proceedings of the International Workshop on Product Line Engineering: The Early Steps: Planning, Modeling, and Managing*, pages 1–4, Erfurt, Germany, September 2001. Fraunhofer IESE.
- [6] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis—Mathematical Foundations*. Springer, 1999.
- [7] Christian Lindig. Concepts 0.3e. Available at <http://www.gaertner.de/~lindig/software/>, 1999.
- [8] Daniel Simon and Thomas Eisenbarth. Evolutionary Introduction of Software Product Lines. In *Proceedings of the 2nd Software Product Line Conference*, San Diego, CA, USA, August 2002. Springer. To appear.
- [9] Gregor Snelting. Reengineering of Configurations Based on Mathematical Concept Analysis. *ACM Transactions on Software Engineering and Methodology*, 5(2):146–189, April 1996.