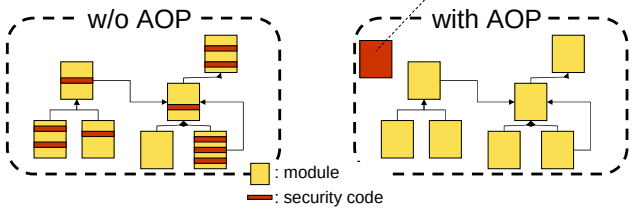
Relaxing Type Restrictions of Around Advice in Aspect-Oriented Programming

Hidehiko Masuhara (University of Tokyo)

Observation:
AspectJ's type system is too restrictive wrt around advice
Proposal:
A less restrictive weaving algorithm based on type inference

Introduction to AOP and *around advice*

AOP can modularize crosscutting concerns (e.g., security, logging) by defining aspects



AspectJ is an AOP language based on Java

Around advice

- runs aspect code ***instead of*** matching join points
  - aka method interceptors
- is useful for e.g.
  - object pooling: "upon new Session(), return an object from the object pool"
  - injecting augmented/different functionality: "return a stream to console instead of creating a file stream"
- is type checked (in AspectJ)

```
pointcut openFile(String fn):
  call(FileOutputStream.new(String)) && args(fn);

OutputStream around(String fn) : openFile(fn) {
  if (fn.equals("stdout"))
    return System.out;
  else return proceed(fn);  }
```

advice return type (T_a)

join point (return type: T_j)

```
OutputStream s = new FileOutputStream(fn);
StorableOutput output = new StorableOutput(s);
output.writeStorable(d);
```

used as OutputStream (T_u)

Print Stream

constructor call

FileOutput Stream

```
class FileOutputStream ... {
  FileOutputStream(String fn) {
    ...
```

a code fragment in JHotDraw that stores figures into a file

Problem: AspectJ compiler rejects around advice that returns objects of different type

– even if it is safe to do so by editing text

```
OutputStream s = new FileOutputStream(fn) System.out;
StorableOutput output = new StorableOutput(s);
output.writeStorable(d);
```

Analysis: too strict typing rules

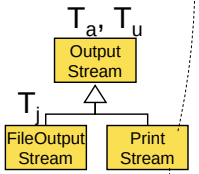
– AspectJ's typing rules require $T_a <: T_j$

– But we want to allow $T_a <: T_u$
if return values are used as T_u object

Proposal: Type Relaxed Weaving

- as an extended AspectJ weaver
- while preserving type safety

– Weaving algorithm Is to
- infer most specific type usage (T_u)
- apply advice only when $T_a <: T_u$



When is Type Relaxed Weaving useful?

– In OO-based AOP languages
- Many chances (see below)
- Often in practice, when creating anonymous objects like event handlers

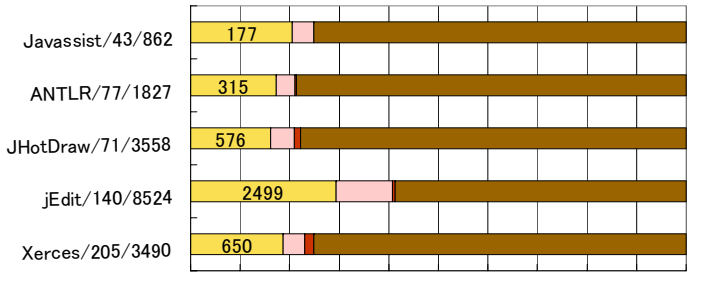
```
button1.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    /* do sometihng */ }
});
```

Compiled into creation of a class that implements ActionListener

– Could also be useful in non-AOP languages with interceptor-like mechanisms

Preliminary assessment: 15–30% locations can be relaxed in real world apps

- We dynamically profiled type usages of objects in 5 real world Java applications
- Usages are classified by usage type is more general than, more specific than, incompatible to, or same to creation type
- Result: 15–30% of locations can be relaxed



Note: a few classes are excluded from jEdit and ANTLR due to "too large code size". / Javassist 3.4 with two sample programs. / ANTLR 2.7.7 generating a Java parser. / JHotDraw 6.0b1 standalone application with manual operations. / jEdit 4.2 final with manual operations. / Xerces 2.9.0 with 8 DOM&SAX samples.