abc: an extensible AspectJ compiler

- Programming Tools Group, University of Oxford
- Sable Research Group, McGill University
- BRICS, Aarhus Universitet
The need for an AOP workbench

**new features:**
- parametric introductions
- dataflow pointcut
- remote pointcut
- predicted cflow
- tracecuts
- event-based AOP
- symmetric composition

**new analyses:**
- pure aspects
- static cflow
- thisJoinPoint escape

**code generation:**
- around without closures
- inline advice or not?

*shared tool building helps progress*
Goals of abc

a compiler workbench for AspectJ to:

- explore AOP language design space (this talk)
- experiment with better code generation
- experiment with static analyses for safety checks and optimisations

clarify AspectJ language definition:
- grammar
- scope rules for ITDS, ...
Does \textit{ajc} meet these goals?

proven workbench: AspectJ language was developed on it!

- fast compiler
- incremental compilation
- tight integration with Eclipse

\textit{ajc} has evolved from a research tool to a production compiler

Difficult to meet \textit{abc} goals:
- 119 changes to the text of the Eclipse Java compiler
- customised BCEL
- no LALR(1) grammar
- no analysis & optimisation framework
- designed for compilation speed
Extensibility of abc

- extensible frontend via *Polyglot*:
  - new syntax
  - new types
- extensible backend via *Soot*:
  - new joinpoints ⇐ focus of this talk
  - new analyses
  - new optimisations
How to add new joinpoint+pointcut?

- architecture of abc
- intermediate representation for pointcuts
- how to find shadows in code
- example: array access joinpoint
Architecture of abc

Polyglot and Soot are used without any modifications.

前端 (Polyglot)

.frontend

 backend (Soot)

Java AST

AspectJ AST

polyglot-based frontend

separator

code generation + static weaving

Jimple IR

advice weaving + postprocessing

bytecode

Aspect Info

.class

.java
Pointcuts and weaving in abc

Java code

Jimple

shadows

AspectJ pointcuts

intermediate pointcuts
(in AspectInfo)

matcher

advice lists

weaver

Jimple
How to add a new joinpoint+pointcut?

- architecture of abc
- intermediate representation for pointcuts
- how to find shadows in code
- example: array access joinpoint
IR for pointcuts

LexicalPointcut
EmptyPointcut
FullPointcut
Within
  - DirectlyWithin
  - WithinAdvice
  - WithinConstructor
  - WithinMethod
  - WithinStaticInitializer

DynamicValuePointcut
  Args
  TargetAny
    - TargetType
    - TargetVar
  ThisAny
    - ThisType
    - ThisVar

PointcutRef
And/Or/Not
If

CflowPointcut
  Cflow
  CflowBelow

LocalPointcutVars

CastPointcutVar

ShadowPointcut
ClassInitialization
ConstructorCall
Execution
GetField
Handler
InterfaceInitialization
MethodCall
Preinitialization
SetField

needed for inlining of named pointcuts

extend this hierarchy for new pointcuts
### Examples: AspectJ pointcut to IR

<table>
<thead>
<tr>
<th>AspectJ</th>
<th>Intermediate Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>execution(int Foo.foo(char))</code></td>
<td><code>withinmethod(int Foo.foo(char))</code>&lt;br&gt;``` &amp;&amp; execution()`</td>
</tr>
<tr>
<td><code>adviceexecution()</code></td>
<td><code>withinadvice()</code>&lt;br&gt;``` &amp;&amp; execution()`</td>
</tr>
<tr>
<td><code>call(Foo.new(int))</code></td>
<td><code>constructorcall(Foo.new(int))</code></td>
</tr>
<tr>
<td><code>initialization(Foo.new(..))</code></td>
<td><code>(withinconstructor(Foo.new(..))</code>&lt;br&gt;<code> &amp;&amp; classinitialization()`&lt;br&gt;</code></td>
</tr>
</tbody>
</table>

new pointcuts also need to be translated to IR
How to add a new joinpoint+pointcut?

- architecture of abc
- intermediate representation for pointcuts
- how to find shadows in code
- example: array access joinpoint
Shadow Kinds

- **BodyShadow**
  - ClassInitializationShadow
  - ExecutionShadow
  - InterfaceInitializationShadow
  - PreinitializationShadow

  *for blocks of Jimple code*

- ** StmtShadow**
  - ConstructorCallShadow
  - GetFieldShadow
  - HandlerShadow
  - MethodCallShadow
  - SetFieldShadow

  *for single Jimple statements (or a pair, for constructor call)*

- **Shadow**
  - *individual shadow instances*

A singleton class for each kind named `<kind>ShadowType` (a subtype of ShadowType) with a method

  `Shadow matchesAt(pos)`

Each new joinpoint requires a new Shadow and ShadowType class.
Pointcuts and weaving in *abc*

Java code

*Jimple*

shadows

AspectJ pointcuts

*intermediate pointcuts*

(in AspectInfo)

matcher

advice lists

weaver

Jimple
for each weavable class C
  for each method M in C
    for each “position” pos in M
      for each shadow type t
        Shadow sh = t.matchesAt(pos);
        for each advice declaration ad
          Pointcut pc = ad.getPointcut();
          Residue r = pc.matchesAt(sh);
          add (pos,ad,r) to “advice list” of M;

“positions” are user-definable

Residue is an IR of the dynamic test to be inserted (e.g. for if, args, ...)

advice lists are applied in a separate weaving pass
How to add a new joinpoint+pointcut?

- architecture of abc
- intermediate representation for pointcuts
- how to find shadows in code
- example: array access joinpoint
Example: array pointcuts

Bar around(Bar array[], int index) :
  arrayget() && target(array) && args(index) {
    Bar value=proceed(array,index);
    return value;
  }

void around (Bar array[], Bar value, int index) :
  arrayset() && target(array) && args(value, index) {
    proceed(array,value,index);
  }

general syntax:

basic_pointcut_expr ::=  
  PC_ARRAYGET LPAREN RPAREN  
  |  PC_ARRAYSET LPAREN RPAREN

with thanks to Bruno Harbulot!
## Six steps for adding new joinpoint

### General
- Extend parser, new AST nodes
- AspectInfo: IR for aspect-specific features
- Extend shadow finder
- New runtime
- Extend driver classes to use new runtime

### This example
- Grammar rule, new pointcut AST class
- New class for `arrayget` pointcut
- How to find `arrayget` shadow
- Dynamic representation of `arrayget` joinpoint
- AbcExtension
Finding *arrayget* shadows in Jimple

Java:

```java
void shift(Object[] arr) {
    for (int i = arr.length; i>0; i--)
        arr[i] = arr[i-1];
}
```

Jimple:

```jimple
void shift(java.lang.Object[])
{
    java.lang.Object[] arr;
    int i, $i0;
    java.lang.Object $r0;
    arr := @parameter0: java.lang.Object[];
    i = lengthof arr;
    goto label1;
label0:
    $i0 = i - 1;
    $r0 = arr[$i0];
    arr[i] = $r0;
    i = i - 1;
label1:
    if i > 0 goto label0;
    return;
}
```

Jimple is:
- typed
- stackless
public static ArrayGetShadow matchesAt(MethodPosition pos)
{
    if (!(pos instanceof StmtMethodPosition)) return null;
    Stmt stmt = ((StmtMethodPosition) pos).getStmt();

    if (!(stmt instanceof AssignStmt)) return null;
    AssignStmt assign = (AssignStmt) stmt;
    Value rhs = assign.getRightOp();

    if (!(rhs instanceof ArrayRef)) return null;
    ArrayRef ref = (ArrayRef) rhs;

    Value index = ref.getIndex();
    ... restructure if necessary, next slide ....

    return new ArrayGetShadow(pos.getContainer(), stmt);
}
Finding `arrayget` shadows (2)

// make sure the index is a local.
// restructure if necessary.
if (!(index instanceof Local)) {
    Body body=pos.getContainer().getActiveBody();
    Chain statements=body.getUnits().getNonPatchingChain();
    LocalGeneratorEx lg=new LocalGeneratorEx(body);

    Local l=lg.generateLocal(index.getType());
    AssignStmt as=Jimple.v().newAssignStmt(l, index);

    statements.insertBefore(as, stmt);
    stmt.redirectJumpsToThisTo(as);

    ref.setIndex(l);
}

label: $r0 = arr[0]$
⇒
int $i3$; ...
label': $i3 = 0$
label: $r0 = arr[$i3];
Goals of abc revisited

a compiler workbench for AspectJ to:

- explore AOP language design space (this talk)
- experiment with better code generation
- experiment with static analyses for safety checks and optimisations

Other goals of abc addressed in PLDI 2005
PLDI 2005 results: runtime speed

compile-time speed is *not* a goal of abc
Papers by users of abc


Tomoyuki Aotani and Hidehiko Masuhara: Compiling conditional pointcuts for user-level semantic pointcuts. SPLAT 2005.

Eric Bodden: Concern-specific languages and their implementation with abc. SPLAT 2005.