Towards Type Safety of Aspect-Oriented Languages

by

Florian Kammüller & Matthias Vösgen
Outline

- Introduction
- Featherweight Java and formalization
- Formalization of aspects
- Formalization of weaving
- AO type soundness
- Future Work
Theorem provers

How do theorem provers work?

- Automatic or human-aided term-rewriting

What are the applications?

- Proofs over complex structures (like prog. languages)
- Extraction of verified programs

Theorem provers and type-safety

Project Bali: Verification of the Java specification
using the prover Isabelle.
Popular theorem provers

- **Isabelle**
  - Classical logic
  - Extensive libraries
  - User friendly

- **Coq**
  - Constructive logic
  - Few libraries

- **PVS**

- **ACL 2**

- **HOL 4**

- **TWELF**

(...)

F. Kammüller & M. Vösgen

Towards Type Safety of Aspect-Oriented Languages
Why did we choose Coq?

Coq is a constructive theorem prover

Constructive proofs can be interpreted as algorithms (Curry-Howard Isomorphism)

-> Coq can extract code from proofs

-> We can extract a typechecker out of a proof for type safety
Definitions of type soundness

Natural language definition:
“Well Typed terms never get stuck.”

Formal definition: Progress & Preservation

Progress: Well-typed terms can be evaluated or they are values.

Preservation: The evaluation of a well-typed term leads to a another well-typed term.
Outline

- Introduction
- Featherweight Java and formalization
- Formalization of aspects
- Formalization of weaving
- AO type soundness
- Future Work
Featherweight Java

Java reduced to:

• Object creation
• Method invocation
• Field access
• Casting
• Variables

“Inside every large language is a small language struggling to get out.”
Properties of Featherweight Java

- Inheritance is part of the language
- Strictly formalized type system
- Very compact
- Quasi-functional language
- Nominal type system
- $\lambda$-calculus can be implemented in it
class Pair extends Object {
    Object fst;
    Object snd;

    Pair(Object fst, Object snd) {
        super(); this.fst = fst; this.snd = snd; }

    Pair setfst(Object newfst) {
        return new Pair(newfst, this.snd); }
}
Coq-FJ-Formalization by Stephanie Weirich

- Nearly complete formalization of FJ in Coq
- Type soundness proofs were made
- Clear top-down structure

Suitable foundation for extensions
Type-soundness in FJ

Coq-Code for progress and preservation

Lemma type_soundness :
for all CT: classTable e:expression e':expression,
  class_table_typing CT // All classes well typed
  -> multi_step CT e e' // Reduction from e to e' ex.
  -> ~(exists e'', reduction CT e' e'') // No reduction from e' ex.
  -> (value e' \∨ failed_cast CT e'). // e' is a value or a failed cast
Outline

- Introduction
- Featherweight Java and formalization
- Formalization of aspects
- Formalization of weaving
- AO type soundness
- Future Work
Formalization of AO

Weaving

Aspect

Advice

Advice Expression

Pointcut

Pointcut Selection

Class

(...)
Aspects

Coq-Code:
Inductive aspectDef : Set :=
  | Aspect : aspectName -> aspectName -> list fieldDef ->
    methodTable -> pointcutTable -> adviceTable -> aspectDef.
Pointcuts

**Execution Pointcut**

- **Name**
- **{Selections}**

---

Coq-Code:

```coq
Inductive pointcutDef : Set :=
    | Execution : pointcutName -> pointcutSelectionList
    -> pointcutDef.
```
Pointcut Selections

Pointcut Selection

Classname  Methodname

Coq-Code:

Inductive pointcutSelection : Set :=
  | methodSel: className -> methodName ->
    pointcutSelection.
Advice

Inductive adviceDef : Set :=
  | aroundAdvice : pointcutName -> adviceExp -> adviceDef.
Advice Expression

They are method expressions including a proceed statement

Coq-Code:

Inductive adviceExp : Set :=
| proceed: adviceExp
| adVar : varName -> adviceExp
| adFieldProj : adviceExp -> fieldName -> adviceExp
| adMethodInvk : adviceExp -> methodName ->
  list adviceExp -> adviceExp
| adNew : className -> list adviceExp -> adviceExp
| adCast : className -> adviceExp -> adviceExp.
Outline

- Introduction
- Featherweight Java and formalization
- Formalization of aspects
- Formalization of weaving
- AO type soundness
- Future Work
Weaving, top-level

An aspect-Table is weaved into a class-Table

Coq-Code:

Definition wv_AT_CT (CT: classTable) (AT: aspectTable) :
    classTable :=
        MapCollect _ _ (fun _ asp => wv_asp_CT CT asp) AT.
Weaving, bottom level

An advice expression is weaved into a method expression

Coq-Code:

Fixpoint merge_expr (mExpr: exp) (aExpr: adviceExp) {struct aExpr}: exp :=
  match aExpr with
  proceed                  => mExpr
  | adVar       v           => Var v
  | adFieldProj  aExpr2 fieldN => FieldProj (merge_expr mExpr aExpr2) fieldN
  (...)                     end.
Outline

- Introduction
- Featherweight Java and formalization
- Formalization of aspects
- Formalization of weaving
- AO type soundness
- Future Work
Type soundness (1)

Is an aspect table well typed?

Parameter \text{asp\_table\_typing}: \text{aspectTable} \rightarrow \text{Prop}.

A well typed aspect table weaves a well typed class table

Axiom \text{type\_soundness\_woven}:
\forall \text{AT:aspectTable} \ (\text{CT:classTable}), \ 
\text{class\_table\_typing \ CT} \rightarrow \text{asp\_table\_typing \ AT} \\
\rightarrow \text{class\_table\_typing \ (wv\_AT\_CT \ CT \ AT)}.
Type soundness (2)

Progress and Preservation with AO:

Lemma weave_type_soundness:
forall (CT0 CT: classTable)(e e': exp)(AT: aspectTable),
   CT = wv_AT_CT CT0 AT
   -> class_table_typing CT0
   -> asp_table_typing AT
   -> multi_step CT e e'
   -> ~(exists e'', reduction CT e' e'')
   -> (value e' \ value e' ∨ failed_cast CT e').
Outline

• Introduction
• Featherweight Java and formalization
• Formalization of aspects
• Formalization of weaving
• AO type soundness
• Future Work
Future work

There is a lot to do

- Completion of the formalization
- Proof type soundness, confinement etc.
- Investigate the runtime weaving problem
Thanks for listening!