UNIVERSITÄT AUGSBURG
Universitätsbibliothek

OPEN ACCESS
OPUS AUGSBURG

Foundations of coloring algebra with consequences for feature-oriented programming

Peter Höfner, Bernhard Möller, Andreas Zelend

Angaben zur Veröffentlichung / Publication details:

Höfner, Peter, Bernhard Möller, and Andreas Zelend. 2012. "Foundations of coloring algebra with consequences for feature-oriented programming." *Lecture Notes in Computer Science* 7560: 33–49. https://doi.org/10.1007/978-3-642-33314-9_3.

Nutzungsbedingungen / Terms of use:

licgercopyright

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:
Deutsches Urheberrecht
Weitere Informationen finden Sie unter: / For more information see:
https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publiz/

Foundations of Coloring Algebra with Consequences for Feature-Oriented Programming

Peter Höfner[1,3], Bernhard Möller[2], Andreas Zelend[2]

[1] NICTA, Australia
[2] Universität Augsburg, Germany
[3] University of New South Wales, Australia

Abstract. In 2011, simple and concise axioms for feature compositions, interactions and products have been proposed by Batory et al. They were mainly inspired by Kästner's Colored IDE (`CIDE`) as well as by experience in feature oriented programming over the last decades. However, so far only axioms were proposed; consequences of these axioms such as variability in models have not been studied. In this paper we discuss the proposed axioms from a theoretical point of view, which yields a much better understanding of the proposed algebra and therefore of feature oriented programming. For example, we show that the axioms characterising feature composition are isomorphic to set-theoretic models.

1 Introduction

Over the last years *Feature Oriented Programming* and *Feature Oriented Software Development* (e.g. [7,8]) have been established in computer science as a general programming paradigm that provides formalisms, methods, languages, and tools for building maintainable, customisable, and extensible software. In particular, *Feature Orientation* (FO) has widespread applications from network protocols [7] and data structures [9] to software product lines [21]. It arose from the idea of level-based designs, i.e., the idea that each program (design) can be successively built up by adding more and more levels (features). Later, this idea was generalised to the abstract concept of features. A *feature* reflects an increment in functionality or in the software development.

Over the years, FO was more and more supported by software tools. Examples are `FeatureHouse` [2], the `AHEAD` Tool Suite [5], `GenVoca` [11] and Colored IDE (`CIDE`) [20]. As shown in several case studies, these tools can be used for large-scale program synthesis (e.g. [2,21,19,20]).

Although the progress over the recent past in the area of FO was quite impressive, the mathematical structure and the mathematical foundations were studied less intensively. Steps towards a structural description and analysis are done with the help of feature models. A *feature model* is a (compact and) structural representation for use in FO. With respect to (software) product lines it describes all possible products in terms of features. Feature models were first introduced in

the Feature-Oriented Domain Analysis method (FODA) [17]. Since then, feature modelling has been widely adopted by the software product line community and a number of extensions have been proposed. A further step towards an abstract description of FO was `AHEAD` [8]. It expresses hierarchical structures as nested sets of equations. Recently, several purely algebraic approaches were developed:

(a) *Feature algebra* [3] captures many of the common ideas of FO, such as introductions, refinements, or quantification, in an abstract way. It serves as a formal foundation of architectural metaprogramming [6] and automatic feature-based program synthesis [14]. The central notion is that of a *feature structure forest* that captures the hierarchical dependence in large products or product lines. Features may be added using the operation of forest superimposition, which allows a stepwise structured buildup.
(b) *Coloring algebra* [10] (CA) captures common ideas such as feature composition, interaction and products. It does not use an explicit tree or forest structure. Rather, the connection between product parts is made through *variation points* at which features or their parts may be inserted or deleted. Next to composition, the algebra also takes feature interaction into account by defining operators for determining conflicts and their repairs.
(c) *Delta modeling* [13] is not centred around a program and its structure. Rather it describes the building history of a product as a sequence of modifications, called *deltas*, that are incrementally applied to an initial product (e.g., the empty one). Conflict resolution is performed using special deltas.

The present paper builds on the second algebraic structure and combines CA with ring theory. Although most of the presented mathematics is well known, the relationship to FO is new and leads to new insights into the mathematical and structural understanding of FO. Starting with a brief recapitulation of the axioms of CA and their motivation in Section 2, we derive some basic properties for CA in Section 3, where we also discuss their relationship to FO. In Section 4, we present one of the main contributions of the paper, namely that in finite models feature composition as axiomatised in the algebra is always isomorphic to symmetric difference on sets of so called base colors. These base colors are studied more closely in Section 5; we show that many properties are already determined by them. In the following Section 6, we analyse small models of CAs, give a generic set-theoretic model and discuss a possible representation theorem for CA. Before summarising the paper in Section 8, we present another model of CA, which is useful for feature oriented software development (Section 7).

2 The Coloring Algebra

CA was introduced by Batory et al. [10], inspired by Kästner's `CIDE` [20]. In `CIDE`, a source document is painted in different colors, one color per feature. Insertion (Composition) of feature f into feature g yields a piece of code with fragments in various colors. Therefore the terms "feature" and " color" become synonymous and we will switch freely between the two in the sequel.

CA offers operations for feature composition ($+$), feature interaction ($\cdot$) as well as full interaction (cross-product) ($\times$)[1]. To illustrate the main ideas behind these operations we give an example named fire-and-flood control [10]. Assume a library building that is equipped with a fire control (*fire*). When a sensor of the control detects a fire, the sprinkling system gets activated. Later the library owner wants to retrofit a flood control system (*flood*) to protect the books and documents from water damage. When the system detects water on the floor, it shuts off the water main valve. If installed separately, both features operate as intended. However, when both are installed, they interact in a harmful way. *fire* activates the sprinklers, after a few moments the flood control shuts off the water and the building burns down. Algebraically the described system can be expressed by $\mathit{flood} + \mathit{fire}$. When using both systems, the interaction $\mathit{flood} \cdot \mathit{fire}$ of both features has to be considered: for example $\mathit{flood} \cdot \mathit{fire}$ could prioritise *fire* over *flood*. The entire system is then established by the cross-product

$$\mathit{flood} \times \mathit{fire} = \mathit{flood} \cdot \mathit{fire} + \mathit{flood} + \mathit{fire} \ .$$

Feature Composition Every program offers a set of features, which (hopefully) satisfy the specified requirements. A feature can be nearly everything: a piece of code, part of some documentation, or even an entire program itself. Such "basic" features may be composed to form a program. In CA the order of composition does not matter[2]. Moreover, CA assumes an "empty feature" 0. Let F be an abstract set of features. Then feature composition $+ : F \times F \to F$ is a binary operator which is associative and commutative.

A crucial point for algebras covering FO is how multiple instances of features are handled. There are three possible solutions:

(a) multiple occurrences are allowed, but do not add more information;
(b) duplicates are removed, so that each feature occurs at most once; or
(c) a feature is removed if it is already present, i.e., composition is *involutory*:

$$\forall f : f + f \ = \ 0 \ . \tag{1}$$

The latter is the design decision taken in CA.[3] In a monoid satisfying Equation (1) every element is its own inverse; therefore such a monoid is also known as *Boolean group* (e.g., [12]). In particular we have $f = g \Leftrightarrow f + g = 0$ and $+$ is *cancellative*, i.e., $f + g = f + h \Leftrightarrow g = h$. Every Boolean group is commutative:

$$0 = f + f = f + 0 + f = f + g + g + f \ ,$$

which implies $f + g = g + f$. Hence the axiom of commutativity can be skipped.

Feature Interaction is a commutative and associative operator $\cdot : F \times F \to F$. On the one hand it might introduce additional features to yield a "consistent"

[1] The original notation in [10] for $+$ and $\cdot$ and the element 0 below was $\cdot$, $\#$ and 1, respectively; we have changed that for a more direct connection with ring theory.
[2] This is a design decision and in contradiction to some other approaches such as [4], but follows approaches such as `CIDE`.
[3] A discussion on the usefulness of this axiom is given in [10]; the aim of the present paper is to discuss consequences of the axioms and not the axioms themselves.

and "executable" program. On the other hand, it might also list features of f and g that have to be removed. This is, for example, the case if f and g contradict each other. We follow the usual notational convention that $\cdot$ binds tighter than $+$.

CA assumes that, next to commutativity and associativity, feature interaction satisfies the following two axioms:

$$f \cdot 0 = 0 \ , \qquad (2) \qquad\qquad f \cdot f = 0 \ . \qquad (3)$$

Equation (2) expresses that no feature is in contradiction with the empty one; Equation (3) states that every feature is consistent with itself.

Moreover, CA assumes that feature interaction distributes over composition:

$$f \cdot (g + h) = f \cdot g + f \cdot h \ . \qquad (4)$$

Full Interaction is the "real" composition of two features, i.e., two features are composed under simultaneous repair of conflicts. In sum, full interaction is defined as $f \times g =_{df} f \cdot g + f + g$.

As mentioned, in CA features are abstractly viewed as colors. Therefore we now combine the above requirments into an abstract algebraic definition of CAs.

Coloring Algebra A *CA* is a structure $(F, +, \cdot, 0)$ such that $(F, +, 0)$ is a (commutative) involutive group and $(F, \cdot)$ is a commutative semigroup satisfying Equations (3) and (4). Equation (2) follows from the other axioms, in particular distributivity. Elements of such an algebra are called *colors*. Following the above motivational discussion, an element h is called a *repair* iff $\exists\, f, g : h = f \cdot g$.

Mathematically, the definition means that a CA is an involutive and commutative ring without multiplicative unit. We list a few straightforward properties of full interaction.

Lemma 2.1 *Assume a CA $(F, +, \cdot, 0)$ and $f, g, h \in F$, then the following equations hold: $(g + h) \times f = (g \times f) + (h \times f)$, $f \times 0 = f$, $f \times g = g \times f$, $(f \times g) \times h = f \times (g \times h)$ and $(g + h) \times f = (g \times f) + (h \times f)$.*

3 First Consequences

We list a couple of interesting further properties and explain their interpretation in FO. All proofs can be found automatically by Prover9 [22]. Hence, we only present those that help understanding the structure of CA.

3.1 Basic Properties of Interaction

For the following lemmas, we assume a CA $(F, +, \cdot, 0)$ and $f, g, h \in F$.

Lemma 3.1 *A repair cannot introduce new conflicts, i.e., $f \cdot g = h \Rightarrow f \cdot h = 0$.*

Lemma 3.2 *The repair of three elements f, g, h satisfies an exchange law:*

$$(f + g) \cdot (f + h) = (f + g) \cdot (g + h) = (f + h) \cdot (g + h) = f \cdot g + f \cdot h + g \cdot h \ .$$

It is easy to see that 0 is the unique fixpoint of $f \cdot x = x$. From this we get

Lemma 3.3 *A repair does not delete one of its components entirely, i.e., if* $f \neq 0$ *then* $f \cdot g \neq f$ *and if* $f + g \neq 0$ *then* $f \cdot g \neq f + g$.

Note that the precondition of the second statement is equivalent to $f \neq g$.

Lemma 3.4 *Colors cannot repair each other in "cycles", i.e.,*

(a) No non-trivial color is its own repair: $f \cdot g = f \;\Rightarrow\; f = 0$.
(b) Repairs are mutually exclusive: $f \cdot h_1 = g \wedge g \cdot h_2 = f \;\Rightarrow\; f = 0$.
(c) Part (b) can be extended to finite chains:
$f \cdot h_1 = h_2 \wedge \left(\bigwedge_{i=1}^{n} h_{3i-1} \cdot h_{3i} = h_{3i+1}\right) \wedge h_{3n+1} \cdot h_{3n+2} = f \;\Rightarrow\; f = 0$.

Proof.

(a) From $f \cdot g = f$ we infer $f \cdot g \cdot g = f \cdot g$. By absorption and strictness the left hand side reduces to 0, so that we have $0 = f \cdot g = f$. The claim also follows from Lemma 3.3.
(b) The assumptions yield $f = g \cdot h_2 = f \cdot h_1 \cdot h_2$ and Part (a) shows the claim.
(c) Straightforward induction on n. □

Moreover, inserting the consequence $f = 0$ into the antecedents of Parts (b) and (c), strictness implies that all colors occurring at the right hand side of an equation of (b) and (c), namely f, g and h_{3i+1}, are equal to 0.

The absence of cycles makes the divisibility relation w.r.t. $\cdot$ into a strict partial order on non-empty colors: we define, with $F^+ =_{df} F - \{0\}$,

$$f < g \;\Leftrightarrow_{df}\; f, g \in F^+ \wedge \exists h \in F : f \cdot h = g \ .$$

Lemma 3.5 *Composition* $+$ *and interaction* $\cdot$ *are not isotone w.r.t.* $<$.

Proof. The smallest counterexample has 8 elements.

$+$	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

$\cdot$	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	2	2	2	2
2	0	0	0	0	0	0	0	0
3	0	0	0	0	2	2	2	2
4	0	2	0	2	0	2	0	2
5	0	2	0	2	2	0	2	0
6	0	2	0	2	0	2	0	2
7	0	2	0	2	2	0	2	0

Assume a CA with $F = \{0, \ldots, 7\}$ and operations defined in the tables given on the left. Then, $1 < 2$, but $1 + 1 = 0 \not< 3 = 1 + 2$ and $1 \cdot 2 = 0 \not< 0 = 2 \cdot 2$. □

3.2 Interaction Equivalence and Ideals

It is useful to group colors according to their behaviour under interaction. To achieve this we define an equivalence relation $\sim$ by

$$f \sim g \;\Leftrightarrow_{df}\; \forall h : f \cdot h = g \cdot h \ .$$

The equivalence class of f under $\sim$ is denoted by $[f] =_{df} \{g \mid f \sim g\}$. Elements $f \in [0]$ are (as usual) called *annihilators*, since $\forall g \in F : f \cdot g = 0$.

We will show that annihilators play a central rôle for the construction of models for the $\cdot$ operator. First, we set up a connection with the strict order $<$.

Lemma 3.6 *An element of F^+ is an annihilator iff it is maximal w.r.t. $<$.*

Proof. By contraposition of Equation (2), we get $f \cdot g \neq 0 \Rightarrow f \neq 0 \wedge g \neq 0$.
($\Rightarrow$) Assume $f \neq 0$ to be maximal but not an annihilator. Then there is an element g with $f \cdot g \neq 0$. The above remark yields $g \neq 0$ and therefore $f < f \cdot g$, which is a contradiction to the maximality of f.
($\Leftarrow$) Assume $f \neq 0$ to be non-maximal but an annihilator. Then there is an element $g \neq 0$ with $f < g$, and by definition $\exists\, h : f \cdot h = g$. Since f is an annihilator, we get $g = 0$, which yields a contradiction. □

For finite $F \neq \{0\}$ there exists at least one maximal element in F^+ and hence a non-zero annihilator. It is well known that the set $[0]$ of annihilators is a ring ideal, i.e., is closed under $+$ and under $\cdot$ with arbitrary elements of F. It even forms a subtractive ideal (e.g. [1]), i.e., $f \in [0] \wedge f + g \in [0] \Rightarrow g \in [0]$.

Lemma 3.7 *If $f + g$ is an annihilator then $f \cdot g = 0$ and $f \cdot h = g \cdot h$ for $h \in F$.*

Proof. The first claim can be shown by $0 = (f + g) \cdot g = (f \cdot g) + (g \cdot g) = f \cdot g$.
The second one by $0 = (f + g) \cdot h = (f \cdot h) + (g \cdot h) \Leftrightarrow (f \cdot h) = (g \cdot h)$. □

Next we link annihilators with the equivalence relation $\sim$.

Lemma 3.8

(a) Composition is cancellative w.r.t. $\sim$, i.e., $f + g \sim f + h \Leftrightarrow g \sim h$.
In particular, $f \sim f + g \Leftrightarrow g \sim 0$.
(b) $f \sim g + h \Leftrightarrow f + g \sim h$. In particular, $f \sim g \Leftrightarrow f + g \sim 0$.
(c) $\sim$ is a congruence w.r.t. $+$ and $\cdot$.
(d) $[f] = \{f \cdot g \,|\, g \in [0]\}$.

4 Models—Feature Composition

So far we have only looked at some basic foundations and properties for FO, most of them well known in mathematics, but unknown for FO. Let us now turn to some concrete models. Looking at the literature, we note that a concrete model has only been sketched [10].

Let us first look at feature composition $(F, +, 0)$. Involution (Equation (3)) expresses that every element has an inverse, namely itself. Therefore every element has order 2. By the classification of finitely generated Abelian groups, any finite 2-group is a power of $\mathbb{Z}_2$ (the two element group); hence there is exactly one finite model satisfying these axioms for each of the cardinalities 2, 4, 8, ... This immediately follows from the Kronecker Basis Theorem (e.g. [18]).

Theorem 4.1 *Every finite algebra satisfying the axioms for feature composition is isomorphic to a model that can be obtained by using symmetric difference on a power set of a finite set.*

Due to the nature of software engineering, the set F of colors, i.e., the set of all possible combinations of features is always finite, so that the assumption of Theorem 4.1 is satisfied in that context. Moreover, by this theorem there is no need to distinguish between the abstract CA and the set model any longer. Hence we can freely use more operators and relations, such as set union $\cup$, intersection $\cap$ or subset $\subseteq$ in every finite model of CA. Both, the neutral element of CA (0) and the empty set ($\emptyset$) will be denoted by 0 in the remainder. In particular, we have $0 \subseteq f$ for all $f \in F$. The theorem states that there are generic models: assume a set B of *base colors*. Then $(2^B, \Delta, 0)$ satisfies the axioms for feature composition, where Δ is the symmetric difference of sets, defined, for $M, N \in 2^B$, as

$$M \,\Delta\, N \;=_{df}\; (M \cup N) - (M \cap N)\ .$$

The greatest element B of 2^B is denoted by $\top$.

Lemma 4.2 *In general, neither $+$ nor $\cdot$ is isotone w.r.t. $\subseteq$. Therefore, none of these operations distributes over $\cup$ or $\cap$.*

Proof. To show the first claim we give a counterexample in $(2^B, \Delta, 0)$. Let $a = \{1\}$ and $b = \{1, 2\}$. Then $a \subseteq b$ holds, but $a \,\Delta\, b = \{2\}$ is not a subset of $b \,\Delta\, b = 0$. To prove that $\cdot$ is not isotone, we use contraposition and calculate again in the set model $(2^B, \Delta, 0)$. Consider a finite model of CA. Assume that the above implication holds and that $\cdot$ is not trivial, i.e., there are base colors a, b with $a \cdot b \neq 0$. Then, since $a, b \subseteq a + b$ (as a and b are base colors), by isotony we would have $a \cdot b \subseteq (a + b) \cdot (a + b) = 0$. Since 0 is the least element w.r.t. $\subseteq$, we get $a \cdot b = 0$ i.e., a contradiction. □

We can enrich the set algebra to a first (albeit not very interesting) model for CA by assuming that there are no interactions at all between sets, i.e., for sets $M, N \in 2^B$, we define $M \cdot N =_{df} 0$. Then $(2^B, \Delta, \cdot, 0)$ forms a CA. In a later section we will discuss more sophisticated models and will show how these could be constructed systematically.

5 Base Colors

By Theorem 4.1, we can use set-theoretical knowledge for FO. In particular we can assume that every element of a finite CA is finitely generated, i.e., there is a set B of base colors (base features) from which all other colors are built. In general we call a color f of a CA F *base* iff it is isomorphic to a singleton set[4]; the set of all base colors is again denoted by B. If F is finitely generated, every element is a sum of base colors, i.e., for all $f \in F$

$$f \;=\; \sum_{i \in I} a_i$$

for an index set I and base colors $a_i \in B$.

[4] In set theory singleton sets of base colors are also called *atoms*.

In the remainder of the paper we use $a, b, c, \ldots$ to denote base colors and $f, g, h, \ldots$ for arbitrary colors. Moreover, we assume that sums (of base colors) are *reduced*, i.e., if $f = \sum_{i \in I} b_i$ then $b_i \neq b_j$ for all $i, j \in I$ with $i \neq j$.

Due to distributivity of $\cdot$ over $+$ it is possible to reduce general interaction to the one between base colors only. More precisely, assume two finitely-generated colors $f = \sum_{i \in I} a_i$ and $g = \sum_{j \in J} b_j$, then

$$f \cdot g \;=\; (\sum_{i \in I} a_i) \cdot (\sum_{j \in J} b_j) \;=\; \sum_{i \in I} \sum_{j \in J} (a_i \cdot b_j) \;. \tag{5}$$

Hence only the interaction (conflicts) of base colors has to be considered. This reduces the number of possible models.

6 Models for Coloring Algebra

6.1 Small Models

Let us now look at some possible models; we will construct them systematically. The most trivial one was already given at the end of Section 4; it had *no* interaction at all. The next example has *exactly one non-trivial* repair h. Formally, we calculate in a CA $(F, +, \cdot, 0)$ with distinguished base colors $a, b \in B$ and $h \in F$ satisfying $a \cdot b = h$ and $c \cdot d = 0$ for all other base colors $c, d \in B - \{a, b\}$.

By Lemma 3.4, a, b and h must be different. By Lemma 3.3, h differs from $a + b$, hence this example has at least three base colors. Vice versa this means that all models with at most two base colors can only have the trivial interaction.

By Equation (5), we can determine all interactions:

$$f \cdot g \;=\; \begin{cases} h & \text{if } C \\ 0 & \text{otherwise} \;, \end{cases} \qquad \text{where} \quad C \;\Leftrightarrow_{df}\; \begin{cases} (a \subseteq f \;\wedge\; b \not\subseteq f \;\wedge\; b \subseteq g) \vee \\ (a \not\subseteq f \;\wedge\; b \subseteq f \;\wedge\; a \subseteq g) \vee \\ (a \subseteq g \;\wedge\; b \not\subseteq g \;\wedge\; b \subseteq f) \vee \\ (a \not\subseteq g \;\wedge\; b \subseteq g \;\wedge\; a \subseteq f) \;. \end{cases}$$

The first case in the definition of $f \cdot g$ describes all situations where either a occurs in f and b in g, or vice versa. Therefore the repair has to be introduced. However, it forbids the case, where a and b occur in both f and g: in this setting the repair is "introduced twice" and therefore does not show up.

Let us now assume that we have two repairs for base colors in our model, i.e., there are base colors $a, b, c, d \in B$ with $a \cdot b = h_1 + h_2$, $c \cdot d = h_1 + h_3$, and $a_1 \cdot a_2 = 0$ otherwise ($a_1 \in B - \{a, b\}$, $a_2 \in B - \{c, d\}$). Note that we do not require c, d to be different from a, b. Moreover, we assume that the repairs contain a common part; the case of disjoint repairs is just a special case ($h_1 = 0$). Using again Equation (5), we can determine all interactions:

$$f \cdot g \;=\; \begin{cases} h_1 + h_2 & \text{if } \; (c \not\subseteq f \;\wedge\; d \not\subseteq f) \vee (c \not\subseteq g \;\wedge\; d \not\subseteq g) \;\wedge\; C \\ h_1 + h_3 & \text{if } \; (a \not\subseteq f \;\wedge\; b \not\subseteq f) \vee (a \not\subseteq g \;\wedge\; b \not\subseteq g) \;\wedge\; C[a/c, b/d] \\ h_2 + h_3 & \text{if } \; C \;\wedge\; C[a/c, b/d] \\ 0 & \text{otherwise} \;. \end{cases}$$

Here the formula $C[a/c, b/d]$ is C with a replaced by c and b by d. These two examples show how the interaction operation can be derived from interaction on base colors. Of course one has to keep in mind when defining the interaction on base colors that some equations such as $f \cdot g = f$ are not possible (see Section 3). However, the examples give rise to the conjecture that there cannot be many different models for CA, since everything can be reduced to base colors and the variety there is limited. To underpin this conjecture, we generated all models of a particular size using Mace4, a counterexample generator [22]. The results are presented in Table 1. Of course generation of algebras with Mace4 requires isomorphism checking; although this is offered by the tool suite, it is resource intensive. Hence we could also determine numbers up to algebras of size 16. The table shows (a) the number of possible algebras (up to isomorphism) when only the axioms for interaction $(\cdot)$ are used, and (b) the number of CAs.

#base colors/ #colors	#interact. (up to iso.)	# CA (up to iso.)
1/2	1	1
2/4	2	1
3/8	557	2
4/16		2

Table 1. Number of Models for CA

6.2 A General Model for Coloring Algebra

In this chapter we show how to construct a general model for CA. The presented model is based on sets only and can already be applied straight away to FO when assuming that a color (feature) is a set of base colors. Later we will present a model that is even more practicable for FO, based on variation points.

As before we assume a set B of base colors and set F as 2^B. As we have seen, the only possibility for composition is $f + g = f \,\Delta\, g$. (cf. Theorem 4.1).

Furthermore, Equation 5 shows that interaction needs only be considered at the level of base colors. Next to that, we assume that the interaction of base colors is again a base color. In sum, we assume an associative interaction operator $\circ$ on B, i.e., $(B, \circ)$ is a semigroup, and a special element $e \in B$ that satisfies the annihilation properties $e \circ a = e = a \circ a$ for all $a \in B$. A structure $(B, \circ, e)$ with these properties is called a *base color semigroup*.

Based on that, feature interaction $(\cdot)$ can be defined as

$$f \cdot g =_{df} \bigtriangleup_{a \in f} \bigtriangleup_{b \in g} \iota(a \circ b) \ ,$$

where the injection $\iota : B \to F$ is given by $\iota(e) = 0$ and $\iota(a) = \{a\}$ for $a \in B - \{e\}$. By associativity and commutativity of Δ this is well defined. Within $\cdot$, multiple occurrences of the same $\circ$-product cancel out so that at most one of them is left.

By this remark we immediately obtain $f \cdot f = 0$, since every product $a \circ a = e$ and hence $\iota(a \circ a) = 0$, while for any two different colors $a, b \in f$ we have both products $\iota(a \circ b)$ and $\iota(b \circ a)$ in the Δ-aggregation, which by commutativity of $\circ$ are equal and hence cancel out. Another straightforward consequence of the definition is $f \cdot 0 = 0$.

Lengthy but straightforward calculations show that interaction, as defined earlier in this section, is associative commutative and distributes over $+$ (Δ) [15].

Thus we have defined a concrete model of a CA, solely based on a set of base colors and an abstract interaction operation $\circ$ on them. In the remainder of this section, we will give a possible concrete definition of this abstract operation.

For that, we assume a finite set P of *pigments*. The idea is to define base colors as certain sets of pigments. As before, sets of base colors will be used to define colors. Formally, a *set of base colors* is a non-empty subset $B \subseteq 2^P$ that is downward closed: $a \in B \wedge b \subseteq a \Rightarrow b \in B$. Hence every set of base colors contains $\emptyset$. The set B is called *full* iff $B = 2^P$.

For two base colors (sets of pigments) $a, b \in B$ we define a non-conflict predicate *noconf* by

$$noconf(a,b) \Leftrightarrow_{df} a \neq \emptyset \wedge b \neq \emptyset \wedge a \cap b = \emptyset \ .$$

Intuitively, two base colors do not show any conflict, if they do not share a common resource (pigment). The condition $a \neq \emptyset \wedge b \neq \emptyset$ is needed to exclude the empty base color, which by definition has no conflict with any other base color, but should also not interact with any. As we will see in the next definition, we have to distinguish these two behaviours.

The interaction $\circ : B \times B \to B$ of base colors is defined by

$$a \circ b =_{df} \begin{cases} a \cup b & \text{if } noconf(a,b) \wedge a \cup b \in B \ , \\ \emptyset & \text{otherwise.} \end{cases}$$

This definition entails $a \circ a = \emptyset = \emptyset \circ a$ for all $a \in B$. Hence $\emptyset$ (as element of B) plays the rôle of the annihilating element e above. Moreover, $\circ$ is commutative and associative. A proof can be found in [15]. Hence $(B, \circ, \emptyset)$ is a base color semigroup that can be used to create a CA, according to the definitions for composition and interaction given earlier in this section.

The definitions given above immediately entail the following property.

Lemma 6.1 *If $a \in B$ is $\subseteq$-maximal in B then a is an annihilator.*

Let us illustrate the construction with an example.

Example 6.2 Assume three pigments r, g, b and the full set of base colors, i.e., $B = 2^{\{r,g,b\}}$. Using the interaction operation $\circ$ on base colors, we get for example

$$\{r\} \circ \{g,b\} = \{r,g,b\} \qquad \text{and} \qquad \{r\} \circ \{r,g\} = \emptyset = \{r,g\} \circ \{g,b\} \ .$$

We also illustrate the CA over $(B, \circ, \emptyset)$. For convenience and readability we leave out set braces for colors (elements of $F = 2^B$), that consist of only one base color. For example rgb is used instead of $\{\{r,b,g\}\}$. We can now define a color that consists of all base colors containing the pigment r as

$$red =_{df} \{\{r\},\{r,g\},\{r,b\},\{r,g,b\}\} = r + rg + rb + rgb \ .$$

As an example how interaction $\cdot$ on colors works, consider

$$
\begin{aligned}
& (r + rg + rb + rgb) \cdot (b + rb + g) \\
= \;& rb + \emptyset + rg + rgb + \emptyset + \emptyset + \emptyset + \emptyset + rbg + \emptyset + \emptyset + \emptyset \\
= \;& rb + rg \; .
\end{aligned}
$$

To conclude the example we briefly resume the discussion on the relationship to `CIDE` [20]. As mentioned, in that tool code can be colored. Singly colored code comprises the features that a customer can choose. But code can also be endowed with more than one color. If a code fragment is, for example, marked with the colors red and blue, that fragment is the repair of the two singly colored fragments with colors red and blue. Similarly, code fragments marked with three colors are repairs of three features, etc. By the full interaction operator, a repair is then composed with the two conflicting features into a new one, where by cancellativity of composition the conflicting parts are removed and supplemented by new ones if necessary.

Let us assume that a customer chooses the pigments r and g from the above set P. The one can automatically compute their repair: $r \cdot g = rg$. To create and deliver the final product full interaction ($\times$) can be used:
$r \times g = rg + r + g$. It can be seen that the repair rg has indeed been added. If now a customer wants have the additional feature b, the model of CA can determine all the bits needed for the final product;

$$
\begin{aligned}
r \times g \times b &= (rg + r + g) \times b \\
&= (rg + r + g) \cdot b + rg + r + g + b \\
&= rgb + rb + gb + rg + r + g + b \; .
\end{aligned}
$$

We see that now not only the singly colored fragments are used, but also all repairs. □

6.3 Towards a General Representation Theorem

We have presented a representation theorem for composition, and defined and discussed a number of possible interaction operations. So far, we have not found a complete representation theorem for CA; in this section we present a couple of useful properties that are hopefully steps towards one.

We investigate *generating systems* w.r.t. interaction $\cdot$, i.e., subsets $G \subseteq F$ such that every element in the image set of $\cdot$ equals a combination $g_1 \cdot \ldots \cdot g_n$ for some $n \in \mathbb{N}$ and $g_i \in G$. A generating system is *minimal* if no proper subset of it is a generating system. In a finite algebra such systems always exist.

Lemma 6.3 *Let G be a minimal generating system for $\cdot$. Then no two distinct elements of G can be related by $\sim$.*

Proof. A relation $g_1 \sim g_2$ would mean that in every $\cdot$-product of the above form g_1 could be replaced by g_2 without changing its value. Hence one of g_1, g_2 could be omitted from G while still yielding a generating system, in contradiction to the choice of G. □

Theorem 6.4 *Let G be a minimal generating system for $\cdot$. Then the elements of G form a system of representatives for the equivalence classes of $\sim$. Since $\sim$ is a congruence, the set of these classes can be made into a quotient semiring by defining $[f] + [g] =_{df} [f + g]$ and $[f] \cdot [g] =_{df} [f \cdot g]$.*

Proof. Every element lies in its own equivalence class, while, by the previous lemma, different generators lie in different classes. This shows the first claim. The second one is standard semiring theory. □

When analysing the constructed and the generated models, we also looked more closely at the structure of the equivalence classes generated by the relation $\sim$. We made the following observations that are underlying the conjectures below on the representation theorem:

(a) The smallest element (w.r.t. $\subseteq$) of each class is a base color.
(b) All other elements are formed by composition with every possible combination of the annihilating base colors.

This and some observations presented earlier motivates us to state the following

Conjectures The first conjecture is that the assumption we made at the beginning of Section 6.2 is always true, namely that the repair of two base colors is *always* a base color itself. If this conjecture is correct, the presented model might be *the* generic model, i.e., all interaction operations of all models of CA are always isomorphic to such a model. In particular, the only freedom to define interaction is given by the underlying semigroup of pigments. That would also imply that in FO only repairs of really simple fragments have to be considered and the number of possible CAs could be determined by the number of semigroups satisfying the additional annihilation requirements. This latter claim is underpinned by our third conjecture stating that in case of a non-trivial interaction operation, the elements $f \in 2^N$ form a system of representatives for the equivalence classes, where $[f] = f + [0] =_{df} \{f + g \,|\, g \in [0]\}$ and N is the set of all non-annihilating base colors.

The last conjecture is partially substantiated by the following property, which is immediate from Lemma 3.8(b):

Lemma 6.5 *The equivalence classes $[f]$ under $\sim$ are closed under composition with annihilators:*

$$[f] + [0] \subseteq [f]$$

7 A Model Based on Variation Points

Although the set-theoretic model given in Sections 4 and 6 is interesting and already covers a lot of the aspects of `CIDE`, it is, of course, not fully adequate for FO, since it does not take details of the program structure, such as classes and objects, into account. To get a handle on such aspects, in FO *variation points* are used. "A *variation point* identifies a location at which a variable part may occur.

It locates the insertion point for the variants and determines the characteristics (attributes) of the variability" [23]. Variation points are also called *extension points* (e.g., [20]) or *hot spots* (e.g., [16]).

The model we present here can be used directly for FO, as it is based on variation points and code fragments. We assume disjoint sets VP of variation points and C of code fragments. Variation points might, e.g., be given as line numbers before or after which further elements can be inserted.

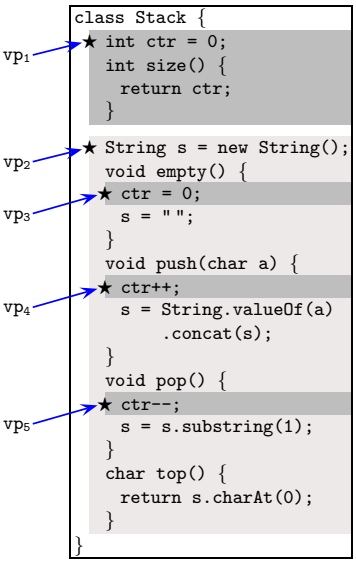

Fig. 1. The Counted Stack With Variation Points [10]

An illustrative example with a "Counted Stack" is presented in Figure 1; it was taken from [10]. Due to lack of space we skip an explanation of the details; the figure should just give an impression how things look like.

Our model will be based on the set model presented before. Therefore we only consider code fragments that commute with each other. For example, we could look at entire methods, i.e., code fragments that start with something like "`void empty(){`" and end with "}" and may contain variation points from VP as well as code fragments from C. Note that a code fragment can contain variation points again.

A *program* is now a (total) function $p : VP \to 2^{VP \cup C}$.[5] Its semantics is as follows: if, for a variation point `vp` the value $p(\texttt{vp})$ is not the empty set then $p(\texttt{vp})$ is installed at `vp`; otherwise `vp` remains empty. This construction is well defined, since by the standard isomorphism $(A^B)^C \simeq A^{C \times B}$ for function spaces we have $(2^{VP \cup C})^{VP} \simeq 2^{VP \times (VP \cup C)}$, where a program p is represented by the relation $\{(\texttt{vp}, \texttt{vqc}) \mid \texttt{vqc} \in p(\texttt{vp})\}$.

The empty program e is the empty function, i.e., $e(\texttt{vp}) = \emptyset$ for all $\texttt{vp} \in VP$. A program *white* with a method body at variation point `start` is given as

$$\mathit{white}(\texttt{vp}) = \begin{cases} \{\, \texttt{class Stack\{ vp}_1\ \texttt{vp}_2\ \texttt{\} } \,\} & \text{if } \texttt{vp} = \texttt{start} \\ \emptyset & \text{otherwise .} \end{cases}$$

This coincides with the "white" part of Fig. 1, while the "darkgray" fragment is

$$\mathit{darkgray}(\texttt{vp}) = \begin{cases} \{\, \texttt{int ctr = 0; ... ctr;} \,\} & \text{if } \texttt{vp} = \texttt{vp}_1 \\ \{\, \texttt{ctr = 0;} \,\} & \text{if } \texttt{vp} = \texttt{vp}_3 \\ \{\, \texttt{ctr++;} \,\} & \text{if } \texttt{vp} = \texttt{vp}_4 \\ \{\, \texttt{ctr--;} \,\} & \text{if } \texttt{vp} = \texttt{vp}_5 \\ \emptyset & \text{otherwise .} \end{cases}$$

To build a program from a given function p, we just replace all occurrences of each `vp` by its value $p(\texttt{vp})$. This yields a function with no variation points in

[5] An isomorphic model uses partial functions which are undefined for empty variation points.

its values. Of course, this is only possible if the values of p do not depend on each other cyclically. If we now choose one variation point as the `start` point, a program (with filled variation points) has been derived.

By this simple algorithm for program derivation, the presented model is particularly interesting for FO. The remaining question is how to define feature composition and feature interaction.

Similar to the set-theoretic model, feature composition can be defined via the symmetric difference:

$$(p + q)(\mathtt{vp}) =_{df} p(\mathtt{vp}) \,\Delta\, q(\mathtt{vp}) \ .$$

This definition satisfies the laws for feature composition and behaves naturally. Identical parts of the values $p(\mathtt{vp})$ and $q(\mathtt{vp})$ are deleted, differing parts are collected in the result set.

Let us explain this with a simple example. Assume two programs p and q:

$$p(\mathtt{vp}) =_{df} \begin{cases} \{v_2\} & \text{if } \mathtt{vp} = \mathtt{vp}_1 \\ \{v_3\} & \text{if } \mathtt{vp} = \mathtt{vp}_2 \\ \{v_2, v_4\} & \text{if } \mathtt{vp} = \mathtt{vp}_3 \\ \emptyset & \text{otherwise ,} \end{cases} \qquad q(\mathtt{vp}) =_{df} \begin{cases} \{v_3\} & \text{if } \mathtt{vp} = \mathtt{vp}_1 \\ \{v_3\} & \text{if } \mathtt{vp} = \mathtt{vp}_2 \\ \{v_1, v_4\} & \text{if } \mathtt{vp} = \mathtt{vp}_3 \\ \emptyset & \text{otherwise .} \end{cases}$$

Both programs assign non-trivial information only to the variation points $\mathtt{vp}_1$, $\mathtt{vp}_2$ and $\mathtt{vp}_3$. The values at $\mathtt{vp}_1$ are disjoint; the composition unites the values. The values at $\mathtt{vp}_2$ are identical; the composition removes them and leaves $\mathtt{vp}_2$ "unset". The values at $\mathtt{vp}_3$ are neither disjoint nor identical—they have a non-empty intersection; the composition deletes all values occurring in both parts and retains the rest. Formally this means

$$(p + q)(\mathtt{vp}) =_{df} \begin{cases} \{v_2, v_3\} & \text{if } \mathtt{vp} = \mathtt{vp}_1 \\ \emptyset & \text{if } \mathtt{vp} = \mathtt{vp}_2 \\ \{v_1, v_2\} & \text{if } \mathtt{vp} = \mathtt{vp}_3 \\ \emptyset & \text{otherwise .} \end{cases}$$

The same construction can be applied to implement feature interaction:

$$(p \cdot q)(\mathtt{vp}) =_{df} p(\mathtt{vp}) \cdot q(\mathtt{vp}) \ .$$

In the concrete model this turns into $(p \cdot q)(\mathtt{vp}) =_{df} \Delta_{a \in p(\mathtt{vp})} \Delta_{b \in q(\mathtt{vp})} \iota(a \circ b)$. To complete this definition we need to specify the underlying base color semigroup. We choose the pigment set $P =_{df} VP \cup C$ and the full base color set $B =_{df} 2^P$.

In this paper we have shown that, for a given size, each definition of $+$ is isomorphic to that of symmetric difference in the set-model; by this we do not have much freedom to define composition and the presented definition seems to be canonical. In contrast to that, the interaction operation $\cdot$ offers much more flexibility. Of course, we cannot give a compact definition, since interaction really depends on implementation details and therefore on the source code. However, as we have shown in Section 5, only the interaction between base colors has to be defined, interaction for arbitrary elements then lifts by Equation (5). We can even do better and give the programmer some guidelines on how repairs should be defined by the lemmas given in Section 3. The most important of these is

Lemma 3.4 which can easily be implemented and offers a quick consistency check on interactions.

8 Conclusion and Outlook

We have carried out a careful analysis of coloring algebra (CA) [10]. The study has yielded several interesting and sometimes surprising results.

First, we have presented a series of properties for FO. Most of them could be proven fully automatically using an automated theorem prover such as Prover9.

Second, we have used Mace4 not only to falsify conjectures (as we do regularly), but also for the generation of finite models. Doing this, and by creating and analysing models by hand, it turned out that there exist only very few models of CA, up to isomorphism. This was a surprise: when the algebra was designed, it was believed that the operation for feature interaction $(\cdot)$ offers a lot of freedom, and probably the composition operation does so as well. However, we have shown that composition is always isomorphic to symmetric difference in a set model. By this representation theorem more operations, such as set union and intersection, could be introduced in CA for free. This has allowed us the definition of base colors, which come into play naturally as the "smallest" features available. We also gave a derivation towards a representation theorem for CA in general. So far we could not entirely prove our conjecture. This is not surprising, since representation theorems are generally hard to prove. However, the lemmas presented indicate that our conjecture holds.

As the last contribution of the paper, we have given a concrete model for FO. It is based on functions over sets and is more useful for FO than the generic set-theoretical one. To show this, we have given a simple algorithm to describe how elements of this model can be transformed into executable programs.

In sum, the analysis, even without the missing representation theorem, has yielded deeper insights for CA and will hopefully lead to a much better understanding of the basics underlying feature oriented programming and feature oriented software development.

Acknowledgement We are grateful to Peter Jipsen for pointing out Kronecker's base theorem. Part of the work was carried out during a sponsored visit of the second author at NICTA. The work of the third author was funded by the German Research Foundation (DFG), project number MO 690/7-2 `FeatureFoundation`.

References

1. Allen, P.J.: A fundamental theorem of homomorphisms for semirings. Proceedings of the American Mathematical Society 21(2), pp. 412–416 (1969)
2. Apel, S., Kästner, C., Lengauer, C.: FeatureHouse: Language-independent, automated software composition. In: 31th International Conerence on Software Engineering(ICSE). pp. 221–231. IEEE Press (2009)
3. Apel, S., Lengauer, C., Möller, B., Kästner, C.: An algebra for features and feature composition. In: AMAST 2008: Algebraic Methodology and Software Technology. LNCS, vol. 5140, pp. 36–50. Springer (2008)

4. Apel, S., Lengauer, C., Möller, B., Kästner, C.: An algebraic foundation for automatic feature-based program synthesis. Sc. Comp. Prog. 75(11), 1022–1047 (2010)
5. Batory, D.: Feature-oriented programming and the AHEAD tool suite. In: ICSE '04: 26th International Conference on Software Engineering. pp. 702–703. IEEE Press (2004)
6. Batory, D.: From implementation to theory in product synthesis. ACM SIGPLAN Notices 42(1), 135–136 (2007)
7. Batory, D., O'Malley, S.: The design and implementation of hierarchical software systems with reusable components. ACM Transactions Software Engineering and Methodology 1(4), 355–398 (1992)
8. Batory, D., Sarvela, J.N., Rauschmayer, A.: Scaling step-wise refinement. In: ICSE '03: 25th International Conference on Software Engineering. pp. 187–197. Proceedings of the IEEE (2003)
9. Batory, D., Singhal, V., Sirkin, M., Thomas, J.: Scalable software libraries. ACM SIGSOFT Software Engineering Notes 18(5), 191–199 (1993)
10. Batory, D., Höfner, P., Kim, J.: Feature interactions, products, and composition. In: 10th ACM international conference on Generative Programming and Component Engineering (GPCE'11). pp. 13–22. ACM Press (2011)
11. Batory, D., Singhal, V., Thomas, J., Dasari, S., Geraci, B., Sirkin, M.: The GenVoca model of software-system generators. IEEE Software 11(5), 89–94 (1994)
12. Bernstein, B.: Sets of postulates for boolean groups. Annals of Mathematics 40(2), 420–422 (1939)
13. Clarke, D., Helvensteijn, M., Schaefer, I.: Abstract delta modeling. In: Visser, E., Järvi, J. (eds.) GPCE. pp. 13–22. ACM (2010)
14. Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley (2000)
15. Höfner, P., Möller, B., Zelend, A.: Foundations of coloring algebra with consequences for feature-oriented programming. Tech. Rep. 2012-06, Institut für Informatik der Universität Augsburg (2012)
16. Johnson, R., Foote, B.: Designing reusable classes. Journal of Object Oriented Programming 1(2), 22–35 (1988)
17. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Carnegie-Mellon University Software Engineering Institute (1990)
18. Kargapolov, M., Merzliakov, I.: Fundamentals of the Theory of Groups. Graduate texts in mathematics, Springer (1979)
19. Kästner, C., Apel, S., Batory, D.: A case study implementing features using AspectJ. In: Software Product Lines, 11th International Conference (SPLC). pp. 223–232. IEEE Computer Society (2007)
20. Kästner, C.: Virtual Separation of Concerns: Toward Preprocessors 2.0. Ph.D. thesis, University of Magdeburg (2010)
21. Lopez-Herrejon, R., Batory, D.: A standard problem for evaluating product-line methodologies. In: Bosch, J. (ed.) GCSE '01: Generative and Component-Based Software Engineering. LNCS, vol. 2186, pp. 10–24. Springer (2001)
22. McCune, W.W.: Prover9 and Mace4. <`http://www.cs.unm.edu/~mccune/prover9`>, (accessed July 12, 2012)
23. Reinhartz-Berger, I., Tsoury, A.: Experimenting with the comprehension of feature-oriented and UML-based core assets. In: Halpin, T., Nurcan, S., Krogstie, J., Soffer, P., Proper, E., Schmidt, R., Bider, I. (eds.) Enterprise, Business-Process and Information Systems Modeling. Lecture Notes in Business Information Processing, vol. 81, pp. 468–482. Springer (2011)