Additive Aspects and General References

Sam Sanjabi, Luke Ong and William Greenland

{sams,lo,willg}@comlab.ox.ac.uk.

Oxford University Computing Laboratory

Overview

- Aspect-Oriented Programming
- Formalisation
- A Fully Abstract Translation

Aspect-Oriented Programming

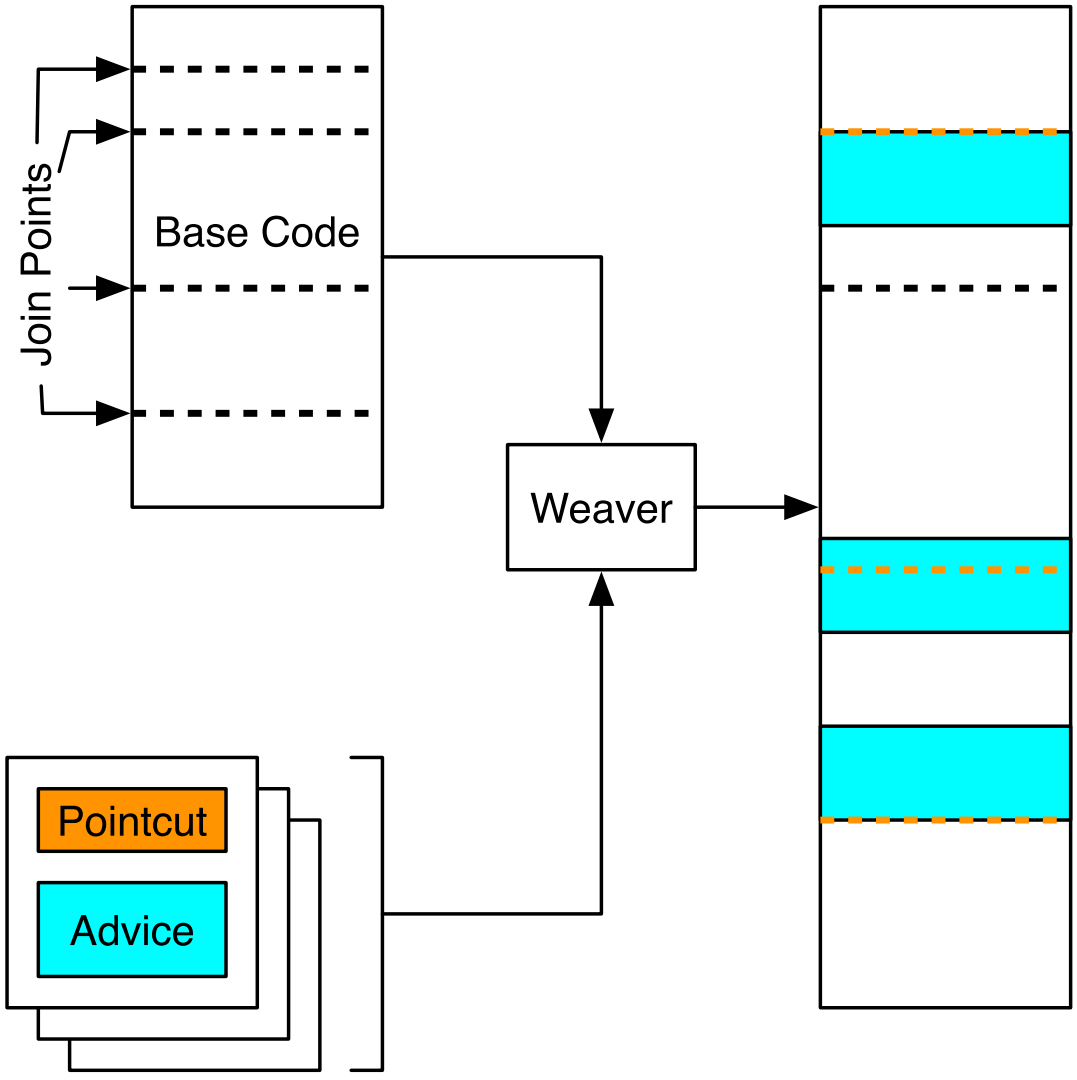
"Aspect-Oriented Programming can be understood as the desire to make quantified statements about the behaviour of programs, and to have these quantifications hold over programs written by oblivious programmers"

– Filman & Friedman (2000)

Modern aspect oriented programs consist of:

- A 'base' program with
- A distinguished subset of its program points (join points)
- A way to specify predicates (pointcuts) over join points
- A way to transfer control to an exeternal piece of code (advice) when the predicate matches at run-time
- An aspect encapsulates advice with a pointcut specifying when it should execute.

Components of an Aspect Oriented Language



Mini (Additive) Aspect ML

An Aspect-Oriented ML-like language. Join points are function declarations:

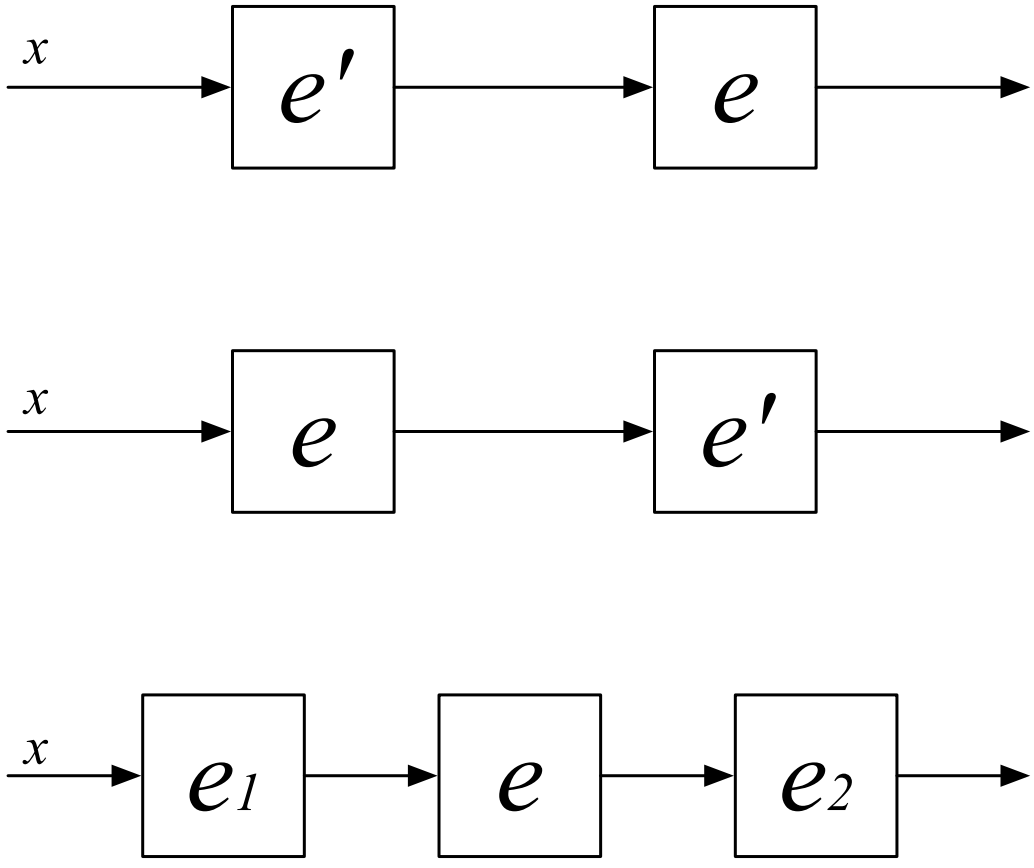
$$(\texttt{fun}\ \ f(x) = e)$$

Each such declaration creates two join points immediately before and after the execution of e. We can then declare the following kinds of advice:

$$\begin{aligned}
\texttt{before}\{f\}(y) &= e' \\
\texttt{after}\{f\}(y) &= e' \\
\texttt{around}\{f\}(y) &= e_1\ ;\ \texttt{proceed}\ \ z \rightarrow e_2
\end{aligned}$$

With non-additive aspects, `around` advice doesn't execute f at all unless `proceed` is called. Our treatment does not support this feature.

Effect of Additive Advice



A Worrying Example

```
let
  fun succ (x) = x + 1
  fun add2 (x) = succ (succ x)

in
  add2 7 (* result = 9 *)
end
```

A Worrying Example

```
let
  fun succ (x) = x + 1
  fun add2 (x) = succ (succ x)
  before {succ} (x) = x + 1 (* aspect *)
in
  add2 7 (* result = 11 *)
end
```

- We need to find a way to preserve the benefits of aspects, while limiting their undesirable properties
- We approach the problem by allowing semantic models of aspects to guide our design decisions
- This requires a suitable core calculus of aspects
- Walker *et al*. define the semantics of MinAML by translating it into such a calculus

Walker's Calculus: The Idea

Key features of CMAML (Core MinAML):

1. Simply typed call-by-value λ-calculus with products (including the empty product):

$$\theta ::= () \mid \texttt{bool} \mid \theta \times \theta \mid \theta \to \theta$$

2. A set of *labels* used to explicitly label join points

$$\theta ::= \quad \cdots \quad \mid \texttt{lab}[\theta]$$

3. Pointcuts are tests of label equality
4. Aspects encapsulate a pointcut and advice to be executed:

$$\texttt{asp}[\theta] = \texttt{lab}[\theta] \times (\theta \to \theta)$$

CMAML **(cont. . .)**

A term is evaluated in the context of a sequence A of aspects. The following constructs are added to the usual λ-calculus.

- $\texttt{newlab}_\theta$
  - Introduce a new label of type θ
  - Analogous to block allocated locations or exceptions in imperative programs.
- a « e and a » e
  - Add aspect $a = \langle \ell, \lambda x.e \rangle$ at the head or tail of the sequence A.
  - Evaluate e in the updated context.
- $\ell\langle e \rangle$
  - Mark the point after execution of e with label ℓ. To evaluate:
    1. evaluate e to value v
    2. compose all ℓ-labeled advice in A into a single λ term f
    3. return $f(v)$

Intuitive Examples

Supposing that we trivially extend CMAML with integer arithmetic...
The same label can be used at multiple program points:

$$\texttt{newlab}\ \ell : \mathbb{Z}\ \texttt{in}\ \{\ell.x \to x^2\} \ll (\ell\langle 3\rangle + \ell\langle 2\rangle) \Downarrow 13 = 3^2 + 2^2$$

Intuitive Examples

Supposing that we trivially extend CMAML with integer arithmetic...
The same label can be used at multiple program points:

$$\texttt{newlab}\ \ell : \mathbb{Z}\ \texttt{in}\ \{\ell.x \to x^2\} \mathbin{«} (\ell\langle 3\rangle + \ell\langle 2\rangle) \Downarrow 13 = 3^2 + 2^2$$

One program point can trigger multiple pieces of advice:

$$\texttt{newlab}\ \ell : \mathbb{Z}\ \texttt{in}\ \underbrace{\{\ell.x \to x+1\}}_{\texttt{inc}} \mathbin{«} \underbrace{\{\ell.y \to y*2\}}_{\texttt{dbl}} \mathbin{«} \ell\langle 3\rangle \Downarrow 7$$

Intuitive Examples

Supposing that we trivially extend CMAML with integer arithmetic...
The same label can be used at multiple program points:

$$\texttt{newlab}\ \ell : \mathbb{Z}\ \texttt{in}\ \{\ell.x \to x^2\} \mathbin{«} (\ell\langle 3\rangle + \ell\langle 2\rangle) \Downarrow 13 = 3^2 + 2^2$$

One program point can trigger multiple pieces of advice:

$$\texttt{newlab}\ \ell : \mathbb{Z}\ \texttt{in}\ \underbrace{\{\ell.x \to x+1\}}_{\texttt{inc}} \mathbin{«} \underbrace{\{\ell.y \to y*2\}}_{\texttt{dbl}} \mathbin{«} \ell\langle 3\rangle \Downarrow 7$$

Ordering of advice is significant:

$$\texttt{newlab}\ \ell : \mathbb{Z}\ \texttt{in}\ \{\ell.x \to x+1\} \mathbin{«} \{\ell.y \to y*2\} \mathbin{»} \ell\langle 3\rangle \Downarrow 8$$

Intuitive Examples

Supposing that we trivially extend CMAML with integer arithmetic...
The same label can be used at multiple program points:

$$\texttt{newlab}\ \ell : \mathbb{Z}\ \texttt{in}\ \{\ell.x \to x^2\} \mathbin{«} (\ell\langle 3\rangle + \ell\langle 2\rangle) \Downarrow 13 = 3^2 + 2^2$$

One program point can trigger multiple pieces of advice:

$$\texttt{newlab}\ \ell : \mathbb{Z}\ \texttt{in}\ \underbrace{\{\ell.x \to x+1\}}_{\texttt{inc}} \mathbin{«} \underbrace{\{\ell.y \to y*2\}}_{\texttt{dbl}} \mathbin{«} \ell\langle 3\rangle \Downarrow 7$$

Ordering of advice is significant:

$$\texttt{newlab}\ \ell : \mathbb{Z}\ \texttt{in}\ \{\ell.x \to x+1\} \mathbin{«} \{\ell.y \to y*2\} \mathbin{»} \ell\langle 3\rangle \Downarrow 8$$

Note: this language can encode general references as a pair $\langle !, :=\rangle$ by installing constant functions as advice (Walker *et al*. 2003)

Translations

- This encoding induces a translation from call-by-value Idealized Algol with general references (IAG) to CMAML.
- We propose an encoding $(\!|-|\!)$ in the other direction.
- If $(\!|-|\!)$ is proved fully abstract, we would inherit a full abstraction result for CMAML due to the extant game semantics model of IAG (Abramsky, Honda, McCusker 1998).

$$\begin{array}{ccccc} M \sim N & \Longleftrightarrow & (\!|M|\!) \simeq (\!|N|\!) & \Longleftrightarrow & [\![(\!|M|\!)]\!] = [\![(\!|N|\!)]\!] \\ \uparrow & & \uparrow & & \uparrow \\ \text{Equivalence} & & \text{Equivalence} & & \text{Semantic} \\ \text{in CMAML} & & \text{in IAG} & & \text{Equality} \end{array}$$

Where $[\![-]\!]$ gives the game semantics of IAG. This technique was introduced by McCusker in 1996.

Aspects to General References

- Model labels of type θ as *locations* that store functions of type $(\!|\theta \to \theta|\!)$:

$$\begin{array}{rcl} (\!|\texttt{lab}[\theta]|\!) & = & \texttt{var}[(\!|\theta \to \theta|\!)] \\ (\!|\texttt{newlab}_\theta|\!) & = & \texttt{new}_{(\!|\theta \to \theta|\!)}[\mathsf{id}_{(\!|\theta|\!)}] \end{array}$$

- Aspect installation composes advice to any function currently in the store location given by the aspects' label

$$\begin{array}{rcl} (\!|a \mathbin{\texttt{«}} e|\!) & = & [(\!|\texttt{fst}\ a|\!) := \texttt{!}(\!|\texttt{fst}\ a|\!) \circ (\!|\texttt{snd}\ a|\!)]\,;(\!|e|\!) \\ (\!|a \mathbin{\texttt{»}} e|\!) & = & [(\!|\texttt{fst}\ a|\!) := (\!|\texttt{snd}\ a|\!) \circ \texttt{!}(\!|\texttt{fst}\ a|\!)]\,;(\!|e|\!) \end{array}$$

- Aspect invocation is a combination of dereferencing and application

$$(\!|\ell\langle e\rangle|\!) = \texttt{!}(\!|\ell|\!)((\!|e|\!))$$

Results: Soundness

Lemma (Soundness)**.**

$$e \Downarrow v \quad \Rightarrow \quad (\!|e|\!) \Downarrow (\!|v|\!)$$

Lemma (Adequacy)**.**

If $(\!|e|\!) \Downarrow V$ *then* $V \stackrel{\alpha}{=} (\!|v|\!)$ *for some* CMAML *value* v*, and moreover* $e \Downarrow v$*.*

Lemma (Equational Soundness)**.**

$$(\!|e_1|\!) \simeq (\!|e_2|\!) \quad \Rightarrow \quad e_1 \sim e_2$$

Note: The strength of the adequacy result is surprising. It indicates that the correspondence between the aspect operators of CMAML and reference operators of IAG is very strong indeed!

Results: Completeness

Lemma (Definability)**.** *For any* IAG *term* M *containing no free locations, there exists a* CMAML *term* e *such that* $(\!|e|\!) \simeq M$

- Proved by structural induction on M
- Must add the `mkvar` constructor to CMAML (but no actual store!)
- Use a slight variant of Walker's technique for encoding general references to prove definability for `new`

Theorem (Full Abstraction)**.**

$$(\!|e_1|\!) \simeq (\!|e_2|\!) \quad \Leftrightarrow \quad e_1 \sim e_2$$

Implications

- Inheritance of fully abstract model of IAG
- Implies a strong notion of equivalence between the two languages
- Suggests a novel implementation strategy for additive aspects

Further research:

- Extend the translation to encompass non-additive aspects
- Applying algorithmic results on game semantics to aspects
  - Study a language allowing only k^{th} order functions of a particular type to be stored
  - How low does k have to be to preserve regularity?
- Call-by-name?
- Dynamic pointcuts?
- A more “satisfying” semantic account of aspects

CMAML: Types and Syntax

$$\Theta ::= () \mid \texttt{bool} \mid \Theta_1 \times \Theta_2 \mid \Theta_1 \to \Theta_2 \mid \texttt{lab}[\Theta]$$

$$\begin{array}{rcl} P, Q & ::= & x \mid \langle\rangle \mid \texttt{true} \mid \texttt{false} \mid \texttt{cond}\ P\ Q_1\ Q_1 \mid \\ & & \lambda x : \Theta.P \mid P\ Q \mid \langle P_1, P_2 \rangle \mid \texttt{fst}\ P \mid \texttt{snd}\ P \mid \\ & & \texttt{newlab}_\Theta \mid P\langle Q \rangle \mid P \raisebox{0.5ex}{\textbf{»}} Q \mid P \raisebox{0.5ex}{\textbf{«}} Q \end{array}$$

Desugar Walker *et al*.'s aspect construct $\{P.x \to Q\}$ as the pair

$$\langle P, \lambda x : \Theta.Q \rangle$$

i.e.,, we identify aspects with the pairing of their pointcut and their advice:

$$\texttt{asp}[\Theta] = \texttt{lab}[\Theta] \times (\Theta \to \Theta)$$

Typing Rules

To the usual rules for the call-by-value λ-calculus, we add

$$\frac{}{\Gamma \vdash \mathtt{newlab}_\Theta : \mathtt{lab}[\Theta]} \qquad \frac{\Gamma \vdash P : \mathtt{lab}[\Theta] \qquad \Gamma \vdash Q : \Theta}{\Gamma \vdash P\langle Q \rangle : \Theta}$$

$$\frac{\Gamma \vdash P : \mathtt{asp}[\hat{\Theta}] \qquad \Gamma \vdash Q : \Theta}{\Gamma \vdash P \mathbin{«} Q : \Theta} \qquad \frac{\Gamma \vdash P : \mathtt{asp}[\hat{\Theta}] \qquad \Gamma \vdash Q : \Theta}{\Gamma \vdash P \mathbin{»} Q : \Theta}$$

- The P « Q and P » Q expressions evaluate Q in an environment in which the aspect (i.e., pair) P is “installed”
- The direction of the arrows prioritize the aspects

Operational Semantics

Assuming a countably many distinct labels ℓ, the big-step semantics are defined as an evaluation relation between triples $\langle L, A, P\rangle$:

- L a mapping from labels to types
- A a sequence of aspects
- P a CMAML term

$$\frac{P \Downarrow \langle L, A, a\rangle \qquad \langle L, (a, A), Q\rangle \Downarrow U}{P \mathbin{«} Q \Downarrow U} \qquad \frac{P \Downarrow \langle L, A, a\rangle \qquad \langle L, (A, a), Q\rangle \Downarrow U}{P \mathbin{»} Q \Downarrow U}$$

$$\frac{P \Downarrow \ell \qquad \vdash \langle\langle \ell \rangle\rangle = F \qquad F(Q) \Downarrow U}{P\langle Q\rangle \Downarrow U} \qquad \frac{}{\langle L, A, \mathtt{newlab}_\Theta\rangle \Downarrow \langle L \cup \{\ell : \Theta\}, A, \ell\rangle}$$

Contextual components L, A have been omitted where possible. F is the functional composition of the ℓ-labelled advice in A, the order of composition is given by the ordering of the sequence.