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 [20]. 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`) [19]. As shown in several case studies, these tools can be used for large-scale program synthesis (e.g. [2,20,18,19]).

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) [16]. 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 mathematical 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 interaction, products and feature composition. 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 product 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 in the form of 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. 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 basic features. These basic features 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 and discuss a possible representation theorem for CA. Before summarising the paper in Section 8, we present a another model of CA, which is useful for feature oriented software development (Section 7).

2 The Coloring Algebra

Inspired by Kästner's `CIDE` [19], CA was introduced by Batory et al. [10]. It offers operations for feature composition ($\cdot$), feature interaction (#) as well as full interaction (cross-product) ($\times$).

To illustrate the main idea 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 $\mathit{flood} \cdot \mathit{fire}$. When using both systems, the interaction $\mathit{flood} \# \mathit{fire}$ of both features has to be considered. The entire system is then established by the cross-product

$$\mathit{flood} \times \mathit{fire} = (\mathit{flood} \# \mathit{fire}) \cdot \mathit{flood} \cdot \mathit{fire} \ .$$

One possibility to resolve the interaction $\mathit{flood} \# \mathit{fire}$ is to prioritise *fire* over *flood*.

Feature Composition Every program offers a set of features, which (hopefully) satisfy a stakeholder's requirements. By this informal description 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[4]. Moreover, CA assumes an "empty program" 1, which does not contain any features. Let F be an abstract set of features. Then feature composition is a binary operator $\cdot : F \times F \to F$, which is associative and commutative.

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

(a) multiple occurrences are possible, but do not add more information;
(b) removing duplicates and guaranteeing that each feature only occurs once; or
(c) remove a feature if it is already present, i.e., is *involutory*:

$$\forall f : f \cdot f \ = \ 1 \ . \tag{1}$$

The latter is the design decision taken in CA.[5]

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 $f = g \ \Leftrightarrow \ f \cdot g = 1$ and $\cdot$ is *cancellative*, i.e., $f \cdot g = f \cdot h \ \Leftrightarrow \ g = h$. Every Boolean group is commutative:

$$f \cdot g = 1 \cdot f \cdot g \cdot 1 = (g \cdot g) \cdot f \cdot g \cdot (f \cdot f) = g \cdot (g \cdot f) \cdot (g \cdot f) \cdot f = g \cdot f \ .$$

Hence the axiom of commutativity can be skipped.

[4] This is a design decision and in contradiction to some other approaches such as [4], but follows approaches such as `CIDE`.

[5] A discussion on the usefulness of this axiom is given in [10]; the aim of the present paper is to discuss consequences of CA and not the axioms.

Feature Interaction is a commutative and associative operator $\# : F \times F \to F$. On the one hand it might add additional features to end up with a "consistent" 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.
CA assumes that, next to commutativity and associativity, feature interaction satisfies the following two axioms:

$$f \# 1 = 1 \,, \qquad (2) \qquad\qquad f \# f = 1 \,. \qquad (3)$$

Intuitively, $f \# g$ describes conflicts between f and g and offers a "repair". Equation (2), also called *strictness*, expresses that no feature is in contradiction with the empty one; Equation (3), also called *absorption*, states that every feature is consistent with itself.

Moreover, CA assumes that feature interaction distributes over composition:

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

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 $f \times g =_{df} (f \# g) \cdot f \cdot g$. For the remainder of the paper we assume that $\#$ binds stronger than $\cdot$.

Coloring Algebra Putting all these requirements together, a *CA* is a structure $(F, \cdot, \#, 1)$ such that $(F, \cdot, 1)$ is a (commutative) involutive group and $(F, \#)$ is a commutative semigroup satisfying Equations (3) and (4). Note that Equation (2) follows from the other axioms, in particular distributivity. Elements of this algebra are called *colors*. An element h is called a *repair* iff $\exists\, f, g : \; h = f \# g$.

Lemma 2.1. *Assume a CA* $(F, \cdot, \#, 1)$ *and* $f, g, h \in F$*, then the following equations hold:* $(g \cdot h) \# f = (g \# f) \cdot (h \# f)$ *,* $f \times 1 = f$*,* $f \times g = g \times f$*, and* $(f \times g) \times h = f \times (g \times h)$*.*

The proofs are straightforward, can be automated by any first-order automated theorem prover or can be found in [10]. We will freely use them in the remainder of the paper. More consequences are discussed in the next section.

3 First Consequences

We now list a couple of interesting properties of the axioms and explain their interpretations in FO. All proofs were found by Prover9 [21]; a proof template is given in Appendix B. Due to this, we skip most of the proofs and only list those which help to understand the structure of CA.

3.1 Basic Properties of Interaction

For the following lemmas, we assume a CA $(F, \cdot, \#, 1)$ and $f, g, h \in F$.

Lemma 3.1 *A repair cannot introduce new conflicts, i.e.,* $f\#g = h \Rightarrow f\#h = 1$.

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

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

It is easy to see that 1 is the unique fixpoint of $f \# x = x$. From this we get

Lemma 3.3 *A repair does not delete one of its components entirely, i.e., if* $f \neq 1$ *then* $f \# g \neq f$ *and if* $f \cdot g \neq 1$ *then* $f \# g \neq f \cdot g$.

Note that the precondition of the second statement is equivalent to $f \neq g$. Moreover, if the CA has a greatest element $\top$ then $f \# \top \neq \top$.

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

(a) No non-trivial feature is its own repair: $f \# g = f \ \Rightarrow \ f = 1$.
(b) Repairs are mutually exclusive: $f \# h_1 = g \ \wedge \ g \# h_2 = f \ \Rightarrow \ f = 1$.
(c) Part (b) can be extended to finite chains:

$$f \# h_1 = h_2 \ \wedge \ \Big(\bigwedge_{i=1}^{n} h_{3i-1} \# h_{3i} = h_{3i+1} \Big) \ \wedge \ h_{3n+1} \# h_{3n+2} = f \ \Rightarrow \ f = 1 \ .$$

Proof.

(a) From $f \# g = f$ we infer $f \# g \# g = f \# g$. By absorption and strictness the left hand side reduces to 1, so that we have $1 = f \# g = f$. The claim also follows from Lemma 3.3.
(b) Plugging in the assumptions we obtain $f = g \# h_2 = f \# h_1 \# h_2$ and Part (a) shows the claim.
(c) Straightforward induction on n. □

Moreover, pluggin the consequence $f = 1$ into the antecedents of Parts (b) and (c) by strictness implies that all colors involved are equal to 1.

The absence of cycles gives raise to define a partial order on colors. Assume a CA $(F, \cdot, \#, 1)$ and $f, g \in F$. We define a strict partial order $<$ over $F - \{1\}$ by

$$f < g \ \Leftrightarrow_{df} \ \exists h \in F : f \# h = g \ ,$$

i.e., $<$ is irreflexive and transitive. The proof follows immediately from Lemma 3.4, is straight forward and can again be automated.

3.2 Interaction Equivalence

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

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

The equivalence class of g under $\sim$ is denoted by $[g] =_{df} \{f \mid f \sim g\}$.

Lemma 3.5

(a) Composition is cancellative w.r.t. $\sim$, i.e., $f \cdot g \sim f \cdot h \Leftrightarrow g \sim h$. In particular, $f \sim f \cdot g \Leftrightarrow g \sim 1$.
(b) $f \sim g \cdot h \Leftrightarrow f \cdot g \sim h$. In particular, $f \sim g \Leftrightarrow f \cdot g \sim 1$.
(c) $\sim$ is a congruence w.r.t. $\cdot$ and $\#$.

4 Models—Feature Composition

So far we have only looked at some basic foundations and properties 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 $(F, \cdot, 1)$. 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. [17]).

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.*

Proof. Although Kronecker's result already provides a proof, we give an independent one that can later partially be re-used for the analysis of the $\#$ operation.

Assume an algebra $(F, \cdot, 1)$ with the properties required by the theorem. We call $G \subseteq F$ a *generating system* if every element of F can be obtained as a product $\prod_{i \in I} g_i$ of a finite family $(g_i)_{i \in I}$ of elements $g_i \in G$. For instance, setting $G = F$ provides a generating system.

A product $\prod_{i \in I} g_i$ is *reduced* if there are no indices $i \neq j$ with $g_i = g_j$, i.e., if every element of G occurs at most once in it. Since $\cdot$ is assumed to be associative, commutative and involutory, every product has the same value as a reduced one and hence can be represented by a set rather than a family of G-elements.

Consider now a generating system $M \subseteq F$ of minimal cardinality (which exists by finiteness of F). Then $1 \notin M$ (if $F = \{1\}$ then $M = \emptyset$, since 1 can always be represented as the empty product). We now show that the set representation of F-elements by reduced M-products is unique.

To this end we denote the product of a finite subset $H \subseteq F$ by $\prod H$, which by associativity and commutativity of $\cdot$ is uniquely defined (note that $\prod \emptyset = 1$). Suppose now $\prod H = \prod K$ for two subsets $H, K \subseteq M$, but $H \neq K$. Let $L =_{df} H \cap K$ and $l =_{df} \prod L$. We multiply both sides of the above equation by l and obtain $\prod(H - L) = \prod(K - L)$. Now one of $H - L$ and $K - L$ must be non-empty, otherwise we would have $H = L = K$ and hence $H = K$. Assume w.l.o.g. that $H - L \neq \emptyset$, choose some $m \in H - L$ and $m \notin K - L$ and set $n =_{df} \prod(H - (L \cup \{m\}))$. By multiplying both sides of the equation $\prod(H - L) = \prod(K - L)$ by n and using involution we obtain $m = \prod(K - L) \cdot \prod(H - (L \cup \{m\})) = \prod((K - L) \cup (H -$

$(L \cup \{m\}))$. But then m could be eliminated from the generating system H to obtain another generating system of smaller cardinality, contradicting the choice of H.

In sum, $\prod : 2^M \to F$ is a bijection, so that F is isomorphic to a power set. Finally, again by associativity, commutativity and involutivity of $\cdot$, we have for $H, K \subseteq M$ that $(\prod H) \cdot (\prod K) = \prod((H \cup K) - (H \cap K)) = \prod(H \,\Delta\, K)$, which shows that $\cdot$ corresponds to symmetric difference. □

Due to the nature of software engineering, the set F of all possible combinations of features is always finite, so that the assumption of Theorem 4.1 is satisfied in that context.

The theorem states that there are generic models: assume a set A of *base colors*. Then $(2^A, \Delta, \emptyset)$ satisfies the axioms for feature composition, where Δ is defined, for $M, N \in 2^A$ as

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

By this we can use $\subseteq$, $\cap$ and $\cup$ in every finite model of CA. We will freely do this in the remainder of the paper. The greatest element A of 2^A is denoted by $\top$.

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

Proof. The claim concerning $\cdot$ can be shown by Mace4 within no time.
$f \subseteq g \nRightarrow f \# h \subseteq g \# h$ could not be found by automation; we show a proof by contraposition. Consider a finite model of CA. Assume that the above implication holds and that $\#$ is not trivial, i.e., there are base colors a, b with $a \# b \neq 1$. Then, since $a, b \subseteq a \cdot b$, by isotony we would have

$$a \# b \subseteq (a \cdot b) \# (a \cdot b) = 1 \ ,$$

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 $M, N \in 2^A$, we define $M \# N =_{df} \emptyset$. Then $(2^A, \Delta, \#, \emptyset)$ 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 A of base features from which all other features are built. In general we call an element f of a CA F *base* iff it is isomorphic to a singleton set[6]; the set

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

of all base colors is again denoted by A. In case F is finite, every element is a product of base colors, i.e., for all $f \in F$

$$f = \prod_{i \in I} a_i$$

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

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

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

$$f \# g = (\prod_{i \in I} a_i) \# (\prod_{j \in J} b_j) = \prod_{i \in I} \prod_{j \in J} (a_i \# b_j). \tag{5}$$

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 $(2^A, \cdot, \#, 1)$ with distinguished base colors $a, b \in A$ and $h \in 2^A$ satisfying $a \# b = h$ and $c \# d = 1$ for all other base colors $c, d \in A - \{a, b\}$.

By Lemma 3.4, a, b and h must be different. By Lemma 3.3, h is different from $a \cdot b$, 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:

$$f \# g = \begin{cases} h & \text{if } C \\ 1 & \text{otherwise} , \end{cases} \quad \text{where} \quad C \Leftrightarrow_{df} \begin{cases} (a \subseteq f \wedge b \not\subseteq f \wedge b \subseteq g) \vee \\ (a \not\subseteq f \wedge b \subseteq f \wedge a \subseteq g) \vee \\ (a \subseteq g \wedge b \not\subseteq g \wedge b \subseteq f) \vee \\ (a \not\subseteq g \wedge b \subseteq g \wedge a \subseteq f) \ . \end{cases}$$

The first case 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 atoms in our model, i.e., there are atoms $a, b, c, d \in A$ with $a \# b = h_1 \cdot h_2$, $c \# d = h_1 \cdot h_3$, and $a_1 \# a_2 = 1$ otherwise ($a_1 \in A - \{c, d\}$, $a_2 \in A - \{a, b\}$). Note that we do not require c, d to be different

#base colors	#elements	# composition (up to iso.)	# interaction (up to iso.)	# CA (up to iso.)
1	2	1	1	1
2	4	1	2	1
3	8	1	557	2
4	16	1		2
5	32	1		

Table 1: Number of Models for CA

from a, b. Moreover, we assume that the repairs contain a common part; the case of disjoint repairs is just a special case ($h_1 = 1$). Using again Equation (5), we can determine all interactions:

$$f \# g = \begin{cases} h_1 \cdot h_2 & \text{if } (c \not\sqsubseteq f \wedge d \not\sqsubseteq f) \vee (c \not\sqsubseteq g \wedge d \not\sqsubseteq g) \wedge C \\ h_1 \cdot h_3 & \text{if } (a \not\sqsubseteq f \wedge b \not\sqsubseteq f) \vee (a \not\sqsubseteq g \wedge b \not\sqsubseteq g) \wedge C[a/c, b/d] \\ h_2 \cdot h_3 & \text{if } C \wedge C[a/c, b/d] \\ 1 & \text{otherwise .} \end{cases}$$

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 $f \# g = f$ are not possible (see Section 3).

However, the examples give also 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 (counter)example generator [21]. 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 the number of possible algebras (up to isomorphism) when (a) only the axioms for composition ($\cdot$) are considered, (b) when only the axioms for interaction (#) are used and finally (c) the number of CAs. This experiment verifies that the actual number of algebras is really limited and suggests looking for a general representation theorem for CA.

6.2 Towards a General Representation Theorem

Since $(F, \cdot, 1)$ is a commutative group and $(F, \#, 1)$ is a commutative semigroup CA $(F, \cdot, \#, 1)$ also forms a commutative semiring with annihilator 1 but without a multiplicative neutral element. To take advantage of this structure, particularly semiring ideals, we first have a look at special elements of the semiring.

An element f of a CA is *erasive* iff

$$\forall\, g \in F : f \# g = 1 \ .$$

It turns out that erasive elements play a central role for the construction of models for the # operator. First, we set up a connection with the strict order $<$ from Section 3.1.

Lemma 6.1 *An element $f \neq 1$ is erasive iff it is maximal with respect to $<$.*

Proof. By contraposition of Equation (2), we get $f \# g \neq 1 \Rightarrow f \neq 1 \wedge g \neq 1$.
($\Rightarrow$) Assume $f \neq 1$ to be maximal but not erasive. Then there is an element g with $f \# g \neq 1$. The above remark yields $g \neq 1$ and therefore $f < f \# g$, which is a contradiction to the maximality of f.
($\Leftarrow$) Assume $f \neq 1$ to be non-maximal but erasive. That means that there is an element $g \neq 1$ with $f < g$, and by definition $\exists h : f \# h = g$. Since f is erasive, we get $g = 1$, which yields a contradiction. □

For finite $F \neq \{1\}$ there exists at least one maximal and hence erasive element, which is not equal to 1.

Lemma 6.2 *The set E of erasive elements is closed under composition $\cdot$ and interaction $\#$ with arbitrary elements of F and hence is a semiring ideal.*

Proof. Assume two erasive elements f, g and an arbitrary element h. By distributivity we get $(f \cdot g) \# h = (f \# h) \# (g \# h) = 1 \cdot 1 = 1$.

Now let f be erasive and g arbitrary. Then, for $h \in F$, $(f\#g)\#h = f\#(g\#h) = 1$ follows by associativity. □

Lemma 6.3 *The set E even forms a subtractive ideal (e.g. [1]), i.e., $f \in E \wedge f \cdot g \in E \Rightarrow g \in E$.*

Proof. Assume $f, f \cdot g \in E$ and $h \in F - E$. Then $1 = (f \cdot g)\#h = (f\#h) \cdot (g\#h) = 1 \cdot (g \# h) = g \# h$ and therefore $g \in E$. □

Lemma 6.4 *If $f \cdot g$ is erasive, then $f \# g = 1$ and $f \# h = g \# h$ for any $h \in F$.*

Proof. The first claim can be shown by $1 = (f \cdot g) \# g = (f \# g) \cdot (g \# g) = f \# g$. The second one by $1 = (f \cdot g) \# h = (f \# h) \cdot (g \# h) \Leftrightarrow (f \# h) = (g \# h)$. □

Next we link erasive elements with the equivalence relation $\sim$ from Section 3.2. Remember that $[x]$ denotes the equivalence classes w.r.t. $\sim$.

Lemma 6.5 *The set of all erasive elements forms an equivalence class: $E = [1]$. Hence e is erasive iff $e \sim 1$.*

Next we investigate *generating systems* w.r.t. the interaction operator $\#$, i.e., subsets $G \subseteq F$ such that every element in the image set of $\#$ equals a combination $g_1 \# \cdots \# g_n$ 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 6.6 *Let G be a minimal generating system for $\#$. Then no distinct elements of G can be related by $\sim$.*

Proof. A relation $g_1 \sim g_2$ would mean that in every $\#$-product of the above form product 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. □

This admits the following.

Theorem 6.7 *Let G be a minimal generating system for $\#$. 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 $[f] \cdot [g] =_{df} [f \cdot g]$ and $[f] \# [g] =_{df} [f \# g]$.*

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 our conjecture of the representation theorem:

(a) The smallest element (w.r.t. $\subseteq$) of each class is a product of non-erasive base colors, e.g., $b = b \cdot 1$ or $b \cdot c \cdot e = (b \cdot c) \cdot e$.
(b) 1 is the empty product of such colors; and
(c) All other elements are formed by composition with every possible combination of the erasive base colors.

This motivates us to state the following,

Conjecture The image of $\#$ is isomorphic to the power set of the erasive base colors. Let further N be the set of all non-erasive base colors. Then the elements $x \in 2^N$ form a system of representatives for the equivalence classes, where $[x] = x \cdot E =_{df} \{x \cdot y \,|\, y \in E\}$.

The last part of the conjecture is partially substantiated by the following property:

Lemma 6.8 *The equivalence classes $[x]$ under $\sim$ are closed under composition with erasive elements:*

$$\forall\, e \in E : x.e \subseteq [x] \ .$$

Proof. This is immediate from Lemma 3.5(b) and Lemma 6.5. □

7 A Model of Variation Points

Although the set-theoretic model given in Section 4 is interesting, 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" [22]. In the literature variation points are also called *extension points* (e.g., [19]) or *hot spots* (e.g., [15]).

The model we present in this section can be used for FO directly, as it is based on variation points and code fragments. We assume the 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.

An illustrative example showing 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.

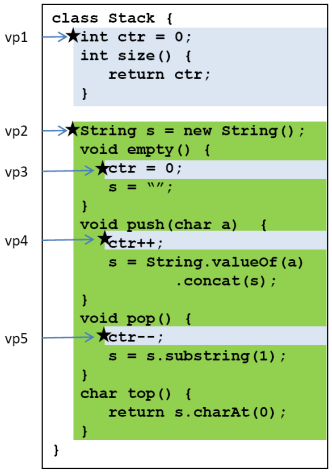

Fig. 1: The Counted Stack With Variation Points [10]

Our model will be based on the set model presented before. Therefore we only consider code fragments that commute with each other. Otherwise commutativity of CA would be lost. 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 $p : VP \to 2^{VP \cup C}$.[7] The semantics of the program is as follows: if, for a variation point $\mathtt{vp}$ the value $p(\mathtt{vp})$ is not the empty set then $p(\mathtt{vp})$ is installed at $\mathtt{vp}$; if $p(\mathtt{vp}) = \emptyset$, then the variation point remains empty.

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

$$white(\mathtt{vp}) = \begin{cases} \{\, \mathtt{class\ Stack\{\ vp_1\ vp_2\ \}} \,\} & \text{if } \mathtt{vp} = \mathtt{start} \\ \emptyset & \text{otherwise .} \end{cases}$$

This function coincides with the "white" part of Figure 1, while the blue fragment is given by

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

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

To build a program from a given function p, we just replace all occurrences of $\mathtt{vp}$ by its value $p(\mathtt{vp})$. This yields a function with no variation points in 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 question that remains is how to define feature composition and feature interaction.

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

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

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 variation point $\mathtt{vp}_1$ are disjoint; the composition unites the values. The values at variation point $\mathtt{vp}_2$ are identical; the composition removes them and leaves the variation point "unset", i.e., it ends up having the empty set as its value. The values at variation point $\mathtt{vp}_3$ 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:

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

In this paper we have shown that, for a given size, each definition of $\cdot$ is isomorphic to 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 # offers much more flexibility. Of course, we cannot give a compact definition, 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 guide lines on how repairs should be defined by the lemmas given in Section 3. Most important is Lemma 3.4 which can easily be implemented and offers a quick sanity 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—a (counter)example generator, which is shipped together with Prover9—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 (#) 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. Johnson, R., Foote, B.: Designing reusable classes. Journal of Object Oriented Programming 1(2), 22–35 (1988)
16. 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)
17. Kargapolov, M., Merzliakov, I.: Fundamentals of the Theory of Groups. Graduate texts in mathematics, Springer (1979)
18. 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)
19. Kästner, C.: Virtual Separation of Concerns: Toward Preprocessors 2.0. Ph.D. thesis, University of Magdeburg (2010)
20. 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)
21. McCune, W.W.: Prover9 and Mace4. <http://www.cs.unm.edu/~mccune/prover9>, (accessed May 8, 2012)
22. 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)

A Omitted Properties

We briefly list a number of properties for feature interaction, which can be useful for more sophisticated calculations but do not have a nice "semantical

counterpart" in FOP, hence we did not list them in the main text. For arbitrary elements f, g, h, i of a CA $(F, \cdot, \#, 1)$ the following holds:

(a) $(f \cdot g) \# f = f \# g$,
(b) $(f \cdot i) \# g = f \Rightarrow f = i \# g$,
(c) $(f \cdot i) \# g = f \Leftrightarrow f = (f \# g) \cdot (i \# g) \Leftrightarrow f \cdot (f \# g) = i \# g$,
(d) $(f \cdot i) \# g = f \Rightarrow f \# g = 1$,
(e) $f \# g = f \cdot i \Rightarrow g \# i = f \cdot i$,
(f) $f \# g = f \cdot i \Rightarrow f \# i = 1$,
(g) $f \# g = h \cdot i \Rightarrow f \# g \# h = h \# i$,
(h) $f \# g = i \wedge f \# h = i \cdot e \Rightarrow f \# (g \cdot h) = e$,
(i) $f \# g = h \wedge h \# i = f \Rightarrow h = 1 \wedge f = 1$.

Propositions (a)–(d) deal with the interaction of composed elements, (e)–(f) with the situation in which the interaction of two elements is a composed element. Implication (h) can be interpreted as "double reparing is useless".

B Prover9-Template

In this appendix we give a proof template for CA to be used with Prover9 and Mace4 [21]. Prover9 is an automated theorem prover for first-order and equational logic and Mace4 searches for finite models and counterexamples, both are freely available. The template can be used as input file, Prover9 will then offer a proof for the conjecture when found.

```
% LANGUAGE SPECIFICATION
 op(500, infix, ";").        % composition
 op(500, infix, "+" ).       % interaction
 op(500, infix, "X").        % full interaction
% AXIOMS
 formulas(sos).
   % --- axioms composition
      x;(y;z) = (x;y);z.
      x;y = y;x.
      x;x = 1.
      x;1 = x.
   % --- axioms interaction
      x+(y+z) = (x+y)+z.
      x+y = y+x.
      x+x = 1.
      x+1 = 1.
      x+(y;z) = (x+y);(x+z).
   % --- full interaction
      x X y = (x+y);(x;y).
  end_of_list.
 % CONJECTURE
  formulas(goals).
    %lemma to be proven, e.g.
    %(x+y);((y+z);(x+z)) = (x;y)+(x;z).
  end_of_list.
```