A Mechanized Framework for Aspects in Isabelle/HOL

Florian Kammüller and Henry Sudhof

Institut für Softwaretechnik und Theoretische Informatik

WMM, 4 October 2007

Motivation and Background

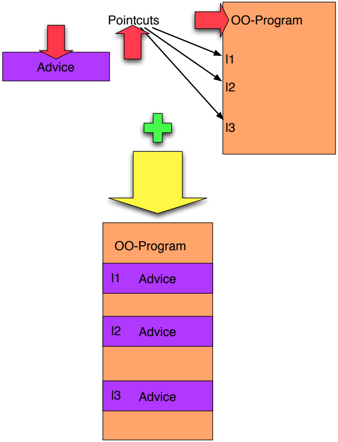
$\Rightarrow$ Verification of object-oriented paradigms
  - Aspect-oriented programming (AOP)
  - Distributed objects (ASP)
- Isabelle/HOL
  - Isabelle: generic interactive theorem prover
  - Embedding: types, constants, and definitions constitute *object logic* (theory)
  - Isabelle/HOL: instance for classical HOL
  - Many applications for programming language semantics, e.g. Java,
  - also specification languages: CSP, TLA, Object-Z, . . .

Overview

1. Aspect-Orientation
2. The Theory of Objects in Isabelle/HOL
3. A Theory of Aspects
4. Discussion

Aspect-Oriented Programing (AOP)

- Idea: Weave Advice into OO-Program
- Advice = code fragments
- Pointcuts: points at which advice is woven in
- Aspect = Advice + Pointcut-definition
- Weave produces combination
- AOP-language: base language (programs and advice)+ Pointcut definition language



AOP-Constructs

- Pointcut selection
  - `call`: syntactic selection of method calls
    e.g., *all methods whose name contains “set”*
  - `cflow`: selection of control flow points
    e.g., *from entry to exit of method x*
- Advice insertion
  - `before`, `after`
  - `around`: instead of selected command,
  - or `around` with `proceed`: before/after original command

⇒ Change of semantics

⇒ Endangers properties of programs

Foundations of AOP

- Formalization of AOP in Isabelle/HOL
- Idea: simple, functional calculus
- Represent pointcuts by *labels*, e.g.

$$\langle L, \lambda\, x\,.\, e\rangle \Downarrow v_1 + l_1(v_2) \xrightarrow{l_1 \in L} v_1 + e[v_2/x]$$

  with poincuts L , advice $\lambda\, x\,.\, e$, and weaving operator $\Downarrow$
- Based on object calculus (Theory of Objects ς)
- Advanced features: type preserving compilation

Theory of Objects: ς-calculus

- Terms in the ς-calculus: "labelled lists" of methods/fields
  - Objects: $[l_1 = \varsigma(x_0)b_0, \ldots, l_n = \varsigma(x_n)b_n]$ where x_j "self"-parameter
  - Method call/ field selection: $a.l_j$ where $j \in 1..n$
  - Update of method/field: $a.l_j := \varsigma(x)b$ where $j \in 1..n$
- Semantics: reduction relation $\rightarrow_\beta$
- Substitution of formal parameter with a it"self"

$$a \equiv [l_j = \varsigma(x_j)b_j]^{j \in 1..n}$$

$$a.l_j \rightarrow_\beta b_j[a/x_j] \qquad\qquad j \in 1..n$$

First step: ς-calculus in Isabelle/HOL

- Formalization of finite maps `L ⇀ T`
- Simple datatype for (de Bruijn) object terms

```
datatype term =
    Var nat
  | Obj Label ⇀ term
  | Call term Label
  | Upd term Label term
```

- Definition of substitution on de Bruijn terms `t [s / k]`
- Reduction relation $\to_\beta$

```
inductive beta
intros
  beta: l ∈ dom f ⟹ Call (Obj f) l →β the(f l)[(Obj f)/0]
  upd : l ∈ dom f ⟹ Upd (Obj f) l a →β Obj (f (l↦a))
  sel : s →βt ⟹ Call s l →β Call t l
  updL: s →βt ⟹ Upd s l u →β Upd t l u
  updR: s →βt ⟹ Upd u l s →β Upd u l t
  obj : [s →βt; l ∈ dom f]⟹ Obj(f(l↦s)) →β Obj(f(l↦t))
```

Confluence and Type Safety for ς-calculus

- Confluence (diamond property)

$$\begin{tikzcd} & M \arrow[dl] \arrow[dr] & \\ N_0 \arrow[dr] & & N_1 \arrow[dl] \\ & L & \end{tikzcd}$$

  - If a term M can be reduced in $n \geq 0$ reduction steps to terms N_0 and N_1, then there exists L such that N_0 and N_1 can be reduced to L.

- We define simple type system for ς-calculus,

$$\mathtt{E} \vdash \mathtt{t} : \mathtt{T}$$

  i.e., *term $\mathtt{t}$ has type $\mathtt{T}$ in type environment $\mathtt{E}$*

- We prove type safety for first-order type system of ς

Theorem (preservation)

$$[|\ \mathtt{t} \rightarrow_{\beta}^{*} \mathtt{t'};\ \mathtt{E} \vdash \mathtt{t} : \mathtt{T}\ |] \Longrightarrow \mathtt{E} \vdash \mathtt{t'} : \mathtt{T}$$

Theorem (progress)

$$[|\ [] \vdash \mathtt{t} : \mathtt{A};\ \nexists\ \mathtt{c}\ .\ \mathtt{t} = \mathtt{Obj}\ \mathtt{c}\ |] \Longrightarrow \exists\ \mathtt{b}\ .\ \mathtt{t} \rightarrow_{\beta} \mathtt{b}$$

Aspects

- Extend terms t by (aspect-)labelled terms, e.g. $l\langle t\rangle$

```
datatype term = Var nat
     | Obj label ⇀ term
     | Call term label
     | Upd term label term
     | Asp Label term ("_ ⟨_⟩")
```

- Aspect = ⟨ pointcut (*set of* `Labels`), advice (term function) ⟩

```
datatype aspect = Aspect (Label set) term ("⟨_, _⟩")
```

Weaving

- Idea of weaving: replace existing labels in program with advice

  `weave :: [ term, aspect ]` $\Rightarrow$ `term ("`$\Downarrow$`")`

- For example, central rule now:

  $\mathtt{l}\langle\mathtt{t}\rangle\Downarrow\mathtt{a} = \mathtt{if\ l} \in \mathtt{pct\ a\ then\ l}\langle\mathtt{adv\ a\ [t/0]}\rangle\ \mathtt{else\ l}\langle\mathtt{t}\rangle$

  where $\mathtt{pct}\ \langle\mathtt{L},\ \mathtt{a}\rangle = \mathtt{L}$ and $\mathtt{adv}\ \langle\mathtt{L},\ \mathtt{a}\rangle = \mathtt{a}$

Typing of Aspects

- Problem: AOP not type safe in general
- Example: around advice exchanges return value [Kammüller, Vösgen: FOAL06]
- Type system to exclude pathological cases:
  - Extend previous type relation by labels `L`

    $$\texttt{E, L} \vdash \texttt{t : T}$$

    i.e., *term* `t` *has type* `T` *in type/label environment* `E, L`
  - Idea: label types represent "legal" advice
- Define *well-formedness* of program `t` wrt set of aspects `A` (`wf t A`)
- Goal: prove that weaving preserves type relation.

Theorem

$$[\![\texttt{wf t A; [], L} \vdash \texttt{t : T}]\!] \Longrightarrow \texttt{[], L} \vdash \texttt{Weave t A: T}$$

Summary

- The ς-calculus as a Basis for AOP (and ASP) in Isabelle/HOL

[1] L. Henrio, F. Kammüller. *A Mechanized Model of the Theory of Objects.* FMOODs'07.

- Labels representing pointcuts in programs
- Definition of weaving function
- Typing of advice and labels :

$\Rightarrow$ type safety for aspects in Isabelle/HOL

Discussion

- Nominal Techniques *vs* HOAS *vs* de Bruijn
- Code extraction
- Structural vs Nominal Type Systems
- Is ς-calculus unrealistic (type preserving compilations)?