An Extension for Feature Algebra

Peter Höfner Bernhard Möller

October, 2009

Introduction

Feature-oriented Software Development

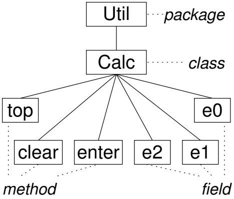
- general programming paradigm
- provides formalisms, methods, languages, and tools
- builds variable, customisable, and extensible software

Feature Algebra

- algebraic framework
- captures many of the common ideas of FOSD
- formal foundation of architectural metaprogramming
- automatic feature-based program synthesis

aim: full congruence between tools for FOSD and feature algebra

A Standard Model

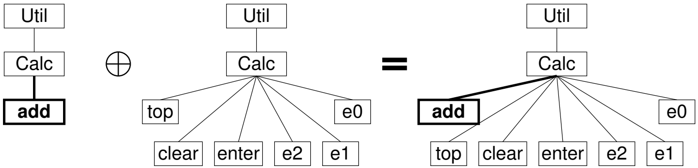


Feature Structure Trees (Forests)

- capture the hierarchical module structure of a system
- can be encoded using strings of node labels

$$\begin{aligned} Base =_{def} \{ & Util,\ Util :: Calc,\ Util :: Calc :: top, \\ & Util :: Calc :: clear,\ Util :: Calc :: enter, \\ & Util :: Calc :: e0,\ Util :: Calc :: e1, \\ & Util :: Calc :: e2 \} \end{aligned}$$

A Standard Model



- order of treebranches does not matter
- feature tree superimposition (addition) can be defined as set union
- feature algebra also comprises modifications which in the concrete model are tree rewriting functions
  (e.g. renaming a node, i.e., renaming a class)

(in the paper we introduce another model that respects the ordering;
based on lists of maximal paths)

Feature Frameworks

Feature Algebra

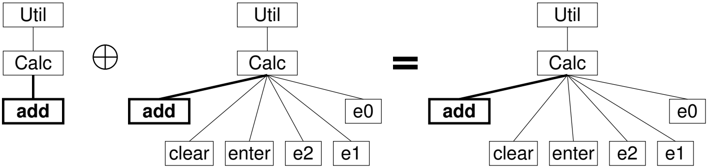
- a tuple $(M, I, +, \circ, \cdot, 0, 1)$ satisfying some axioms like distributivity and

$$i + j + i = j + i$$

I	$\leftrightarrow$	set of introductions
M	$\leftrightarrow$	set of modifications
$+$	$\leftrightarrow$	superimposition
$\circ$	$\leftrightarrow$	composition
$\cdot$	$\leftrightarrow$	application
$i \leq j \Leftrightarrow_{def} i + j = j$	$\leftrightarrow$	subsumption relation (preorder)

- closely related to the DEEP calculus
- definition contains only first-order equational axioms
- predestined for automatic theorem proving

Idempotence



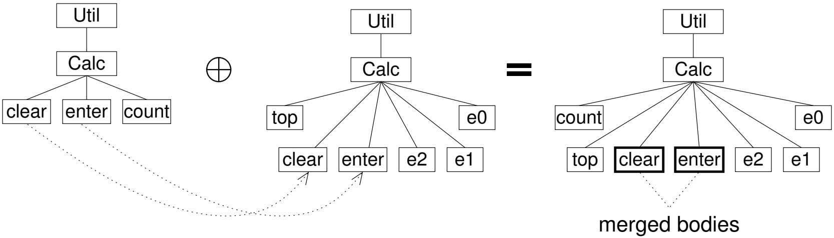
- feature algebras are idempotent
- "duplicating a feature has no effect"
- formally: $i + j + i = j + i$
- for the standard models this fits perfectly
- does not consider feature oriented programming at code level

Feature Frameworks

FeatureHouse

- concrete tool for performing the operations of a feature algebra
- developed by Apel, Kästner and Lengauer
- composition of features written in various languages
  (Java, C#, C, Haskell, and JavaCC)

Extending the Model



- each terminal node is extended by a code fragment
- if two instances occur, the code parts have to be merged
- order of combination *does* matter
- Java example: "updating a function `foo` with an increment statement"

Bar::foo
```
void foo(int a) {
  a++;
  original(a);
}
```

The Lost Idempotence

```
Bar::foo
void foo(int a) {
  a++;
  original(a);
}
```

$\oplus$

```
Bar::foo
void foo(int a) {
  a++;
  original(a);
}
```

$=$

```
Bar::foo
void foo(int a) {
  a++;
  a++;
  original(a);
}
```

- update using FeatureHouse
- no details how FeatureHouse merges code and applies overriding
- the idempotence is lost

Consequence

Mismatch between `FeatureHouse` and the simple definition of feature algebra

Consequence

Mismatch between `FeatureHouse` **and the simple definition of feature algebra**

aim: find a more sophisticated and adequate algebra

Extended Feature Algebra

let C be an abstract set of code fragments

- consider pairs (i, c) where
  i is an introduction corresponding to a maximal path in the forest and
  $c \in C$ is the code fragment contained in the leaf
- pairs (i, c) are denoted by $i[c]$
- add an update or override operation $| : C \times C \to C$ such that $|$ is associative and refines $i + j + i = j + i$ to

$$i[a] + j[c] + i[b] = j[c] + i[a|b]$$

- the original definition can be retrieved by choosing C as containing only the empty code fragment
- the subsumption relation will no longer be a pre-order, but still transitive

Refine the Model with Update

- identify the “common part” of two given implementations
- determine which part of a method body has to be overridden
- determine which part has to be preserved
- highly dependent on the respective language
- define abstract interfaces:

```
int min5(int a) {
  int b=5;
  if(a<b) return a;
  else return b;
}
```

```
int min5(int a) {
  int b;
}
```

- a precise definition of the abstract interface will need to reflect also nested scopes

Refine the Model with Update

$$
\begin{array}{rcll}
X \ddagger U & = & \{x \in X \mid ai(x) \in U\} & \text{restriction} \\
X - U & = & \{x \in X \mid ai(x) \notin U\} & \text{removal}
\end{array}
$$

- $\ddagger$ determines for a set X whose corresponding abstract interfaces lie in a given set U
- the operator $-$ selects its relative complement

$$X|Y =_{def} (Y - ai(X)) \cup X$$

where $ai(X) =_{def} \{ai(x) \mid x \in X\}$.
abstraction to the level of feature algebra is possible
automated reasoning still possible

Conclusion & Outlook

- further step towards an algebraic theory of FOSD
- experience with feature algebra and `FeatureHouse`
- two concrete models for feature algebra
- models and `FeatureHouse` do not coincide
- extended feature algebra is introduced
- additional operators can be modeled

- ongoing work
- all introduced operators like update need further investigation
- check whether the extension is adequate

Appendix

Feature Algebra

a tuple $(M, I, +, \circ, \cdot, 0, 1)$ such that

- $(I, +, 0)$ is a monoid satisfying the additional axiom of distant idempotence, i.e., $i + j + i = j + i$.
- $(M, \circ, 1)$ is a groupoid operating via $\cdot$ on I, i.e., $\circ$ is a binary inner operation on M and 1 is an element of M such that furthermore
  - $\cdot$ is an external binary operation from $M \times I$ to I
  - $(m \circ n) \cdot i = m \cdot (n \cdot i)$
  - $1 \cdot i = i$
- 0 is a right-annihilator for $\cdot$, i.e., $m \cdot 0 = 0$
- $\cdot$ distributes over $+$, i.e., $m \cdot (i + j) = (m \cdot i) + (m \cdot j)$
- the natural preorder is defined by $i \leq j \Leftrightarrow_{def} i + j = j$

I	$\leftrightarrow$	set of introductions (abstraction of feature trees)
M	$\leftrightarrow$	set of modifications (rewrite functions)
$+$	$\leftrightarrow$	feature tree superimposition
$\cdot$	$\leftrightarrow$	application of a modification to an introduction
$\circ$	$\leftrightarrow$	modification composition