Feature Oriented Programming (FOP)

Eduardo Figueiredo

http://www.dcc.ufmg.br/~figueiredo

Definition of Feature

- A feature is an increment in program development or functionality

- Features can be classified
  - Mandatory features
  - Optional features
  - Or features
  - Alternative features (XOR)

Program Configuration

- Programs are described or differentiated by features
  - Entry-level versions may have a minimal set of features
  - Premium versions include most features
- The ability to add and remove features implies that features are modularized

Feature Oriented Programming

- FOP is the study of programming models that support feature modularity
  - It is a general paradigm for programming software product lines
- FOP is based on step-wise refinements

Step-wise Refinement

- Step-wise refinement advocates that complex programs can be constructed from simple programs
  - By incrementally adding details
- In FOP, simple incremental units are called features

Example of Refinement

- Let's consider two different implementations of a class C

```
class C {
  int field1;
  void method2() { ... }
  void method3() { ... }
  void method4() { ... }
}
```

equivalent

```
class C1 {
  int field1;
  void method2() { ... }
}

class C2 extends C1 {
  void method3() { ... }
}

class C extends C2 {
  void method4() { ... }
}
```

The FOP Idea

- FOP was inspired by layer-based designs and levels of abstraction in database systems
- A program can be seen as a stack of layers
  - Each layer adds functionality to previously composed layers
  - Different compositions of layers produce different programs

From Layers to Features

- Later, the idea of layers was generalized to features

- Why using features?
  - A program is composed of several artifacts, such as models and code
  - A feature unifies all artifacts in a single modular unit

FOP and AOP

- Some ideas behind FOP are similar to Aspect-Oriented Programming (AOP)
  - Both FOP and POA can be used to implement a software product line
  - Their goal is to make programs easy to extend and to maintain
  - They aim to improve separation of concerns

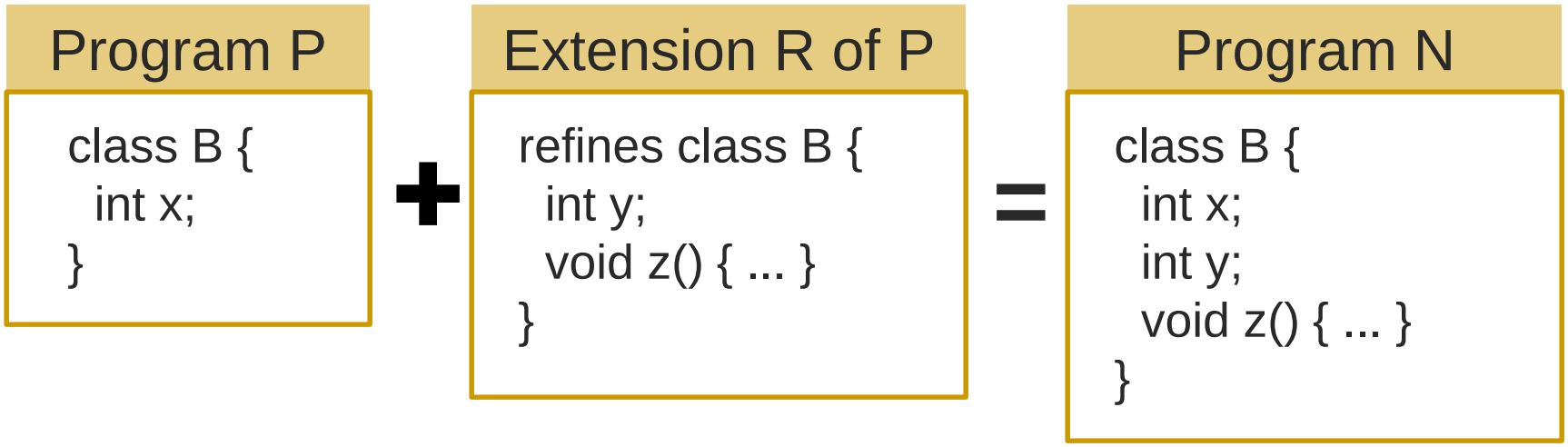
GenVoca

GenVoca

- GenVoca is a compositional paradigm for defining programs of a SPL
  - Combination of the names Genesis and Avoca
- GenVoca is based on the stepwise refinement of programs
  - Programs are constants
  - Refinements are functions

Example of Refinement

- Let's consider a program P
  - P has only a class B



Algebras as Programs

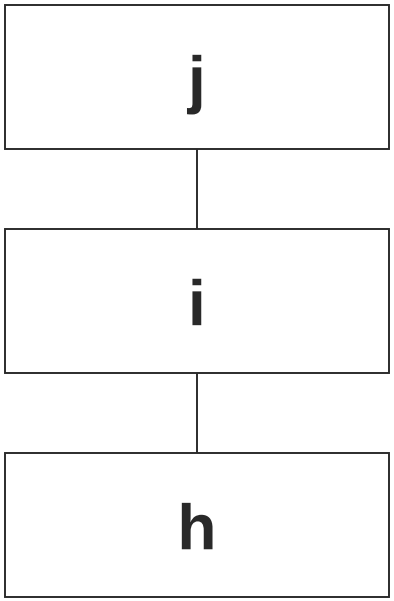
- We can use algebras to express programs
- In the previous example
  - The program P is a constant (base)
  - The refinement R is a function
  - The program N is a the resulting expression

- N = R(P) equivalent to N = R • P

Definitions

- GenVoca foundations
  - A program is a value
  - A feature (i.e., function) changes the program (i.e., value)
- If multiple expressions produce the same value, then these expressions represent equivalent programs
  - fun1 (fun2 (fun3))
  - (fun1 fun2) (fun3))

Examples of Notation



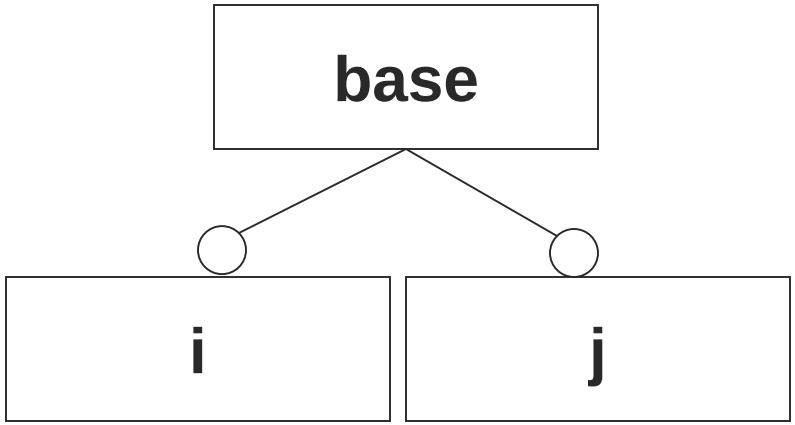
j (i (h))

j ● i ● h

GenVoca for Products

- GenVoca defines products of a software product line
  - Each expression is a product
  - Example: p1 = j • i • h
- Features are presented by functions
- Some combinations of features may not be possible
  - The feature model defines the possible combinations

Example of Feature Model



- Four expressions are possible
  - The symbol • means composition

base	:	no feature is added
i • base	:	only the feature i is added
j • base	:	only the feature j is added
i • j • base	:	both features are added

The Order Matters

- The order for feature application changes the resulting product
  - i • j • h is different of j • i • h

- For instance, p1 = i • j • h means
  1. Development starts from feature h
  2. Then feature j is added
  3. Finally, feature i is added

GenVoca Implementation

- The formalism behind GenVoca can be implemented in different ways
  - For instance, GenVoca can be implemented with conditional compilation (*#ifdef*) or with aspect-oriented programming
- More recently, GenVoca have been implemented with *mixin layers*
  - This solution is used by AHEAD

Bibliography

- D. Batory, J. Sarvela, A. Rauschmayer. **Scaling Step-wise Refinement**. IEEE Transactions on Software Engineering (TSE), pp. 355–371, 2004.
- D. Batory. **A Tutorial on Feature Oriented Programming and the AHEAD Tool Suite**. International Conference on Generative and Transformational Techniques in Software Engineering (GTTSE), 2005.