Feature Interactions, Products, and Composition

Don Batory  
Peter Höfner  
Jongwook Kim  

U of Texas at Austin  
U of Augsburg  
U of Texas at Austin
Introduction

- Relationships between feature modules and feature interactions not well-understood
- Review classic examples of feature interactions
- Discovered that features are composed by three operations:
  - Sequential $+$
  - Interaction $\#$
  - Cross-product $\times$
- Created a formal model, built a tool that solves an interesting and open problem in FOSD tooling and theory
You (should) Know This

• Want to synthesize programs in a software product line (SPL)

• Building blocks are increments in functionality called features
  features in domain \{ a, b, c, d \ldots \} 

• Feature modules encapsulate classes and fragments of classes

• Programs of a domain are sequential (+) compositions of feature modules
  \[ p = a + b + c \]
Feature Interactions

- Classic example
- Design of a building (b)
- Fire control feature
- Flood control feature

- fire$flood$ module repairs (changes) the fire and/or flood modules so that they work correctly together

- Interaction module

\[
\text{fire detected } @i \\
\text{sprinklers on } @i+1 \\
\text{standing water } @i+2 \\
\text{water off } @i+3
\]
Leads to 3 Operations on Features

- Sequential composition +
- Interaction composition #
- Cross-product composition \times

When architects want features a and b, they want the cross-product of a and b, because they want a and b to work together correctly

\[ a \times b = (a \# b) + a + b \]

a, b, and a\#b are modules

Question: what are the axioms that govern these operations?
Basics

- Kästner's CIDE (ICSE 2009)
- Paint the “blue” feature blue
- Paint the “green” feature green

- Overlapping colors indicate feature interactions

\[
\text{blue} \# \text{green} = \text{green} \# \text{blue}
\]
Variation Points and Fragments

- Point at which a code fragment is inserted is a variation point (VP)
- VPs are paired with (precisely one) fragment
Modules and Module (+) Composition

• A feature module is a collection of fragments

• Sequential composition of modules:  \( a + b \)
  • union of their fragments
  • pairing of fragments with their VPs
  • if pairing is not possible, save for later pairing

• Very simple
Axioms of +

• Sequential composition is a “union-like” operation; axioms that govern + are:

• **Identity:** \( a + 0 = a \)

• **Commutative:** \( a + b = b + a \)

• **Associative:** \( (a + b) + c = a + (b + c) \)
Axioms of \#

- $a\#b$ is the set of changes that are needed to make $a$ and $b$ work together correctly

- Again, not too difficult to recognize axioms

- No Interaction: $a\#0 = 0$

- Commutative: $a\#b = b\#a$

- Associative: $(a\#b)\#c = a\#(b\#c)$
Additional Properties

• # binds stronger than +

\[ a \# (b + c) = (a\#b) + c \]

• Distributivity Laws: # distributes over +

\[ a\#(b + c) = a\#b + a\#c \]

• From #-commutativity we immediately know:

\[ (b + c)\#a = c\#a + c\#b \]
Theorems of $\times$

• Follows from axioms of $+$ and $\#$

• Identity: $a \times 0 = a$

• Commutative: $a \times b = b \times a$

• Associative: $(a \times b) \times c = a \times (b \times c)$
Dark Corners Of Features

- What do the following mean?
  
  \[ r + r = ? \] what are the semantics of replicated features?
  
  \[ r \# r = ? \] can a feature interact with itself?
  
  \[ r \cdot r = ? \]

- Do features have inverses?

- Fundamental questions - who's answers we've largely avoided

  Their answers depend on each other...
Replicated Features

- Classical feature modeling precludes replicated features: they never occur, so \( r+r, r\#r, \) and \( r\#r \) never arise.

- But replication arises in CIDE in an odd way:

\[
b\#b + b = b
\]

\[
r\#b\#r + b\#r + r = b\#r + r
\]

- Do we allow meaningless distinctions \( r\#r, r\#r\#r, r\#b\#r, \) etc?
No!

• \#-Involution: \( r\#r = 0 \) (features do not interact with themselves)

\[
\begin{align*}
0 + b &= b \\
0 + b\#r + r &= b\#r + r
\end{align*}
\]

• What about replicated features?

\( r+r = ? \)

• Features are either present or they are not
They toggle from being present to being absent

• +-Involution: \( r+r = 0 \) (modules are inverses of themselves)

• \( \square \)-Involution: \( r\square r = 0 \) (follows from above)
Consistency and Irredundancy of Axioms

- Consistent – they don’t contradict themselves (Mace4)
- Irredundant – using Prover9
- Couldn’t have addressed these questions without tools
Reactions

• We don’t like involutions!!

    Especially \( r + r = 0 \)

• We like inverses!

• In effect, + involution is equivalent to:

    • allowing inverses: \( r + r^{-1} = 0 \)
    • assert their equality: \( r = r^{-1} \)

• Eliminates questions like: what is: \( a + r^{-1} + b + s^{-1} \)
Reactions

• We don’t like the way you handle replicated features:
  1. our solution – which you don’t like
  2. never admit them – we can’t do this with CIDE
  3. idempotence!  \( r + r = r \)

• Here’s the problem: idempotence and inverses don’t mix:
  \[
  r = r + 0 = r + (r + r^{-1}) = (r + r) + r^{-1} = r + r^{-1} = 0
  \]
So What?
Here’s a Practical Problem

• Source of a program is produced and given to a client

• Client modifies the program (to fix bugs, improve performance)

• Back-propagate changes to the original modules to make the updates permanent

• What to back-propagate and how to propagate?

• Client request program

\[ P = T_1 + T_2 + \ldots + T_n \]

• Manually modifies \( P \) to

\[ Q = T_0 + T'_1 + \ldots + T_n \]

\[ = \Box P + P \]

• Solve for \( \Box P \)
Benefit: Can Solve Equations!

\[ \Delta P + P = Q \] // given

\[ \Delta P + P + P = Q + P \] // add P to both sides

\[ \Delta P = Q + P \] // cancel terms by involution

\[ \Delta P = T_0 + T'_1 + \cdots + Tn + T_1 + \cdots + Tn \] // substitute Q, P

\[ \Delta P = \boxed{T_0 + T'_1 + T_1} \] // cancel terms by involution
Paan

- Office Open XML is a standard set of XML schemas adopted by Microsoft Office as its default file format
  - different schemas for Word, Visio, etc.

- Paan is a tool that works with Word documents using the Custom XML Markup facility of MS Word to color regions of text

```xml
<?xml version="1.0"?>
<w:document xmlns:wne="...">
  <w:body>
    <w:customXml w:uri="mySchema" w:element="blue">
      <w:p>
        <w:r>
          <w:t>Hello World</w:t>
        </w:r>
      </w:p>
    </w:customXml>
  </w:body>
</w:document>
Feature-Based Word Documents

- Are painted by colors
- Feature-customized variants of a Word document created by projecting (removing) document fragments, while leaving their variation points behind
- Projected documents can be updated, and back-propagated
Paan Also Supports Wrappers

- Color nesting is reversed
- Wrappers found in OO code

- Can back-propagate edits to wrappers, too
Paan Lessons

• It’s hard to implement “coloring” in MS Word

• Word was never designed for this capability

• Semantics of custom XML markup facility not defined - so a lot of guesswork and hacking to make “coloring” work

• Tools must be developed initially with coloring in mind

• MS Office 2010 no longer supports XML markup facility
Conclusions

• Features and feature interactions have long been of interest

• FOSD brings a twist in its focus on feature modularity and the composition of feature modules to build programs of SPLs

• We now have an algebra that gives precise relationships between features and feature interactions
  • make explicit previously implicit operations $\#$ and $\sqcap$
  • formal model of the CIDE tool, and how to build new tools
  • general formalism to back-propagate changes

• Step closer to understand science of design, where automated software design is governed by mathematics, not ad hoc programming