CONSTRAINTS IN FEATURE ALGEBRA

Andreas Zelend

Universität
Augsburg
University

13th Int'l Conference on Relational and Algebraic Methods in Computer Science
20. September 2012

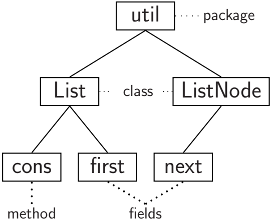
THE FEATURE ALGEBRA

GOALS

- language independent and formal characterisation of important aspects of **feature oriented software development (FOSD)**
- build an algebraic foundation for feature-structure-forests (FSFS), which represent the hierarchical structure of programs (similar to abstract syntax trees)
- provide two operations: superimposition $+$ to compose features (recursive merging of FSFs), and modification application $\cdot$ to modify certain parts of a feature

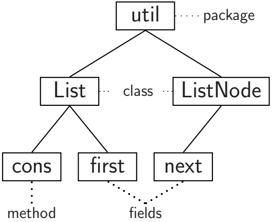
EXAMPLE FST/FSF

```
//feature CONS
package util;

class List{
  ListNode first;

  void cons(ListNode n){
    n.next = first;
    first = n; }
}

class ListNode{
  ListNode next;
}
```



Example FST/FSF

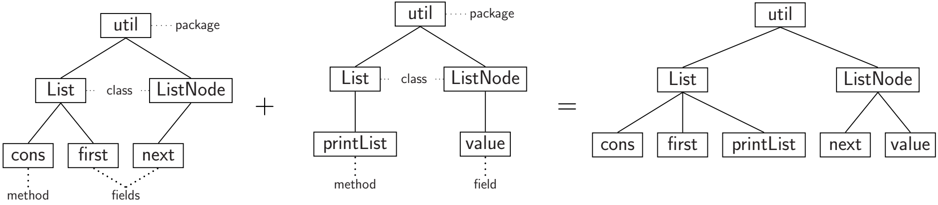
```
//feature CONS
package util;

class List{
  ListNode first;

  void cons(ListNode n){
    n.next = first;
    first = n; }
}

class ListNode{
  ListNode next;
}
```



Or as a prefix-free set:

$$\{util.List.cons,\ util.List.first, util.ListNode.next\}$$

Example superimposition



$$\begin{array}{l}\{util.List.cons,\\ \ util.List.first,\\ \ util.ListNode.next\ \}\end{array} + \begin{array}{l}\{util.List.printList,\\ \ util.ListNode.value\ \}\end{array} = \begin{array}{l}\{util.List.cons,\\ \ util.List.first,\\ \ util.List.printList,\\ \ util.ListNode.next,\\ \ util.ListNode.value\ \}\end{array}$$

FEATURE ALGEBRA

DEFINITION (APEL, LENGAUER, MÖLLER, KÄSTNER 2010)

A **Feature Algebra** is a structure $(M, I, +, \circ, \cdot, 0, 1)$, with the following properties:

- $(I, +, 0)$ is a monoid in which **distant idempotence** holds, i.e., $i + j + i = j + i$.
- I is a external monoid over the structure $(M, \circ, 1)$, i.e.,
  - $\cdot$ is a external binary operation $\cdot : M \times I \mapsto I$
  - $(m \circ n) \cdot i = m \cdot (n \cdot i)$
  - $1 \cdot i = i$
- 0 is a right annihilator, i.e., $m \cdot 0 = 0$
- $\cdot$ distributes over $+$, i.e., $m \cdot (i + j) = (m \cdot i) + (m \cdot j)$

Elements of I are called **introductions**.

TYPES OF CONSTRAINTS

We distinguish three main classes of constraints:

- **Low-level constraints** stem from code level. An example are **dependences** like "feature F builds on feature G".
- **High-level constraints** mainly originate from the program model or the problem domain.
    - Ex: An optional feature
- An example of a **non-functional constraint** is that "the product must run on a mobile phone and has to have less than 1 Mb of compiled source code".

LOW-LEVEL CONSTRAINTS

- **References** describe all code-level constraints where a method, a class, etc. refers to another object.The same type of constraint occurs when program parts are **import**ed.

```
//feature F2
class C1{
  void foo(){
    o.something();
     ... }
}
```

```
//feature F1
class C1{
  Object o;
}
```

FIGURE : Low-level Constraint: Reference

LOW-LEVEL CONSTRAINTS

- **Refinements** are similar to references. In jak, the keyword **refines** indicates that a feature builds on another. The Java keyword **extends** has the same effect.

```
//feature F3
class C3{
  ...
}
```

```
//feature F4
refines class C3{
  ...
}
```

FIGURE : Low-level Constraint: Refinement

We call references and refinements, which behave similarly, **structural dependences**.

LOW-LEVEL CONSTRAINTS

- **Abstract Class Constraints** and **Interface Constraints**. A concrete subclass **class** C of an **abstract class** or **interface** A must implement all inherited abstract methods. Features may introduce new classes inheriting from A or may introduce new abstract methods into A.

```
//feature F5
abstract class C5{
  abstract void foo();
}
```

```
//feature F6
class C6 extends C5{
  void foo(){};
}
```

Figure : Abstract Class Constraint

HIGH-LEVEL CONSTRAINTS

Mandatority

- A certain feature has to be present in a product.
- Ex: A user requires a `toString()`-method for each class.

Optionality

- A feature is optional in an product line.
- Ex: The optional feature F may be part of product P, whereas another product Q does not have F.

Alternative

- provides a choice from a given set of features. It can be seen as "exactly one of m different features".

HIGH-LEVEL CONSTRAINTS

Exclusion

- Two features are not allowed to be within the same product.
- Ex: If a product P has a 64-bit implementation of foo(), P is not allowed to have a 32-bit implementation of the same method.

Implication

- A second feature is required.
- Ex: If P provides a method (feature) to allocate memory, another feature for deallocation has to be provided.

Requirements

- The dependences between features are given by the feature model or the user.
- Ex: A customer demands the implementation of a printer driver whenever a function print () is implemented.

CONSTRAINTS IN FEATURE ALGEBRA

The main idea is to use triples (i, d, c) consisting of

- an introduction i
- a collection d of structural dependences, again represented by an introduction
- a condition c

Constraints in Feature Algebra

Example

```
//feature PRINT_LIST
package util;

class List{

  public void printList(){
    ListNode iter = first;
    while (iter != null){
     System.out.print(iter.toString()+",");
     iter = iter.next; }
  }
}

class ListNode{
  Object value;
}
```

Figure : Implementation of PRINT

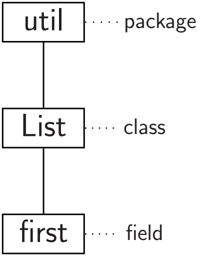


Figure : Structural Dependence of PRINT

CONSTRAINTS IN FEATURE ALGEBRA

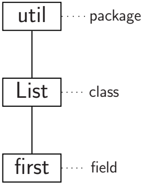
EXAMPLE

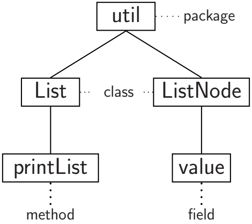
- The introduction of PRINT ($impl$) is just its FSF
- Its structural dependence ($sdpl$) is the FSF given before
- PRINT does not impose any condition, i.e., its condition is the constant predicate **true**

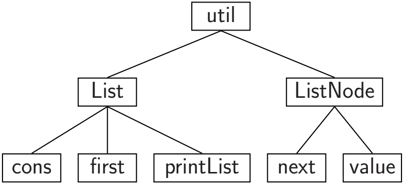
The design D_p can now be defined as $(impl, sdpl, \mathbf{true})$.

CONSTRAINTS IN FEATURE ALGEBRA

EXAMPLE







- D_p does not satisfy its dependence ($sdpl \not\subseteq impl$).
- In contrast CONS + PRINT does, since it includes $sdpl$ as a subtree. ($sdpl \subseteq$ CONS + PRINT)

CONSTRAINTS IN FEATURE ALGEBRA

We model the classes of constraints using a predicate $has(F)(i)$ that checks whether a feature F is included in a given Feature Algebra element $i \in I$. If F can be represented as an introduction f, this can be expressed as the condition $has(f)(i) \iff_{df} f \leq i$.

- Feature exclusion: $has(F)(i) \implies \neg has(G)(i)$
- Feature implication: $has(F)(i) \implies has(G)(i)$.
- Abstract class constraints (and interface constraints): $extends_D(i) \implies has(\textsf{foo()})(i)$

CONSTRAINTS IN FEATURE ALGEBRA

DEFINITION

Let $A = (M, I, +, \circ, \cdot, 0, 1)$ be a Feature Algebra and $\mathcal{P}_I$ be the set of all predicates over the introductions I of A.

- A **design** over A is an element of $I \times I \times \mathcal{P}_I$
- A design (i, d, c) **satisfies** its **dependence** d iff $d \leq i$
- A design (i, d, c) **satisfies** its **condition** c iff $c(i) =$ true
- A design that satisfies its dependence and its condition is called a **product**

DEFINITION

The **conjunction** of predicates $p, q \in \mathcal{P}_I$ is defined by $(p \wedge q)(i) =_{df} p(i) \wedge q(i)$ for all $i \in I$. The predicate that maps every element of I to true is denoted by $\mathbf{true}$.

CONSTRAINTS IN FEATURE ALGEBRA

DEFINITION

Assume an arbitrary set I, the set of predicates $\mathcal{P}_I$ over I and predicates $p, q \in \mathcal{P}_I$.

- By $\mathcal{PT}_I$ we denote the set of all **predicate transformers**
- A predicate transformer t is **conjunctive** if $t(p \wedge q) = t(p) \wedge t(q)$ holds for all $p, q \in \mathcal{P}_I$ (e.g. $id(p) = p$ is conjunctive)
- The set of all conjunctive predicate transformers over A is denoted by $\mathcal{CT}_I$.
- A predicate p is called **stable** iff $p(i) \implies p(i + j)$ for all j.

CONSTRAINTS IN FEATURE ALGEBRA

LEMMA

For a set I of introductions the structure $(\mathcal{CT}_I, \mathcal{P}_I, \wedge, \circ, \cdot, \textbf{true}, id)$ forms a Feature Algebra.

Now the following result is immediate by universal algebra.

THEOREM

For a Feature Algebra $A = (M, I, +, \circ, \cdot, 0, 1)$ the structure DES$_A$ $=_{df}$ $(M \times M \times \mathcal{CT}_I, I \times I \times \mathcal{P}_I, +, \circ, \cdot, \mathbf{0}, \mathbf{1})$ forms a Feature Algebra of designs if $\mathbf{0} =_{df} (0, 0, \textbf{true})$, $\mathbf{1} =_{df} (1, 1, id)$ and the operations as well as modifications are lifted pointwise.

CONSTRAINTS IN FEATURE ALGEBRA

LEMMA

If the designs $f = (i, d, b)$ *and* $g = (j, e, c)$ *are products and* b *and* c *are stable then the composition* $f + g$ *is also a product. Furthermore if* f *is a product and* $(t \cdot b)(m \cdot i) = b(i)$ *then* $(m, m, t) \cdot f$ *is a product.*

- A design with a stable condition can be composed with another design while the condition does not change its value
- For example all $has(f)(i)$ conditions are stable

CONCLUSION AND OUTLOOK

- We have shown how constraints can be embedded into the abstract structure of Feature Algebra

- Future work will be directed towards representative case studies to gain a better insight into Feature Algebra and constraints